UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AMAZON.COM, INC. and AMAZON.COM SERVICES LLC,
Petitioner

v.

NOKIA TECHNOLOGIES OY,
Patent Owner.

IPR2024-01141
U.S. Patent No. 8,036,273

DECLARATION OF DR. KANNAN RAMCHANDRAN

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Date: 07/05/2024 By: 

Kannan Ramchandran, Ph.D.

TABLE OF CONTENTS

Page

I. INTRODUCTION 1
II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE 1
III. MATERIALS CONSIDERED 4
IV. LEGAL PRINCIPLES 4
V. TECHNICAL BACKGROUND 7
VI. OVERVIEW OF THE '273 PATENT 18
A. The Alleged Problem in the Prior Art 18
B. The Alleged Invention of the '273 Patent 25
VII. LEVEL OF ORDINARY SKILL 28
VIII. PRIOR ART OVERVIEW 29
A. TML6 29
1. Motion Prediction During Encoding 39
2. Interpolating Sub-Pixels During Encoding 42
3. Motion Prediction During Decoding 119
4. TML6 Decoding Interpolation 121
B. Fandrianto 123
IX. DETAILED EXPLANATION OF GROUNDS 138
A. Ground 1: Claims 1, 4-17, 19-26, 28-29, 31, 33, 36-49, 51, 53-59, 61-62, 64-78, and 82-83 Are Obvious over TML6 in View of Fandrianto. 139
1. Motivation to Combine 139
2. Claims 1, 4-17, 19-26, and 28-29 160
3. Independent Claim 31 299
4. Claims 33, 36-49, 51, 53-59, and 61-62 301
5. Claims 64-67 312
6. Claims 68-70 322
7. Claims 71-72 327
8. Claims 73-74 331

9. Independent Claim 75:.....333
10. Claims 76-78.....402
11. Independent Claim 82.....405
12. Independent Claim 83.....407
B. Ground 2: Claims 32, 52, and 79-81 are Obvious over TML6 in View of Fandrianto and the Knowledge of a POSITA.....410
1. Independent Claim 32:.....410
2. Dependent Claim 52: "An interpolator according to claim 33, wherein N is set equal to 3.....414
3. Claims 79-81.....414
X. AVAILABILITY FOR CROSS EXAMINATION.....417
XI. SUPPLEMENTATION OF CONCLUSIONS.....418

I, Kannan Ramchandran, declare as follows:

I. INTRODUCTION

1. My name is Kannan Ramchandran. I am a Professor of Electrical Engineering and Computer Science at the University of California Berkeley.

2. I have been retained as an expert in this proceeding by counsel for Amazon.com, Inc. and Amazon.com Services LLC ("Amazon" or "Petitioner"). I have been asked to provide my expert opinions regarding the validity of claims 1, 4-17, 19-26, 28-29, 31, 33, 36-49, 51, 53-59, 61-62, 64-72, 75-78, and 82-83 of United States Patent No. 8,036,273 (the "'273 patent," EX-1001).

II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE

3. My qualifications are stated more fully in my curriculum vitae, which is attached as Appendix A. Below is a summary of my education, work experience, and other qualifications.

4. I received my M.S. in electrical engineering in 1984 and my Ph.D. in electrical engineering in 1993, both from Columbia University. After receiving my undergraduate degree and while studying for my Ph.D., I served on the Technical Staff of AT&T Bell Labs as an R&D Design Engineer, from 1984-1990.

5. I have been an Assistant Professor, or Professor, of electrical engineering since 1993. First, I was an Assistant Professor in the Electrical & Computer Engineering Department at the University of Illinois at Urbana-

Champaign ("UIUC") from 1993-1999. While at UIUC, I taught undergraduate and graduate classes in, among other subjects, digital signal and image processing. I also served as Research Assistant Professor at the Beckman Institute and Coordinated Science Laboratory, and as the Associate Director of Multimedia Laboratory at the Beckman Institute.

6. I have been a Professor of Electrical Engineering and Computer Science at UC Berkeley since 1999. At UC Berkeley, I teach undergraduate and graduate-level classes in, among other subjects, digital signal and image processing and digital communication systems. I am also the Director of the Berkeley Audiovisual Signal Processing and Communication Systems ("BASiCS") Research Group, and Co-Director of the Wireless Foundations Research Center.

7. I have been a Fellow of the Institute of Electrical and Electronics Engineers ("IEEE") since 2005. I have also served as a technical consultant in media compression and networking for over a decade, consulting for both established entities like Microsoft, Hewlett-Packard, Intel, Texas Instruments, Lucent Technologies, and AT&T Research Laboratories, as well as smaller ventures such as PictureTel Corporation, Aware Inc., and numerous media-related startups in Silicon Valley.

8. My primary research focus has been in the field of video compression and multimedia networking over wired and wireless networks. I hold several

patents in the field of image and video compression and transmission, and my work on rate-distortion optimization in image and video compression has contributed to the evolution of video compression standards such as the H.264 standard. Further, a modified version of one of my patents on multi-resolution video broadcast is currently used in the Digital Audio Broadcast standard in Europe.

9. I have published over 650 refereed journal and conference publications, including many articles related to image and video compression and transmission. I have also given several keynote and invited talks at conferences in the fields of image processing, multimedia signal processing, wireless sensor networks and information theory.

10. I have received a number of research accolades in the field of video compression, including two Best Paper Awards from the IEEE Signal Processing Society, and several conference best paper awards. Other awards include an IEEE Information Theory Society and Communication Society Joint Best Paper award for 2012, an IEEE Communication Society Data Storage Best Paper award in 2010, an Okawa Foundation Prize for outstanding research at Berkeley in 2001, and an Outstanding Teaching Award at Berkeley in 2009, and a Hank Magnuski Scholar award at Illinois in 1998. I have received several grant awards from the National Science Foundation and the Department of Defense, in which I have been

the primary investigator on multi-year multi-million dollar projects related to media transmission and compression.

III. MATERIALS CONSIDERED

11. In preparing this declaration, I have reviewed the claims, specification, and file history of the ’273 patent. I understand that the ’273 patent issued on October 11, 2011 from U.S. Patent Application No. 11/839,205 (filed August 15,2007). The ’273 patent claims priority to U.S. Patent Application No. 09/954,608, which was filed on September 17, 2001.

12. I have reviewed and understand the references and exhibits cited in this Declaration.

IV. LEGAL PRINCIPLES

13. I am not an attorney. I have been informed, for the purposes of this declaration, about certain aspects of the law that are relevant to my opinions. I have applied these legal principles in arriving at my opinions expressed in this declaration.

14. I have been informed and understand that a patent claim is invalid if it would have been obvious to a person of ordinary skill in the art (“POSITA”) at the time of the alleged invention. This means that, even if all of the requirements of a claim are not found in a single prior art reference, the claim is unpatentable if the

differences between the subject matter in the prior art and the subject matter in the claim would have been obvious to a POSITA at the time of the alleged invention.

15. I have been informed and understand that determining whether a claim would have been obvious to a POSITA at the time of the alleged invention is based upon several factors, including, among others: (1) the level of ordinary skill in the art at the time of the alleged invention; (2) the scope and content of the prior art; (3) the differences that existed between the claimed invention and the prior art, if any; and (4) any objective considerations of non-obviousness.

16. I have been informed and understand that, when determining any differences between the claimed invention and the prior art, one should consider the claimed invention as a whole and determine whether or not it would have been obvious in light of all of the prior art, rather than look at the individual differences between the patent claims and the prior art in isolation.

17. I have been informed and understand that the teachings of two or more references may be combined in the same way as disclosed in the claims. I have been informed and understand that a patent composed of several elements is not proved obvious merely by demonstrating that each of its elements was independently known in the prior art. I have been informed that, instead, it can be important to identify a teaching, suggestion, or motivation, in either the prior art or the knowledge of persons skilled in the art, that would have prompted a person of

ordinary skill in the relevant field to combine the elements in the prior art in the way the claimed invention does.

18. In determining whether a combination based on multiple references would have been obvious, it is appropriate to consider, among other factors:

- whether the teachings of the prior art references disclose known concepts combined in familiar ways, and when combined, would yield predictable results;
- whether a POSITA would implement a predictable variation, and would see the benefit of doing so;
- whether the claimed elements represent one of a limited number of known design choices, and would have a reasonable expectation of success by a POSITA;
- whether a POSITA would have recognized a reason to combine known elements in the manner described in the claim;
- whether there is some teaching or suggestion in the prior art to render the modification or combination of elements claimed in the patent; and
- whether the innovation applies a known technique that had been used to improve a similar device or method in a similar way.

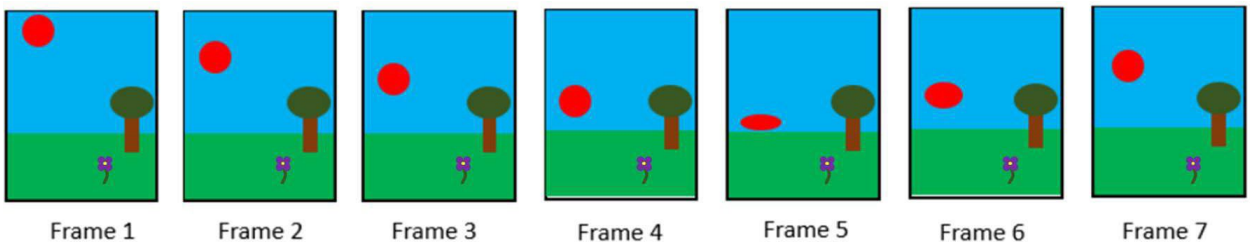
19. I have been informed and understand that one of ordinary skill in the art has ordinary creativity. I have also been informed and understand that, in

considering obviousness, it is important not to use hindsight derived from the patent being considered.

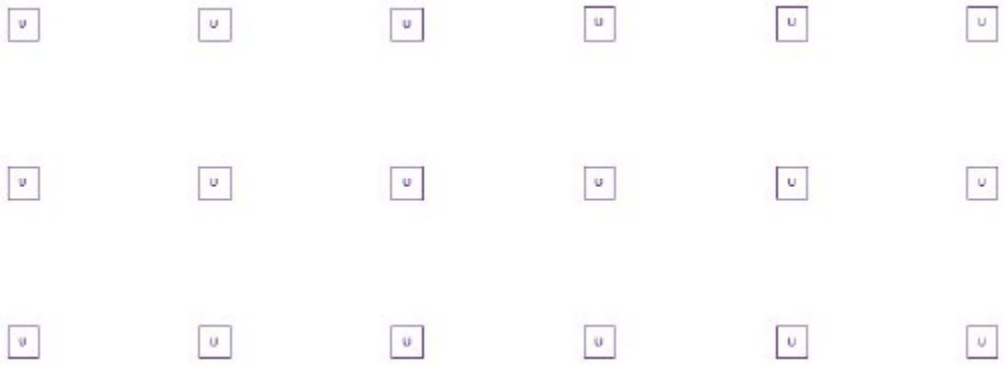
20. I have been informed and understand that, to show a motivation to combine, one is not required to show how the references would be physically combined to render a claim obvious. The analysis is focused on whether a POSITA would have been motivated to combine the teachings of the references to achieve the claimed invention, and not on whether the features of one reference can be physically incorporated into the structure of another reference. For example, a POSITA would have understood how predictable variations of known concepts could be combined and would recognize that where multiple tools were available in a toolbox, any could be chosen.

V. TECHNICAL BACKGROUND

21. Videos are made up of a series of still images called "frames." Playing these frames in rapid succession, such as at 60 frames-per-second, gives viewers the impression of motion across the video. For example, below is an illustration of seven still frames of a video. Each frame shows a **red** ball at a different location in the image, and when displayed sequentially, these seven frames will give the impression that the ball is bouncing as it moves down in Frames 1-4, hits the ground in Frame 5, and comes back up again in Frames 6-7.



22. Each frame in the video is broken up into many "picture elements," or "pixels," which are arranged in rows and columns. For example, the below pixels "U" in **purple** could represent a portion of the small **purple** flower in the frames, above.

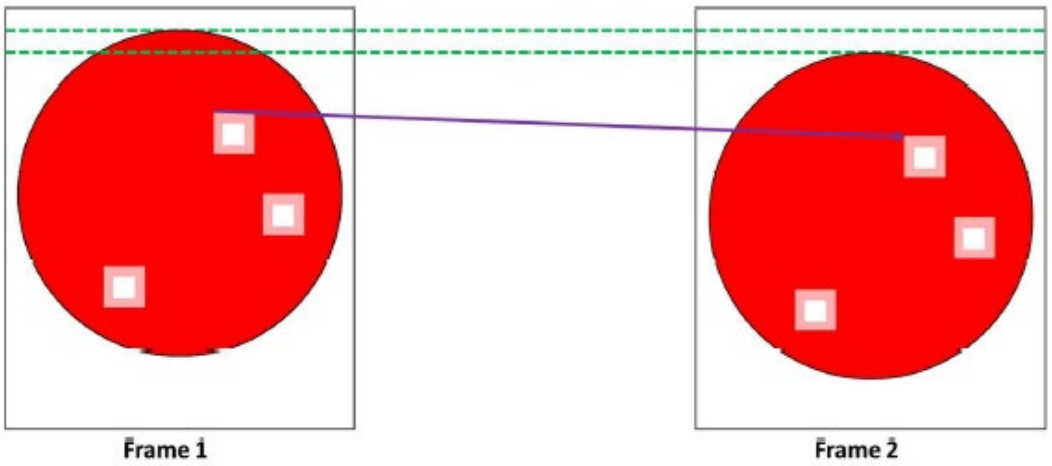


Each pixel is defined by data representing its brightness and color. High-resolution frames each have millions of pixels. Therefore, the amount of pixel data needed to represent even a single frame is substantial, and grows with each additional frame in the video.

23. Long before the '273 patent video compression techniques have been used to reduce the amount of data needed to represent the frames. Some unique properties of videos are used by these compression techniques to reduce the amount of data. For example, many of the pixels in one frame are often exactly the

same (or at least very similar) in successive frames. This is illustrated in the seven frames shown above: pixels representing objects that appear to be stationary in the frames above (for example, the grass, tree, the flower, and much of the sky) do not change from one frame to the next. As a result, one way to reduce the amount of data representing the video signal is to transmit data representing a stationary item once for an initial frame, and then to reuse that same data in subsequent frames in which the item has not moved.

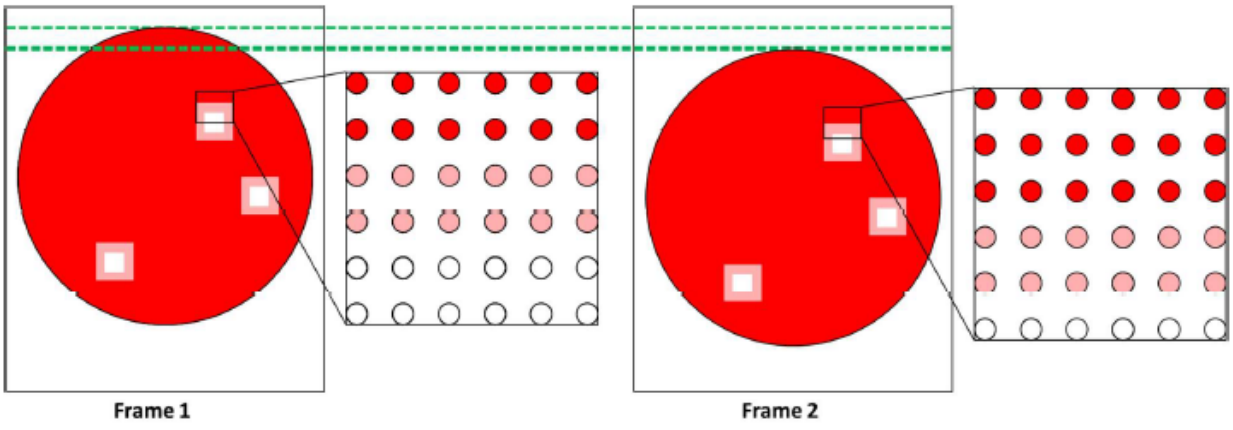
24. Likewise, some objects in the frame (for example, the **red** ball illustrated above) may move to a different location from one frame to the next but do not otherwise change. The frames below illustrate this: the pixel data representing a portion of the **red** ball is the same in the two frames, but the location of the ball is different:



As illustrated, it has moved down one pixel, as illustrated by the difference in the two **green** dashed lines. Thus, in order to avoid retransmitting the same pixel data

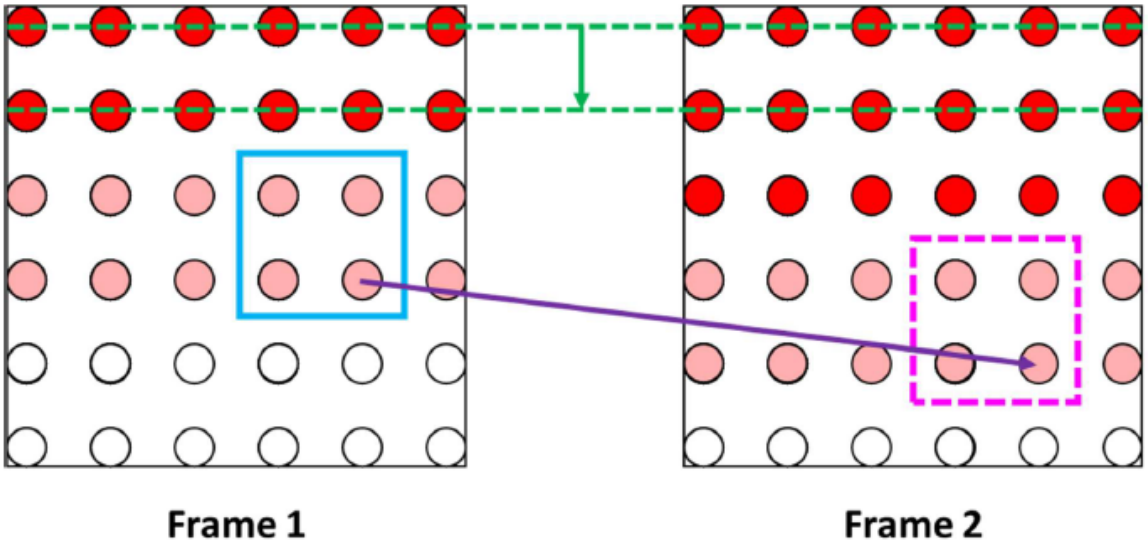
for the ball multiple times, the data can be transmitted once for Frame 1, and then a "motion vector" (represented by the **purple** arrow above) can be transmitted for Frame 2. The motion vector identifies where the pixels representing the ball have moved from Frame 1 to Frame 2, but requires a much smaller amount of data. Similarly, if the location of the pixels representing the ball changes in subsequent frames, additional motion vectors are transmitted to indicate the changes in location. As a result, the data for the pixels representing the ball need not be retransmitted as it moves from one frame to the next.

25. A motion vector may be determined by identifying pixels in one frame that match pixels in a subsequent or preceding frame. The figure below, for example, shows some of the "pixel locations"—circles outlined in black—in Frames 1 and 2 of the video signal:

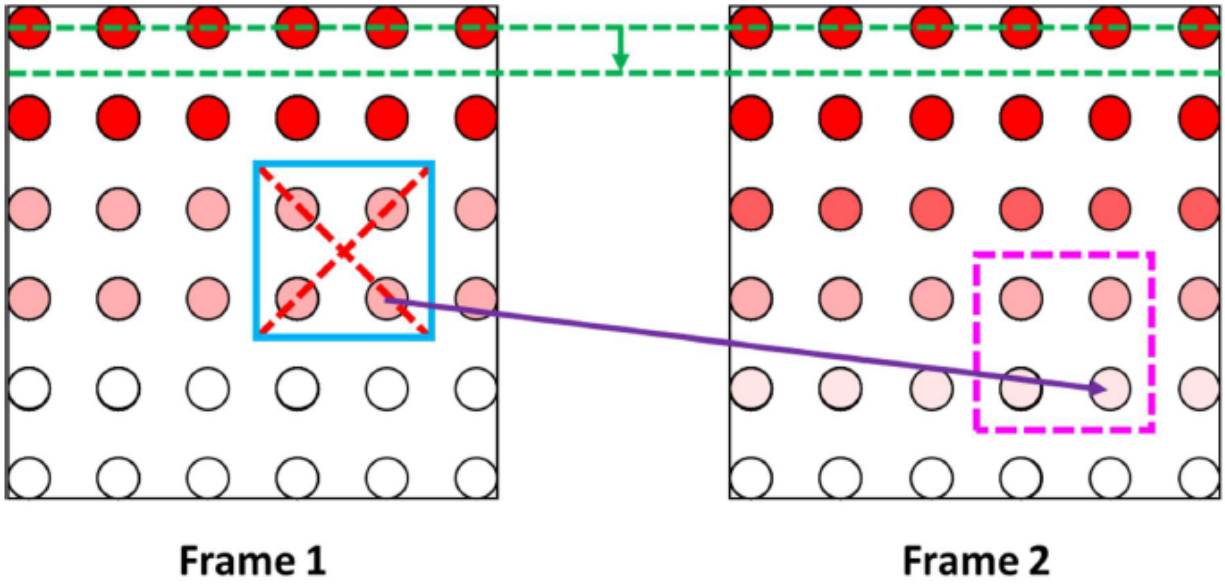


Because the ball has moved down one pixel, the pixel data illustrated in the black box is also shown as moving down one pixel. As illustrated below, a 2x2 group of

pixels within the **blue** box that represents a portion of the ball in Frame 1 matches another 2x2 group of pixels within the **pink** box that represents the ball in Frame 2. Accordingly, the movement of the 2x2 group of pixels from the pixel locations in Frame 1 to the pixel locations in Frame 2 can represented by the **purple** motion vector:



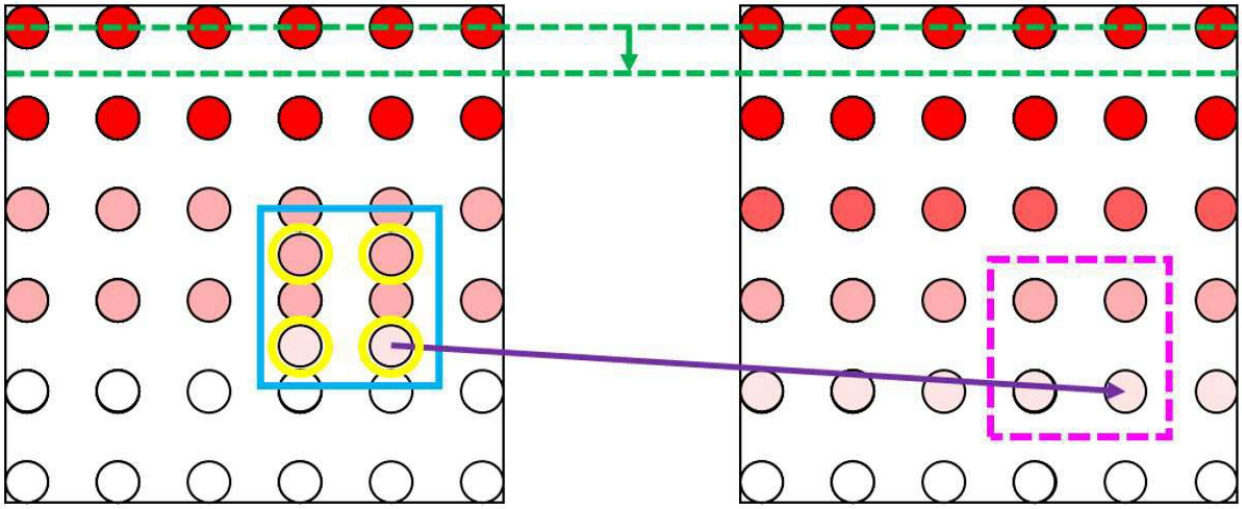
26. But objects represented in a video do not always move in "pixel-length" increments, and can also move distances that do not precisely align with actual pixel locations. That is, they may more accurately be described as moving a distance that is a fraction of the distance between the displayed pixels. This is illustrated in the figure below, where the ball instead moves down by a fraction of a pixel:



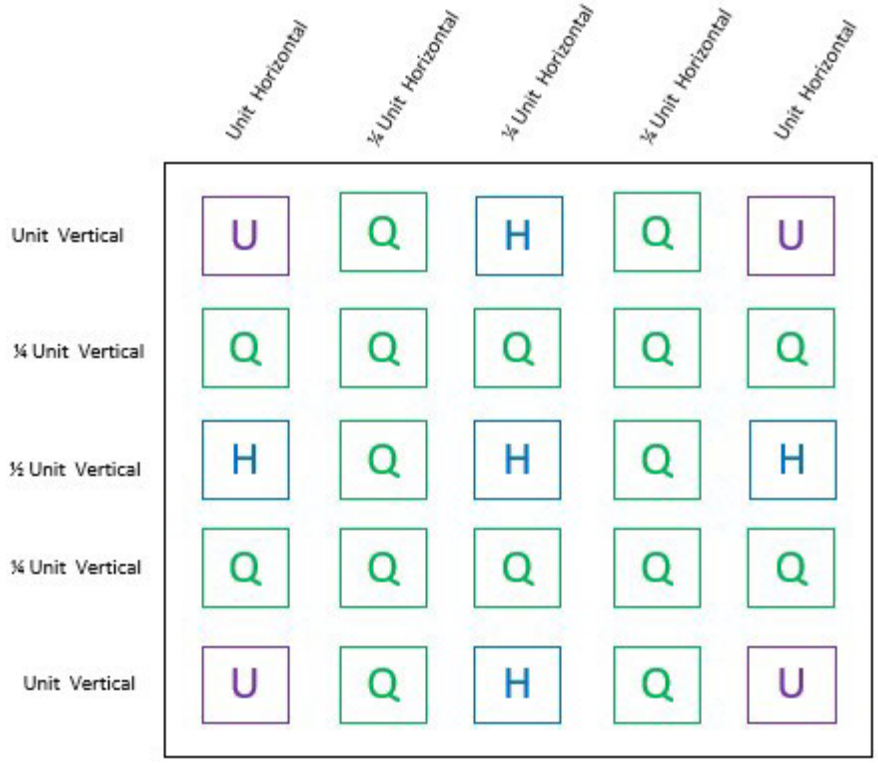
Because of this fractional pixel movement, the pixels in the **pink** box in Frame 2 do not have matching pixels in Frame 1.

27. Nevertheless, motion of the ball can still be represented by a motion vector even when objects in the video move by distances that do not line up with actual pixels. To do so, instead of matching groups of actual pixels in Frame 1 to groups of actual pixels in Frame 2, values of virtual pixels (which are not actually displayed)—known as "sub-pixels"—that are located ***between*** the actually displayed pixels in one frame can be matched to actual pixels in another frame. The values of these virtual sub-pixels can be determined using a well-known mathematical process known as "interpolation." This interpolation technique is based on the theory that the virtual sub-pixels that lie between actual pixels will have color and brightness values "between" the color and brightness values of their

neighbouring actual pixels. For example, the values of "sub-pixels" can be interpolated by using a weighted sum of the neighbouring pixels. As illustrated in the figure below, the four pixels in the **pink** box in Frame 2 match the four interpolated, virtual sub-pixels in the **blue** box that are highlighted with yellow circles in Frame 1. Therefore, the motion of these sub-pixels can be represented by the **purple** motion vector originating from sub-pixels in Frame 1 and ending at actual pixel locations in Frame 2:



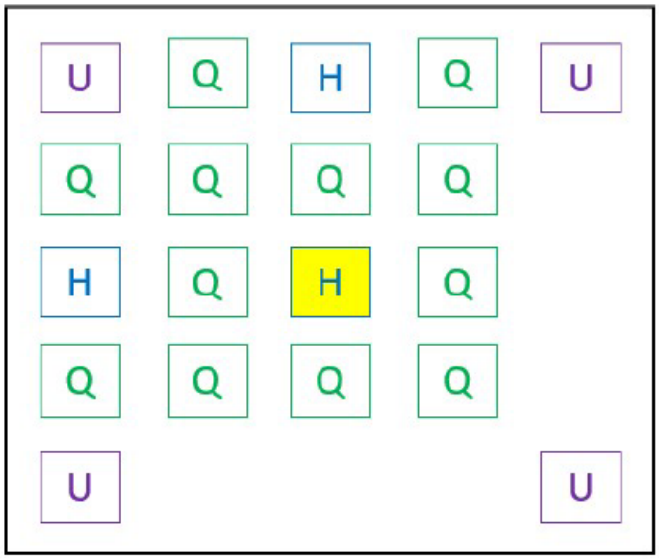
28. Terminology for describing the various pixel and sub-pixel locations is illustrated in the figure below. The figure shows four actual pixels, called "unit pixels U," in **purple**, and certain exemplary "sub-pixels H" in **blue** and "sub-pixels Q" in **green**:



The sub-pixels H in **blue** are called "½ sub-pixels" because they are located halfway between two unit pixels U in **purple** in at least one of the horizontal direction or the vertical direction. The sub-pixels Q in **green** are called "¼ sub-pixels" because they are located a quarter-way between two unit pixels U in **purple** in at least one of the horizontal direction or the vertical direction.

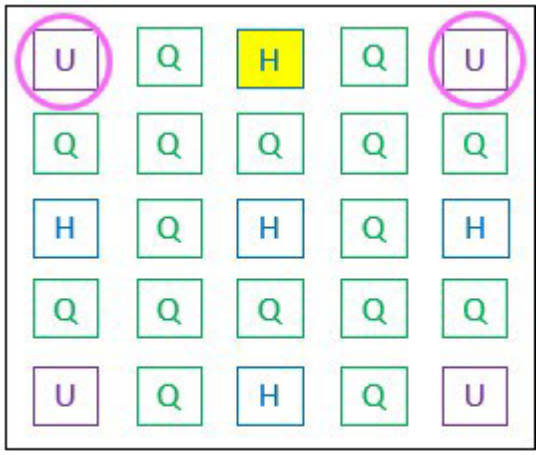
29. Further, the term "unit ***horizontal*** location" describes a location in a ***column*** where actual pixels are located. "Unit ***vertical*** location" describes a location in a ***row*** where actual pixels are located. An actual pixel U in **purple** is described as being located at unit horizontal and unit vertical locations. Additional terms identify locations between the unit horizontal and unit vertical locations. A

location halfway between two unit horizontal locations is described as a "half unit horizontal location," while a location halfway between two unit vertical locations is described as a "half unit vertical location." Similarly, the terms "quarter unit horizontal location" and "quarter unit vertical location" refer, respectively, to locations in columns and rows that are a quarter of the way between the unit horizontal and/or unit vertical locations. For instance, the highlighted sub-pixel H in **blue** below is located at the half unit horizontal and half unit vertical locations because it is located halfway between unit horizontal locations and halfway between the unit vertical locations.



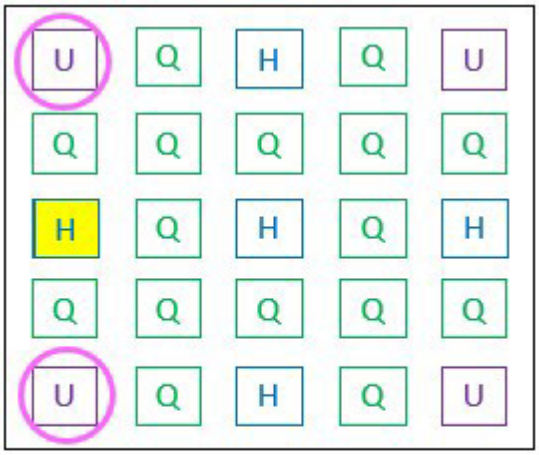
30. One of the most basic and widely used forms of interpolation, known as linear interpolation, refers to interpolating values of sub-pixels using the values of pixels or other sub-pixels along a horizontal, vertical, or diagonal line. The below figure, for example, shows that the highlighted ½ sub-pixel H in **blue** at the

half unit horizontal and unit vertical locations may be horizontally interpolated using values of the two nearest unit pixels U in **purple** and circled in the same row.



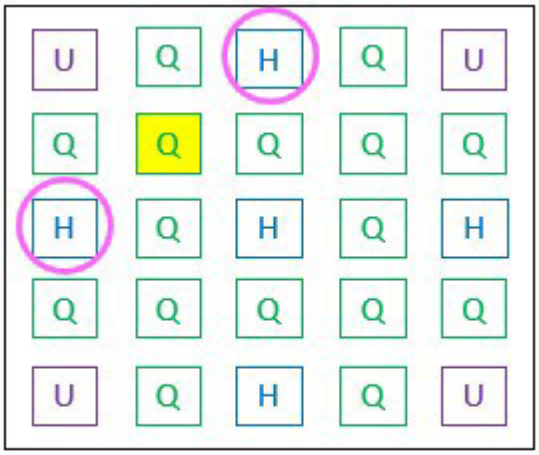
½ unit horizontal
unit vertical

31. Similarly, as illustrated in the figure below, one can interpolate the highlighted ½ sub-pixel H in **blue** at the unit horizontal and half unit vertical locations vertically using values of the unit pixels U in **purple** and circled in the same column.



unit horizontal
½ unit vertical

32. Likewise, the below figure shows that the highlighted ¼ sub-pixel Q in **green** at the quarter unit horizontal and quarter unit vertical locations may be diagonally interpolated using the values of the two nearest ½ sub-pixels H in **blue** and circled that were already interpolated, as I discuss above.

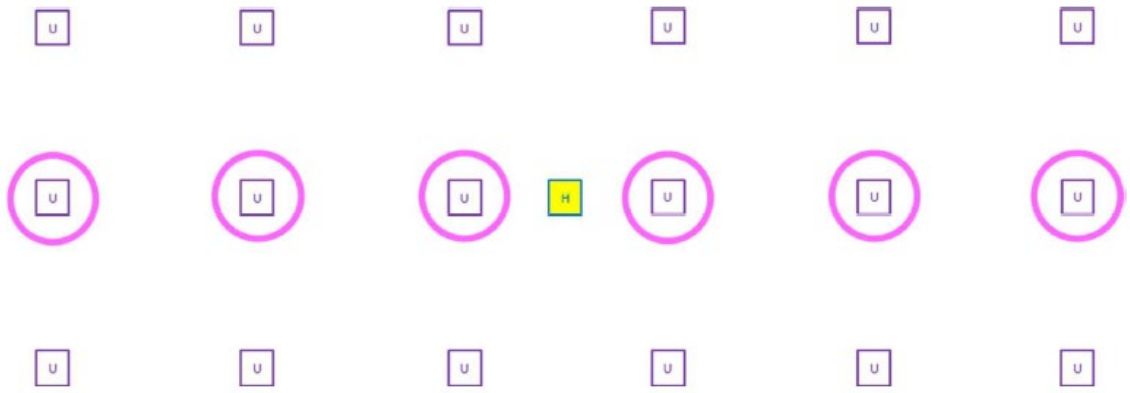


¼ unit horizontal
¼ unit vertical

Linear interpolation, including horizontal, vertical, and diagonal interpolation, was well-known before the '273 patent. *See, e.g.,* VCEG-M28; Fandrianto, Figure 5, 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels [¼ sub-pixels] Q1-Q8 may be practiced, if desired. For example, the Q2 pixels may be derived from a ***horizontal*** interpolation of the Q3 pixels with the results of a ***vertical*** interpolation of the data in the B block 74 and the X block 70. As a further example, the Q2 pixels may be derived from a

diagonal interpolation of the data in the A block 72 and the B block 74." (emphases added)).

33. While the above paragraphs discuss interpolation using values for pixels and sub-pixels that are located next to the sub-pixel being interpolated, sub-pixels can also be interpolated by using other nearby pixels and sub-pixels. An illustration of this concept is shown below: the highlighted ½ sub-pixel "H" in **blue** may be interpolated based on the values of the six nearest unit pixels U in **purple** and circled in the same row:



For example, as was well-known, such interpolation could be used as a low-pass filter to smooth out local extremes.

VI. OVERVIEW OF THE '273 PATENT

A. The Alleged Problem in the Prior Art

34. As I explain above, ½ sub-pixel and ¼ sub-pixel interpolation was well-known before the '273 patent. And the '273 patent recognizes this fact. EX-1001, 6:35-41 ("[T]o enable more accurate modelling of real motion and to help

reduce the amount of PE information that must be transmitted from encoder to decoder, many video coding standards, such as H.263(+)(++) and H.26L, allow motion vectors to point 'in between' image pixels. In other words, the motion vectors can have 'sub-pixel' resolution."), 7:1-19 (explaining that the prior art systems: "Having interpolated the values of the sub-pixels at half-pixel resolution, each of the eight search blocks is compared with the macroblock whose motion vector is being sought. As in the block matching process performed in order to determine the motion vector with full pixel resolution, the macroblock is compared with each of the eight search blocks according to some criterion, for example a [sum of absolute differences, or] SAD. As a result of the comparisons, a minimum SAD value will generally be obtained. Depending on the nature of the motion in the video sequence, this minimum value may correspond to the location specified by the original motion vector (having full-pixel resolution), or it may correspond to a location having a half-pixel resolution. Thus, it is possible to determine whether a motion vector should point to a full-pixel or sub-pixel location and if sub-pixel resolution is appropriate, to determine the correct sub-pixel resolution motion vector. It should also be appreciated that the scheme just described can be extended to other sub-pixel resolutions (for example, one-quarter-pixel resolution) in an entirely analogous fashion."), 9:65-12:64.

35. The background section of the ’273 patent describes the sub-pixel interpolation scheme of TML6’s sub-pixel interpolation method with reference to Figure 13(a), which is copied below. The ’273 patent explains that the values of the ¼ sub-pixels “d” and “e” in **green** and not circled below are calculated using horizontal or vertical linear interpolation:

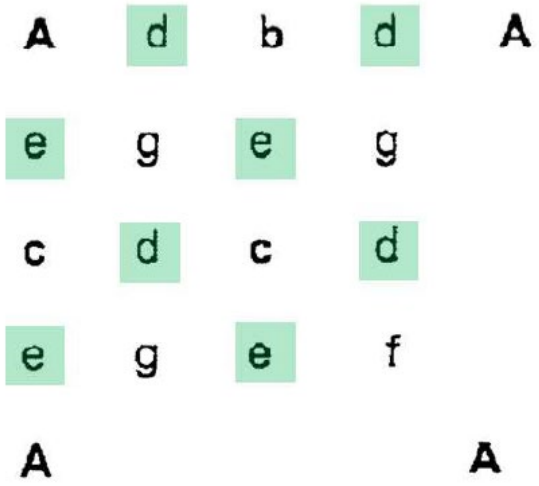


Fig. 13(a)

EX-1001, 11:64-12:33 (“Values for the ¼ resolution sub-pixels labelled d [in Figure 13(a)] are calculated as follows. Values for ¼ resolution sub-pixels d located at quarter horizontal and unit vertical locations, are calculated from the value of the immediately neighbouring pixel at unit horizontal and unit vertical location (pixel A) and the intermediate value b calculated in step (1) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and unit vertical location (½ resolution sub-pixel b), according to d=(32A+b+32)/64.

Values for ¼ resolution sub-pixels d located at quarter horizontal and half vertical locations, are interpolated using the intermediate values c calculated for the immediately neighbouring ½ resolution sub-pixels c which lie at unit horizontal and half vertical location and half horizontal and half vertical locations respectively, according to $d=(32c_1+c_2+1024)/2048$. Again operator / indicates division with truncation and the finally obtained ¼ resolution sub-pixel values d are clipped to lie in the range [0, 2^n-1].”), 28:36-37 (Figure 13(a) “shows a notation for describing pixel and sub-pixel locations specific to TML6.”).

36. But the ’273 patent further states that, in TML6, ¼ sub-pixels “g” and “f” in **green** and circled below are not calculated using horizontal or vertical interpolation. In fact, they are not calculated using linear interpolation. Instead, they are calculated using the values for the pixels and sub-pixels in the four corners of a square surrounding the ¼ sub-pixels. EX-1001, 8:38-48 (“Two interpolation schemes have been developed as part of the work ongoing in the ITU-Telecommunications Standardization Sector, Study Group 16, Video Coding Experts Group (VCEG), Questions 6 and 15. These approaches were proposed for incorporation into ITU-T recommendation H.26L and have been implemented in test models (TML) for the purposes of evaluation and further development. The test model corresponding to Question 15 is referred to as Test Model 5 (TML5), while that resulting from Question 6 is known as Test Model 6 (TML6). The

interpolation schemes proposed in both TML5 and TML6 will now be described."), 12:34-64 ("Values for ¼ resolution sub-pixels labelled g are computed using the value of the nearest original pixel A and the intermediate values of the three nearest neighbouring ½ resolution sub-pixels, according to $g=(1024A+32b+32c_1+c_2+2048)/4096$. As before, operator / indicates division with truncation and the finally obtained for ¼ resolution sub-pixel values g are clipped to lie in the range $[0, 2^n-1]$. The value for ¼ resolution sub-pixel f is interpolated by averaging the values of the 4 closest pixels at unit horizontal and vertical locations, according to $f=(A_1+A_2+A_3+A_4+2)/4$, where the locations of pixels A_1, A_2, A_3 and A_4 are the four nearest original pixels. In the decoder, sub-pixel values can be obtained directly by applying 6-tap filters in horizontal and vertical directions. In the case of ¼ sub-pixel resolution, referring to FIG. 13(a), the filter coefficients applied to pixels and sub-pixels at unit vertical location are [0, 0, 64, 0, 0, 0] for a set of six pixels A, [1, −5, 52, 20, −5, 1] for a set of six sub-pixels d, [2, −10, 40, 40, −10, 2] for a set of six sub-pixels b, and [1, −5, 20, 52, −5, 1] for a set of six sub-pixels d. These filter coefficients are applied to respective sets of pixels or sub-pixels in the same row as the sub-pixel values being interpolated. After applying the filters in the horizontal and vertical directions, interpolated value c is normalized according to $c=(c+2048)/4096$ and clipped to lie in the range $[0, 2^n-1]$. When a motion vector points to an integer pixel position in either the horizontal or

vertical direction, many zero coefficients are used. In a practical implementation of TML6, different branches are used in the software which are optimised for the different sub-pixel cases so that there are no multiplications by zero coefficients."), Figure 13(a).

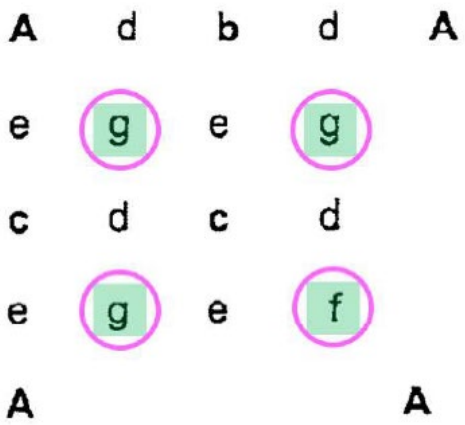


Fig. 13(a)

37. In particular, the '273 patent states that TML6 interpolates ¼ sub-pixel "g" in **green** by using the nearest unit pixel "A" in **purple** and the values for the three nearest ½ sub-pixels "b," "c," and "c," all in **blue**. This is illustrated below.

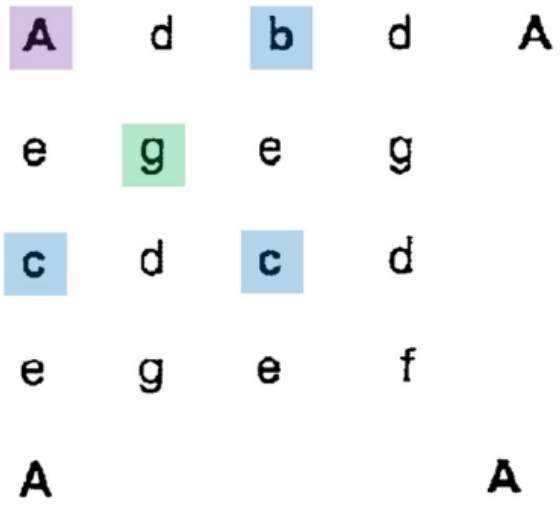


Fig. 13(a)

38. The ’273 patent states that TML6’s method is computationally intensive because it requires “high precision arithmetic,” as not “all of the ¼ resolution sub-pixel values are calculated from [unit] pixels or ½ resolution sub-pixel values using linear interpolation.” EX-1001, 13:11-18 (“[A] disadvantage of TML6 is that high precision arithmetic is required both in the encoder and in the decoder. High precision interpolation requires more silicon area in ASICs and requires more computations in some CPUs. Furthermore, implementation of direct interpolation as specified in TML6 in an on-demand fashion has a high memory requirement. This is an important factor, particularly in embedded devices.”), 44:9-26 (“In comparison with TML6, the method according to the invention does not require high precision arithmetic to be used in the

calculation of all sub-pixels. Specifically, as all of the ¼ resolution sub-pixel values are calculated from original image pixels or ½ resolution sub-pixel values using linear interpolation, lower precision arithmetic can be used in their interpolation."). The '273 patent further states that this "requires more silicon area in [Application Specific Integrated Circuits] and requires more computations in some [processors]." *Id.,* 13:11-18 ("[A] disadvantage of TML6 is that high precision arithmetic is required both in the encoder and in the decoder. High precision interpolation requires more silicon area in ASICs and requires more computations in some CPUs. Furthermore, implementation of direct interpolation as specified in TML6 in an on-demand fashion has a high memory requirement. This is an important factor, particularly in embedded devices.").

B. The Alleged Invention of the '273 Patent

39. The '273 patent purports to address this problem by claiming, as its allegedly novel concept, using a diagonal linear interpolation of two sub-pixels to calculate ¼ sub-pixels "h" rather than four. This is shown in Figure 14(a) of the '273 patent.

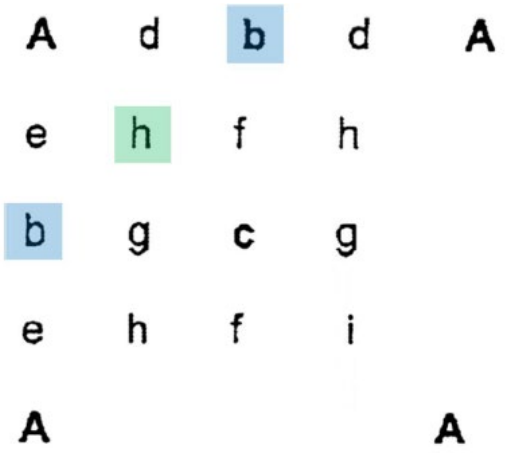


Fig. 14(a)

EX-1001, Figure 14(a), 28:42-43 (Figure 14(a) "shows a notation for describing pixel and sub-pixel locations specific to the invention"). As illustrated above, unlike in TML6, the ¼ sub-pixel "h" in **green** at the quarter unit vertical and quarter unit horizontal locations may be calculated by using just two diagonally opposite ½ sub-pixels "b" in **blue**. *Id.,* Figure 14(a), 42:53-43:7 ("There are two alternatives for interpolating values for the ¼ resolution sub-pixels labelled h. Both involve linear interpolation along a diagonal line linking ½ resolution sub-pixels neighbouring the ¼ resolution sub-pixel h being interpolated. In a first embodiment, a value for sub-pixel h is calculated by averaging the values of the two ½ resolution sub-pixels b closest to sub-pixel h. In a second embodiment, a value for sub-pixel h is calculated by averaging the values of the closest pixel A and the closest ½ resolution sub-pixel c. ***It should be appreciated that this***

provides the possibility of using different combinations of diagonal interpolations to determine the values for sub-pixels h within the confines of different groups of 4 image pixels A." (emphasis added)).

40. The '273 patent asserts that this technique improves on TML6 because it "does not require high precision arithmetic to be used in the calculation of all sub-pixels." *Id.,* 44:9-34 ("In comparison with TML6, the method according to the invention does not require high precision arithmetic to be used in the calculation of all sub-pixels."), 44:9-11 ("Specifically, as all of the ¼ resolution sub-pixel values are calculated from original image pixels or ½ resolution sub-pixel values using linear interpolation, lower precision arithmetic can be used in their interpolation."). According to the patent, this may reduce the amount of silicon dedicated to the interpolation function and thus increase the speed at which the calculations are performed. *Id.,* 44:16-34 ("Consequently, in hardware implementations of the inventive method, for example in an ASIC (Application Specific Integrated Circuit), the use of lower precision arithmetic reduces the number of components (e.g., gates) that must be devoted to the calculation of ¼ resolution sub-pixel values. This, in turn, reduces the overall area of silicon that must be dedicated to the interpolation function. As the majority of sub-pixels are, in fact, ¼ resolution sub-pixels (12 out of the 15 sub-pixels illustrated in FIG. 14 a), the advantage provided by the invention in this respect is particularly

significant. In software implementations, where sub-pixel interpolation is performed using the standard instruction set of a general purpose CPU (Central Processor Unit) or using a DSP (Digital Signal Processor), a reduction in the precision of the arithmetic required generally leads to an increase in the speed at which the calculations can be performed. This is particularly advantageous in 'low cost' implementations, in which it is desirable to use a general purpose CPU rather than any form of ASIC.").

VII. LEVEL OF ORDINARY SKILL

41. I have considered the '273 patent from the perspective of a POSITA at the time of the alleged invention. A POSITA at the time of the alleged invention of the '273 patent, *i.e.,* September 17, 2001, would have had either (1) a Bachelor's degree in electrical engineering, computer engineering, computer science, or a comparable field of study such as physics, and (2) approximately two to three years of practical experience with video encoding/decoding. Additional experience can substitute for the level of education, and vice-versa. At the time of the claimed invention, I would have qualified as a POSITA under this standard.

42. I note that I previously submitted declarations in four IPRs challenging the '273 patent: IPR2021-00040, IPR2021-00041, IPR2021-00042, and IPR2021-00043. The POSITA definition I propose now differs slightly from before but is substantially similar. The definition that I proposed in the previous

IPRs is: "A POSITA at the time of the alleged invention of the '273 patent, *i.e.*, September 17, 2001, would have had either (1) a bachelor's degree plus four years of experience in electrical engineering, computer engineering, computer science, or a related field; or (2) a master's degree plus two years of experience in electrical engineering, computer engineering, computer science, or a related field." This difference does not affect my opinions on the '273 patent. Thus, the Challenged Claims are invalid under both definitions.

VIII. PRIOR ART OVERVIEW

43. It is my opinion that all elements of the claimed invention of the '273 patent, including the purportedly novel feature I described above, were well-known before the patent's September 17, 2001 alleged priority date.

A. TML6

44. "Test Model Long 6," known as "TML6," is a technique for encoding and decoding a video signal that was developed by the Video Coding Expert Group (VCEG). *See* EX-1001, 8:38-48 ("Two interpolation schemes have been developed as part of the work ongoing in the ITU-Telecommunications Standardization Sector, Study Group 16, Video Coding Experts Group (VCEG), Questions 6 and 15. These approaches were proposed for incorporation into ITU-T recommendation H.26L and have been implemented in test models (TML) for the purposes of evaluation and further development. The test model corresponding to

Question 15 is referred to as Test Model 5 (TML5), while that resulting from Question 6 is known as Test Model 6 (TML6).”). The VCEG is a working group of the ITU Telecommunication Standardization Section (ITU-T) organization that is tasked with developing standards for video coding. *Id.* The VCEG created TML6 as a part of developing a standard for adoption by ITU. *Id.* The technique is summarized in a description document entitled “VCEG-L45d0” (VCEG-L45) and was implemented in source code (“TML6 source code”) (TML6). The TML6 source code is a fully developed codec (*i.e.,* encoder and decoder) and could be run on a processor to perform the encoding and decoding functions I describe below, among other functions. *See* EX-1001, 8:38-48 (“Two interpolation schemes have been developed as part of the work ongoing in the ITU-Telecommunications Standardization Sector, Study Group 16, Video Coding Experts Group (VCEG), Questions 6 and 15. These approaches were proposed for incorporation into ITU-T recommendation H.26L and have been implemented in test models (TML) for the purposes of evaluation and further development. The test model corresponding to Question 15 is referred to as Test Model 5 (TML5), while that resulting from Question 6 is known as Test Model 6 (TML6). The interpolation schemes proposed in both TML5 and TML6 will now be described.”).

45. I understand that the TML6 source code was publicly available as a printed publication no later than March 28, 2001, and is therefore prior art to the

’273 patent. Wenger, ¶¶20-38. It was common for subscribers to the VCEG email reflector—including not only VCEG participants but also non-participants such as students and members of the general public—to be notified by email when files were uploaded to the FTP folder. Wenger, ¶¶ 26-27. And because the folder was not password-protected, and was publicly accessible, it was common for files to be downloaded from the folder, especially in connection with VCEG meetings. Wenger, ¶¶ 20-22, 27-28.

46. VCEG-L45, titled “H.26L Test Model Long Term Number 6 (TML6) draft0,” explains that the TML6 code is “for the development of a new compression method ITU-T recommendation - H.26L.” VCEG-L45, §1. I understand that VCEG-L45 was publicly available as a printed publication no later than March 10, 2001, and is therefore prior art to the ’273 patent. Wenger, ¶¶18-38. As Dr. Wenger explains, the TML6 comprises a subset of a group of files that together make up the fully functional source code of a working video encoder and decoder. Wenger, ¶¶19-21. These files were packaged in a single compressed file names “***tml6pre.zip***” and made publicly available via the VCEG FTP server by March 28, 2001. Wenger, ¶¶18-28. Except for the “readme.txt,” the files that comprise TML6 originate from subfolders in “***tml6pre.zip***” named “***lencod***,” “***ldecod***,” and “***bin***” containing files with executable code. TML6 thus excerpts the source code that was packaged and made publicly available in “***tml6pre.zip***” by

VCEG. VCEG-L45 was prepared by Study Group 16 of the VCEG of ITU-T and addresses Question 6 ("Q.6"). Question 6 was well known to a POSITA to address standardization of video coding for audio/visual services such as videoconferencing, video telephony, streaming, broadcast, and file download. VCEG-L45 describes encoding and decoding aspects of the TML6 code, such as sub-pixel interpolation for fractional pixel motion compensation (§3.6) and Network Adaptation Layer for IP networks (§8).

47. TML6 discloses methods of interpolating ½ and ¼ sub-pixels. EX-1001, 11:25-13:27; TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632). In particular, TML6 discloses pixels and sub-pixels arranged into rows and columns, where sub-pixel values are interpolated as follows:

48. The '273 patent explains, with reference to Figure 13(b) below, that in TML6's sub-pixel interpolation method, "[t]he value for the sub-pixel at half unit horizontal and unit vertical location [in **blue**], that is ½ resolution sub-pixel b in FIG. 13*a*, is obtained by first calculating an intermediate value b using a six tap filter. The filter calculates b based upon the values of the 6 pixels (A_1 to A_6) [in **purple**] situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13*b*."

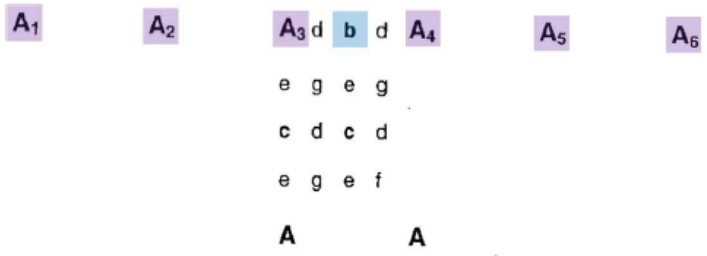


Figure13(b)

EX-1001, Figures 13(a), (b), 11:35-45 ("The value for the sub-pixel at half unit horizontal and unit vertical location, that is ½ resolution sub-pixel b in FIG. 13*a*, is obtained by first calculating an intermediate value b using a six tap filter. The filter calculates b based upon the values of the 6 pixels (A_1 to A_6) situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13*b*, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value of b is then calculated as $b=(b+16)/32$ and is clipped to lie in the range $[0, 2^n-1]$. As before, the operator / denotes division with truncation."). As I explain below with respect to Element 1[d], I have confirmed that the TML6 code discloses this interpolation method.

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

```
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 563-575, 588).

49. The ’273 patent further explains with reference to Figure 13(c) below, that in TML6, “[v]alues for the ½ resolution sub-pixels labelled c [in **blue**] are obtained by first calculating intermediate values c. Referring to FIG. 13(c), an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) [in **purple** on the left] situated in a column at unit horizontal locations and unit vertical locations symmetrically about c. . . . Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to [values of the 6 previously-interpolated sub-pixels b_1-b_6 in **blue** on the right].”

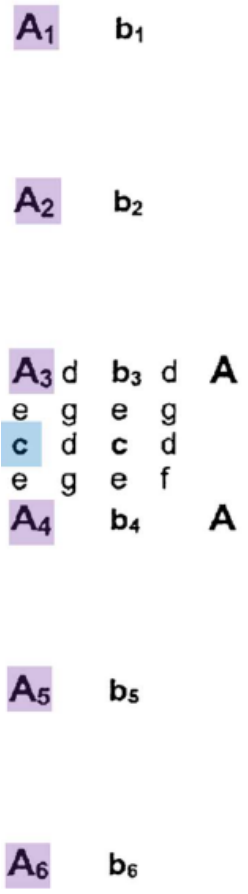


Figure 13(c)

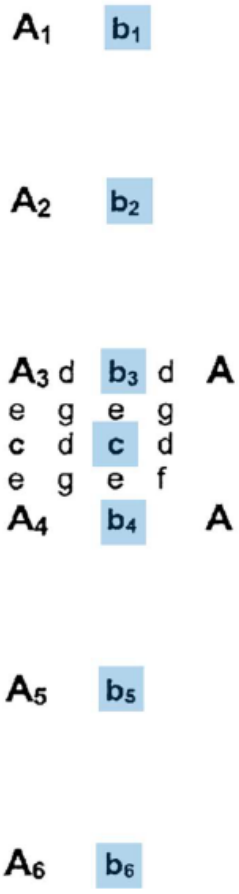


Figure 13(c)

EX-1001, Figure 13(c), 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13c, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c

located at unit horizontal and half vertical location is then calculated according to c=(c+16)/32. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.”). As I explain below in my analysis of Elements 1[d]-1[e], I have confirmed that the TML6 code discloses this interpolation method. Elements 1[d]-1[e], *infra*.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).

50. And as the ’273 patent describes with reference to Figure 13(a) (below), in TML6 “[v]alues for ¼ resolution sub-pixels labelled g [in **green**] are computed using the value of the nearest original pixel A [in **purple**] and the intermediate values of the three nearest-neighbouring ½ resolution sub-pixels [**blue**].”

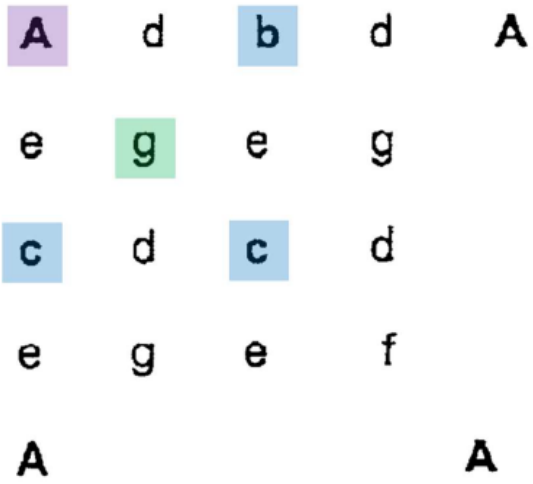


Fig. 13(a)

EX-1001, Figure 13(a), 12:34-40 ("Values for ¼ resolution sub-pixels labelled g are computed using the value of the nearest original pixel A and the intermediate values of the three nearest neighbouring ½ resolution sub-pixels, according to $g=(1024A+32b+32c_1+c_2+2048)/4096$. As before, operator / indicates division with truncation and the finally obtained for ¼ resolution sub-pixel values g are clipped to lie in the range $[0, 2^n-1]$."). As I described below with respect to Element 1[c], I have confirmed that the TML6 code discloses this interpolation method.

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                ][i/2                ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                ] +
612                       img4Y_tmp[j/2                ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
614
615                   /* "funny posision" */
616                   if( ((i&3) == 3)&&(((j+1)&3) == 3))
617                   {
618                       mref[img->frame_cycle][j+1][i] = (mref[img->
                        frame_cycle][j-2         ][i-3         ] +
619                           mref[img->frame_cycle][min(je2,j+2)][i-3
                                ] +
620                           mref[img->frame_cycle][j-2         ][min(ie2,i
                           +1)] +
621                           mref[img->frame_cycle][min(je2,j+2)][min(ie2,i
                           +1)] + 2 )/4;
622                   }
623               }
624           }
625
626           /*  Chroma: */
627           for (uv=0; uv < 2; uv++)
628               for (j=0; j < img->height_cr; j++)
629                   memcpy(mcef[img->frame_cycle][uv][j],imgUV[uv][j],img
                    ->width_cr); /* just copy 1/1 pix, interpolate
                    "online" */
630
631   }
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-625).

1. Motion Prediction During Encoding

51. As shown in TML6, portions of the TML6 source code are stored in a main folder, ***tml6pre***, and subfolders including a ***lencod*** folder, a ***ldecod*** folder,

and a ***bin*** folder. *See generally* TML6. This folder structure is represented by the file names at the top right corner of TML6. *Id.* The folders and subfolders include files with executable code. *Id.* Upon my review of the code, the ***lencod*** folder generally includes files of code to encode video data, and likewise, the ***ldecod*** folder generally includes files of code to decode video data. *Id.* The TML6 code includes functions that perform specific tasks during encoding and decoding of video frames, such as motion prediction and sub-pixel interpolation to aid in motion prediction with fractional pixel resolution. With respect to motion prediction in the encoder, these functions include:

- ***main()***: initializes encoding, calls functions to perform encoding, and writes output of the encoding. *See, e.g.,* TML6, 69-70 (/tml6pre.zip/lencod/src/lencod.c, ll. 2-112).
- ***encode_one_frame()***: called by the ***main()*** function and encodes an image ("frame") of video data. *See, e.g.,* TML6, 48-50 (/tml6pre.zip/lencod/src/image.c, ll. 33-163 (the *encode_one_frame ()* function "[e]ncodes one frame")), 70 (/tml6pre.zip/lencod/src/lencod.c, ll. 79-95 (the *main()* function calls the *encode_one_frame()* function to "encode one I- or P-frame")).
- ***encode_one_slice()***: called by the ***encode_one_slice()*** function and encodes a slice, *i.e.,* part of a frame of video data. *See, e.g.,* TML6, 50-51

(/tml6pre.zip/lencod/src/image.c, ll. 164-225 (the *encode_one_slice()* function "[e]ncodes one slice")), 49 (/tml6pre.zip/lencod/src/image.c, ll. 85-96 (the *encode_one_frame()* function "loop[s] over slices" and calls the *encode_one_slice()* function to "[e]ncode the current slice")).

- ***encode_one_macroblock()***: called by the ***encode_one_slice()*** function and encodes macroblocks of a certain pixel size of a slice. *See, e.g.,* TML6, 115-116 (/tml6pre.zip/lencod/src/macroblock.c, ll. 769-842 (the *encode_one_macroblock()* function "[e]ncode[s] one macroblock depending")), 51 (/tml6pre.zip/lencod/src/image.c, ll. 198-220 (the *encode_one_slice()* function "loop[s] over macroblocks" and calls the *encode_one_macroblock()* function to "[e]ncode one macroblock")).
- ***motion_search()***: called by ***encode_one_macroblock()*** function and searches for motion vectors. *See, e.g.,* TML6, 16-24 (/tml6pre.zip/lencod/src/block.c, ll. 734-1180 (the *motion_search()* function performs "routine motion search")), 115 (/tml6pre.zip/lencod/src/macroblock.c ll. 787-769-842 (the *encode_one_macroblock()* function calls the *motion_search()* function to "[m]otion vector search")).
- ***oneforthpix()***: interpolates sub-pixels in the encoder to ¼ pixel resolution. *See, e.g.,* TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632 (the

oneforthpix() function "[u]psample[s] 4 times [*i.e.,* to ¼ pixel resolution] and store[s] in buffer [***mref***] for multiple reference frames")), 49 (tml6pre.zip/lencod/src/image.c, ll. 99-109 (the *encode_one_frame()* function calls the *oneforthpix()* function)).

2. Interpolating Sub-Pixels During Encoding

a) ***Brief Overview of the Oneforthpix() Function***

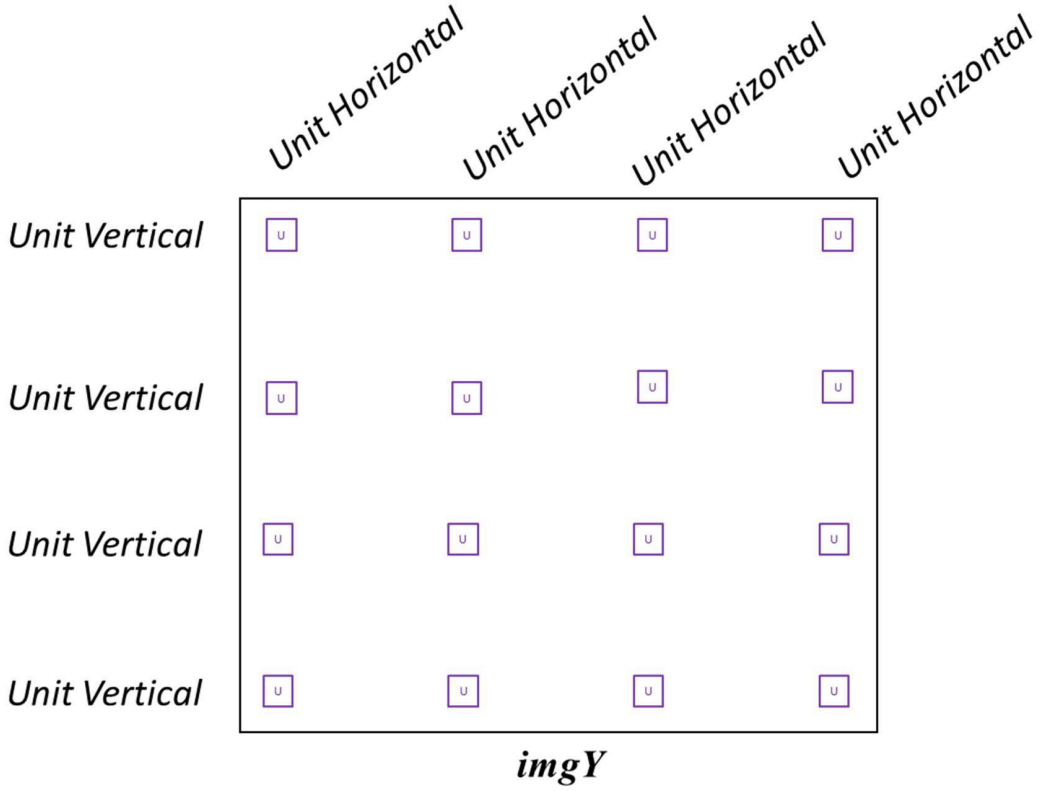
52. When the TML6 source code performs motion prediction to encode a video data stream, it also executes the source code in a file called "***image.c***" contained in the ***lencod*** folder. TML6, 57-59 (/tml6pre.zip/lencod/src/image.c). Within the ***image.c*** file, lines 538 to 632 of the source code are the ***oneforthpix()*** function[1] for interpolating ½ and ¼ sub-pixels based on the actual pixels (which I refer to as "unit pixels") representing a frame. TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632).

[1] This function is used to interpolate I-frames and P-frames. The TML6 encoder uses this function, as well as the ***oneforthpix_2()*** function at lines 633 to 719 of the ***image.c*** file to interpolate sub-pixels for B-frames. *See* TML6, 149 (/tml6pre/lencod/src/image.c ll. 97-109). Both functions use the same interpolation scheme. Therefore, descriptions of the interpolation method for ***oneforthpix()*** apply equally to ***oneforthpix_2()***.

53. Namely, the ***oneforthpix()*** function uses the data structure "***imgY***" that represents the brightness—or "luminance"—values[2] of the pixels for one frame of the video data stream.[3] TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632), 49-50 (/tml6pre.zip/lencod/src/image.c, ll. 97-109 ("I- and P-frames:loop-filtered ***imgY***, imgUV ***-> mref[][][][]***, mcef[][][][]" (emphasis added))), 110-112 (/tml6pre.zip/lencod/src/macroblock.c, ll. 480-600 (the *LumaResidualCoding_P()* function populates the ***imgY*** data structure with luminance data from the ***img*** data structure)),115-116 (/tml6pre.zip/lencod/src/macroblock.c, ll. 769-842 (the *encode_one_macroblock()* function calls the *LumaResidualCoding_P()* function for "[r]esidual coding of Luminance")), 40 (/tml6pre.zip/lencod/inc/global.h, l. 284 ("imgY . . . Encoded luma images")), 131 (/tml6pre.zip/lencod/src/lencod.c, ll. 1338-1344). For the sake of illustration, the data from data structure ***imgY*** for a group of 16 pixels, arranged in four columns and four rows, is shown in the figure below.

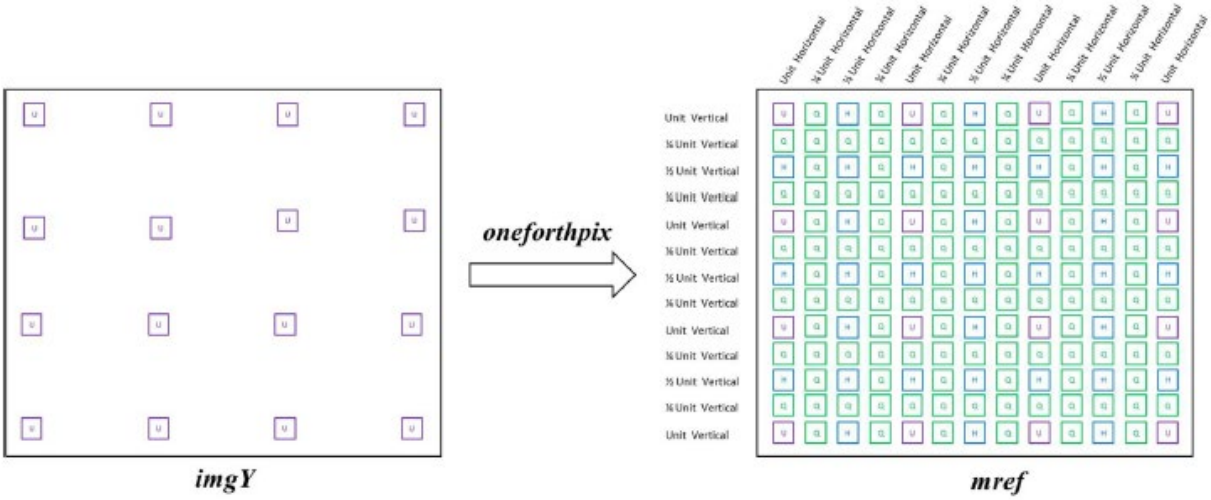
[2] In the following sections, unless otherwise specified, "values" of pixels or sub-pixels refers to luminance values of the pixels or sub-pixels.

[3] Note that the pixel data is only the residual pixel data.



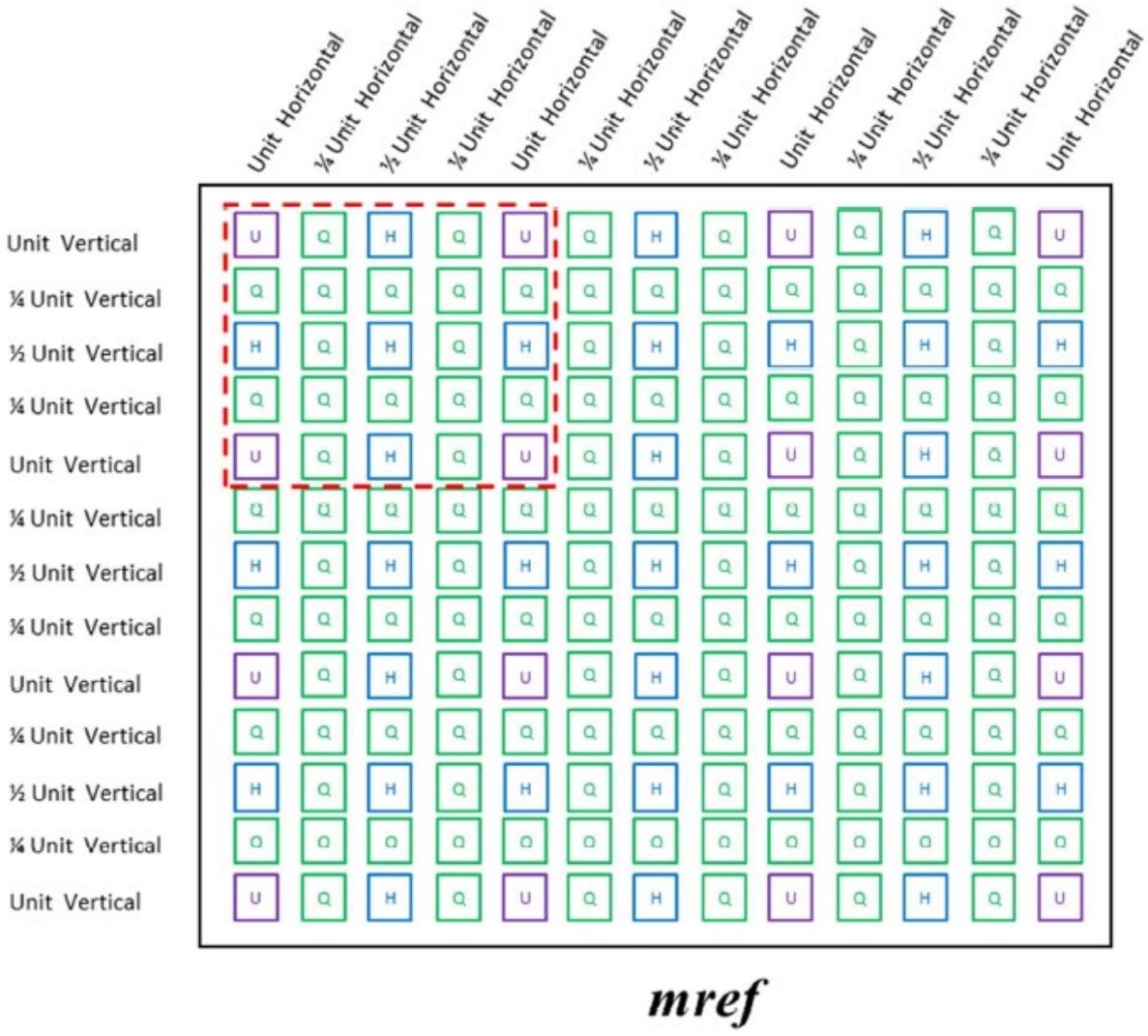
See TML6, 93 (/tml6pre.zip/lencod/src/lencod.c, ll. 1339-1344 ("allocate memory for encoding frame buffers: imgY . . . memory_size += get_mem2D(&imgY, img->height, img->width)")). In the figure, the "unit" pixels in **purple**—*i.e.,* the actual pixels— are shown at "unit horizontal" locations (*i.e.,* columns) along the width of ***imgY*** and "unit vertical" locations (*i.e.,* rows) along the height of ***imgY***. Then, the ***oneforthpix()*** function interpolates the luminance values for the sub-pixels located between the actual pixels based on the pixel luminance data in ***imgY*** and fills the sub-pixel luminance data in a data structure called "***mref***," as illustrated below.

TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632 (*oneforthpix()* function)), 40 (/tml6pre.zip/lencod/inc/global.h ("***mref*** . . . 1/4 pix luma")).



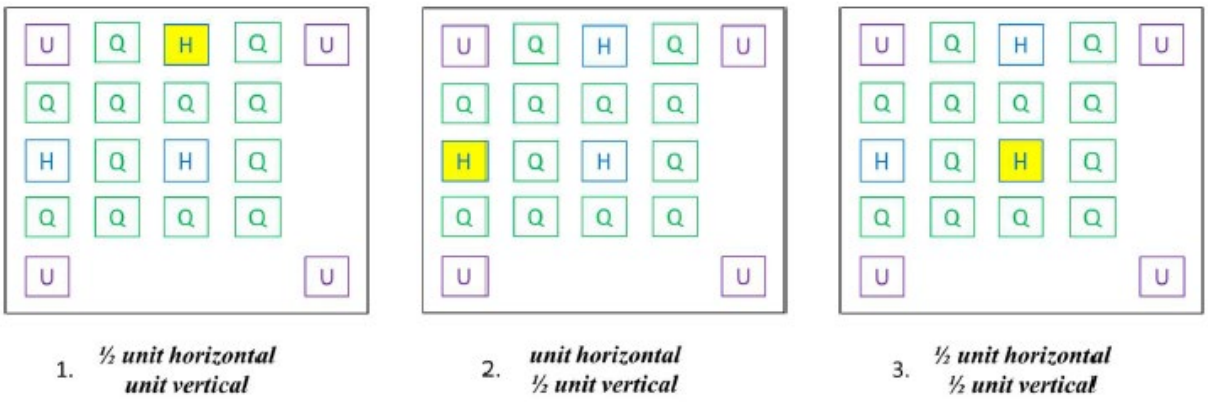
The ***oneforthpix()*** function uses luminance values—from the ***imgY*** data structure—of the actual unit pixels in **purple** located at the unit horizontal and unit vertical locations to interpolate values for the ½ and ¼ sub-pixels in **blue** and in **green** located at "half unit" and "quarter unit" locations, respectively. *Id.* After being stored in the ***mref*** data structure, these interpolated luminance values are used in motion prediction to predict motion to and from fractional pixel locations. *See, e.g.,* TML6, 16-24 (/tml6pre.zip/lencod/src/block.c, ll. 734-1180 (the *motion_search()* function performs "routine motion search" including "1/2 pixel search" and "1/4 pixel search")), 115 (/tml6pre.zip/lencod/src/macroblock.c ll. 769-842 (the *encode_one_macroblock()* function calls the *motion_search()* function to "[m]otion vector search")).

54. More specifically, the locations of the interpolated ½ sub-pixels in **blue** and ¼ sub-pixels in **green** can be shown for the sake of simplicity within the **red** dashed box consisting of two rows and two columns of unit pixels in **purple**.

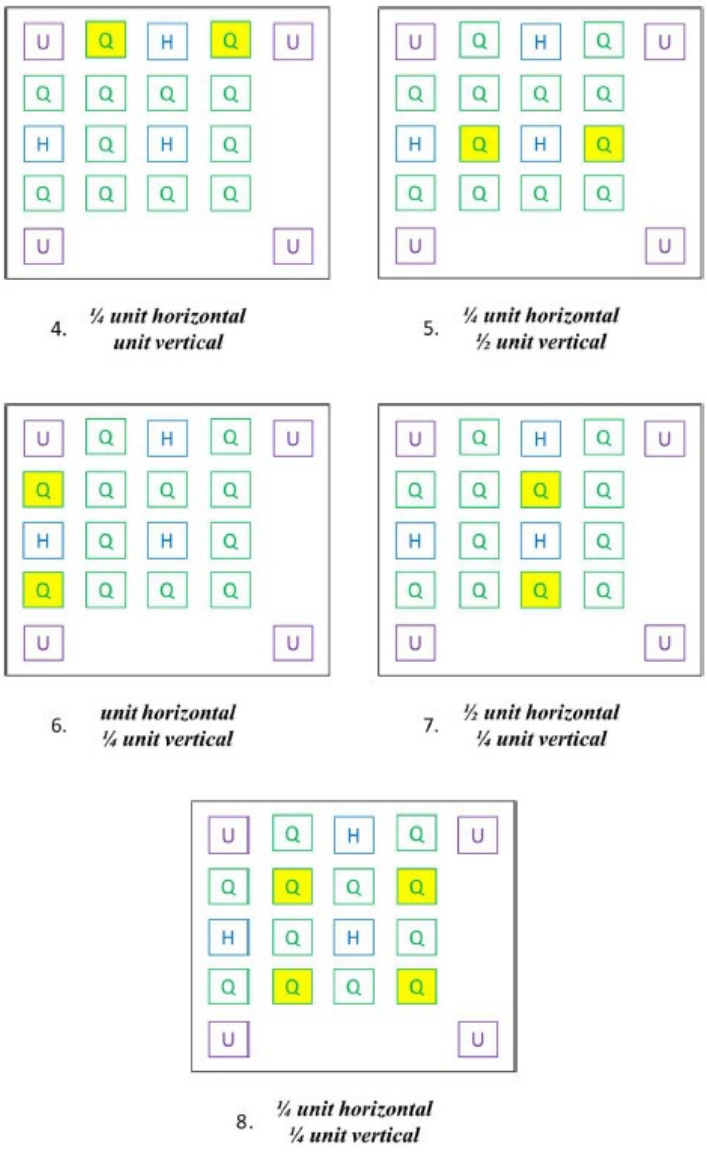


These two rows and two columns of unit pixels in **purple** (and the sub-pixels between them) in the **red** dashed box are reproduced below to illustrate sub-pixel locations as they are described in the '273 patent. EX-1001, 11:35-12:45.

The highlighted ½ sub-pixels in **blue** include sub-pixels located at (1) half unit horizontal and unit vertical locations, (2) unit horizontal and half unit vertical locations, and (3) half unit horizontal and half unit vertical locations. *See, e.g.,* EX-1001, 11:35-11:63. In other words, ½ sub-pixels are located halfway between unit horizontal locations, halfway between unit vertical locations, or halfway between both unit horizontal and unit vertical locations.

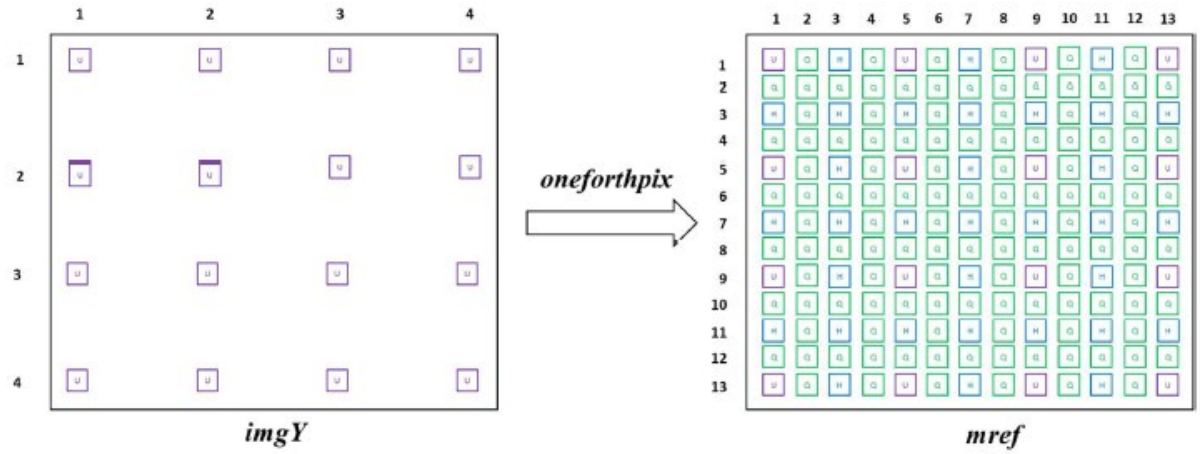


As further illustrated below, the highlighted ¼ sub-pixels in **green** include sub-pixels located at (4) quarter unit horizontal locations (*i.e.,* locations 25% or 75% between the horizontally arranged actual unit pixels) and unit vertical locations, (5) quarter unit horizontal locations and half unit vertical locations, (6) unit horizontal and quarter unit vertical locations (*i.e.,* locations 25% or 75% between the vertically arranged actual unit pixels), (7) half unit horizontal and quarter unit vertical locations, and (8) quarter unit horizontal locations and quarter unit vertical locations. *See, e.g.,* EX-1001, 11:64-12:45.



55. As illustrated below, when the ***oneforthpix()*** function interpolates the values of the unit pixels in the data structure ***imgY*** to generate the luminance values of ½ and ¼ sub-pixels, these values are stored in the data structure ***mref*** along with the unit pixel luminance values. The result is that the ***mref*** data structure is larger than the ***imgY*** data structure to include additional rows and

columns to represent values of sub-pixels ½ and ¼ of the way between the unit pixels. For example, as illustrated below the illustrative ***imgY*** data structure has four rows and four columns of unit pixels.



In the illustrative ***mref*** data structure shown above, there are an additional nine rows, three rows between each pair of rows storing values of unit sub-pixels, to store the ½ and ¼ sub-pixel values. Thus, there are a total of thirteen rows in the illustrative ***mref*** data structure. By the same token, the illustrative ***mref*** data structure has an additional nine columns, three columns between each pair of columns storing values of unit sub-pixels, to store the ½ and ¼ sub-pixel values. Thus, there are a total of thirteen columns in the illustrative ***mref*** data structure. *See, e.g.,* TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632).

56. The ***oneforthpix()*** function contains three main steps that are performed to interpolate sub-pixels to generate the ***mref*** data from the ***imgY*** data.

TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632). The three steps are performed in three respective "for loops" in the TML6 source code, as shown below. *Id.*

Step 1

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

Step 2

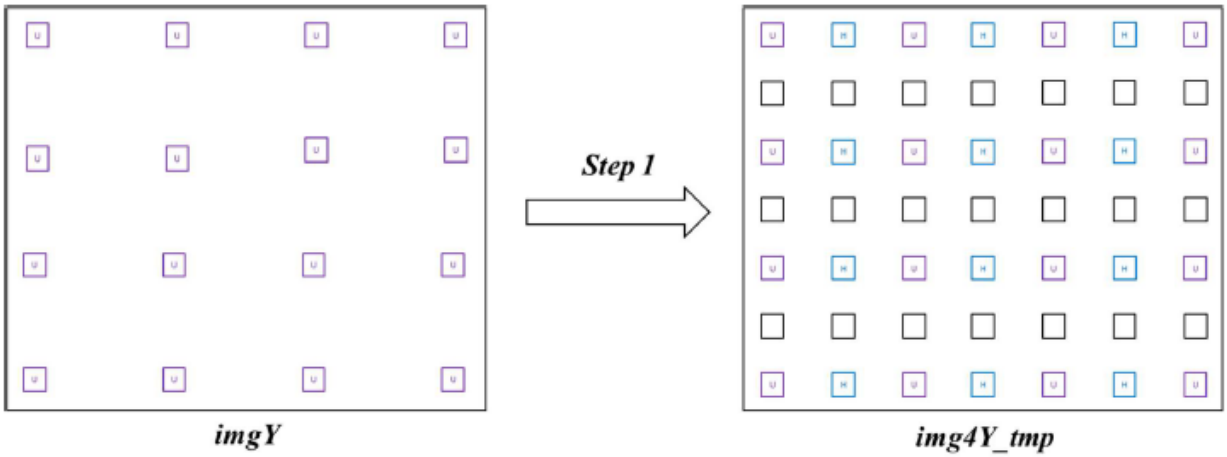
```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

Step 3

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                          )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                          )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                  ][i/2                  ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                  ] +
612                       img4Y_tmp[j/2                  ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                          + 2048)/4096));
614
615                   /* "funny posision" */
616                   if( ((i&3) == 3)&&(((j+1)&3) == 3))
617                   {
618                       mref[img->frame_cycle][j+1][i] = (mref[img->
                          frame_cycle][j-2          ][i-3          ] +
619                           mref[img->frame_cycle][min(je2,j+2)][i-3
                                  ] +
620                           mref[img->frame_cycle][j-2          ][min(ie2,i
                          +1)] +
621                           mref[img->frame_cycle][min(je2,j+2)][min(ie2,i
                          +1)] + 2 )/4;
622                   }
623               }
624           }
625
626           /*  Chroma: */
627           for (uv=0; uv < 2; uv++)
628               for (j=0; j < img->height_cr; j++)
629                   memcpy(mcef[img->frame_cycle][uv][j],imgUV[uv][j],img
                      ->width_cr); /* just copy 1/1 pix, interpolate
                      "online" */
630
631   }
```

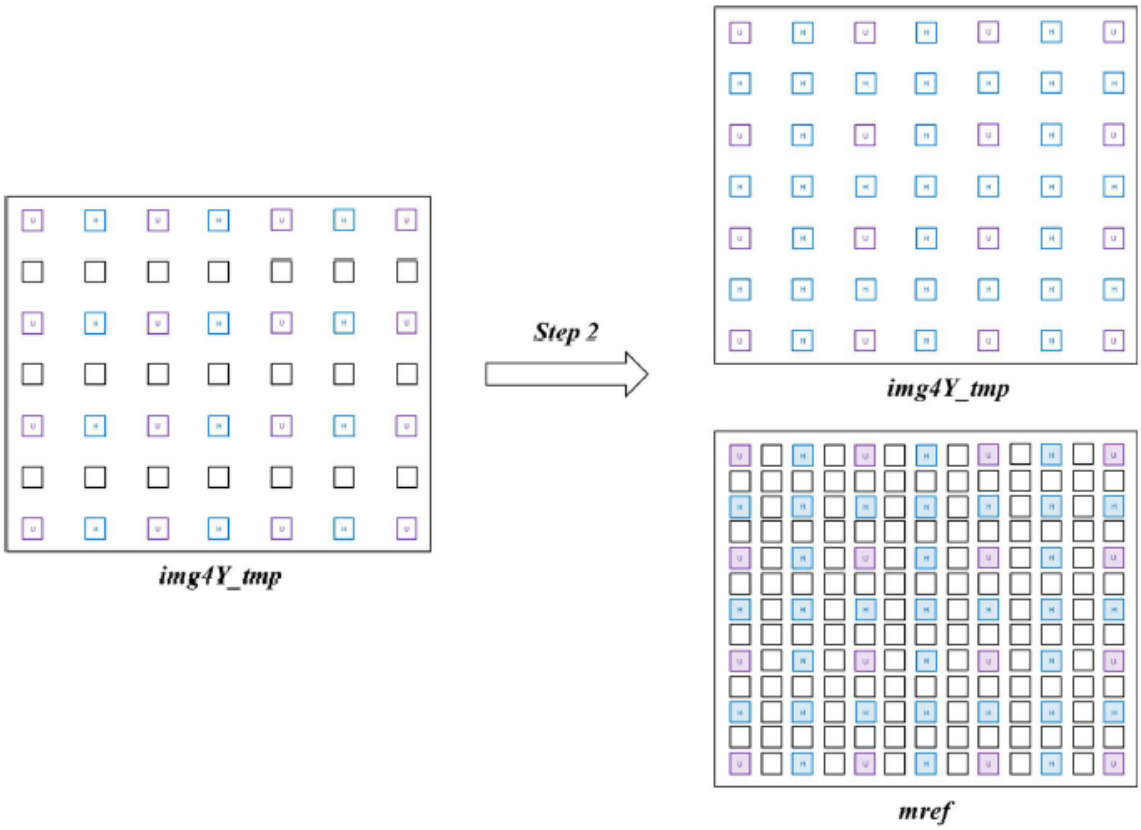
As I discuss in greater detail below, the three steps interpolate values of sub-pixels as follows:

> **Step 1**: determine the intermediate interpolated values for ½ sub-pixels in **blue** at half unit horizontal and unit vertical locations, and store these values along with values of the unit pixels in **purple** in a temporary data structure called ***img4Y_tmp***. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 553-575); *see also* EX-1001, 11:35-45 ("The value for the sub-pixel at half unit horizontal and unit vertical location, that is ½ resolution sub-pixel b in FIG. 13*a*, is obtained by first calculating an intermediate value b using a six tap filter. The filter calculates b based upon the values of the 6 pixels (A_1 to A_6) situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13*b*, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value of b is then calculated as $b=(b+16)/32$ and is clipped to lie in the range $[0, 2^n-1]$. As before, the operator / denotes division with truncation.").

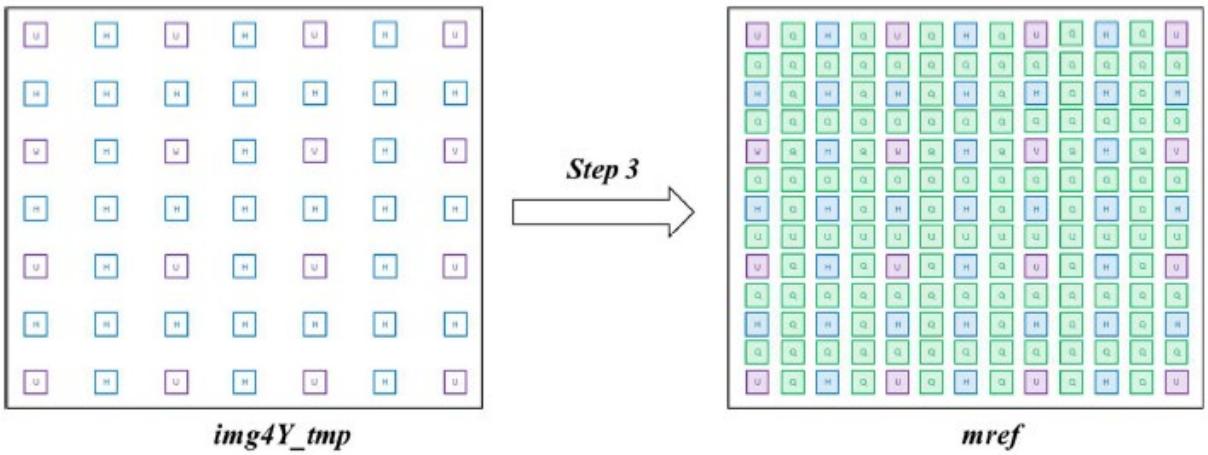


Step 2: determine intermediate interpolated values for the remaining ½ sub-pixels in **blue** at (1) unit horizontal and half unit vertical locations and (2) half unit horizontal and half unit vertical locations based on the values for the ½ sub-pixels and unit pixels stored in the data structure ***img4Y_tmp***, and store these additional ½ sub-pixel values in the data structure ***img4Y_tmp***. Also, normalize, round, and clip the intermediate interpolated values of all of the ½ sub-pixels in **blue** in the data structure ***img4Y_tmp*** (*i.e.,* all of the ½ sub-pixels for which values were interpolated in both step 1 and step 2) and store them in the ***mref*** data structure. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13c, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to

A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to $(c+512)/1024$. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").

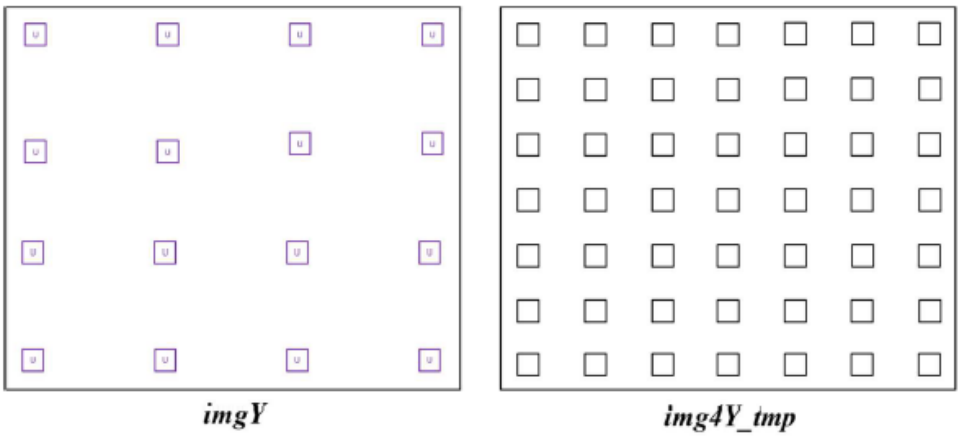


Step 3: interpolate values for all ¼ sub-pixels in **green**, based on values stored in the data structure ***img4Y_tmp***, and normalizes, round and clip the ¼ sub-pixel values, and store them in the data structure ***mref***. TML6, 58-59 (/tml6pre.zip/lencod/src/image.c, ll. 593-632); *see also* EX-1001, 11:64-12:45.



b) *Detailed Operation of the Three Steps of the Oneforthpix() Function*

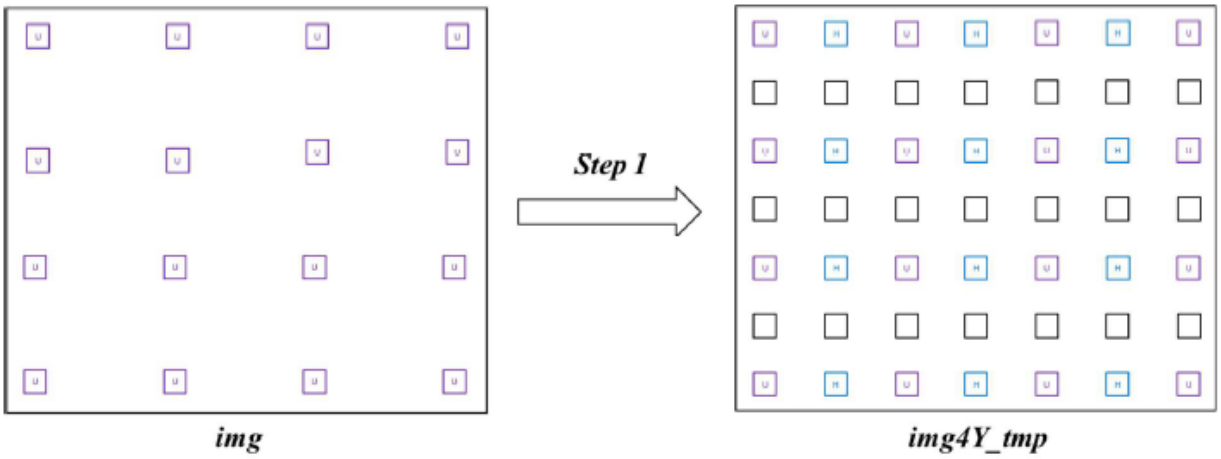
57. The first step of the ***oneforthpix()*** function uses the temporary data structure ***img4Y_tmp***, which is a "2x expansion" of the data structure ***imgY***. In other words, the data structure ***img4Y_tmp*** has an additional row and an addition column added between each respective row and column of ***imgY***. A comparison of the data structures ***imgY*** and ***img4Y_tmp*** is illustrated below. The blank boxes in **black** in the data structure ***img4Y_tmp*** indicate empty data locations.



58. The ***oneforthpix()*** function fills the data structure ***img4Y_tmp*** with the luminance values of the unit pixel in **purple**, as illustrated below. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 553-575). While ***oneforthpix()*** is filling the unit sub-pixels **purple**, it also "horizontally interpolates"[4] them to generate intermediate interpolated luminance values of ½ sub-pixels in **blue** located at the half unit horizontal and unit vertical locations, as also illustrated below. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 553-575); *see also* EX-1001, 11:35-45 ("The value for the sub-pixel at half unit horizontal and unit vertical location, that is ½ resolution sub-pixel b in FIG. 13*a*, is obtained by first calculating an intermediate value b using a six tap filter. The filter calculates b based upon the values of the 6 pixels (A_1 to A_6) situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13*b*, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value of b is then calculated as $b=(b+16)/32$ and is clipped to lie in the range $[0, 2^n-1]$. As before, the operator / denotes division with truncation."). These values of the ½ sub-pixels in **blue** are known as intermediate values because they have not been normalized, rounded, or clipped so that they are integers lying within the dynamic range of 0 to 255, *i.e.,* representable by 8 bits). TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 553-575);

[4] Further details of how these sub-pixels are interpolated are discussed below.

see also EX-1001, 11:35-45 ("The value for the sub-pixel at half unit horizontal and unit vertical location, that is ½ resolution sub-pixel b in FIG. 13*a*, is obtained by first calculating an intermediate value b using a six tap filter. The filter calculates b based upon the values of the 6 pixels (A_1 to A_6) situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13b, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value of b is then calculated as $b=(b+16)/32$ and is clipped to lie in the range $[0, 2^n-1]$. As before, the operator / denotes division with truncation.").



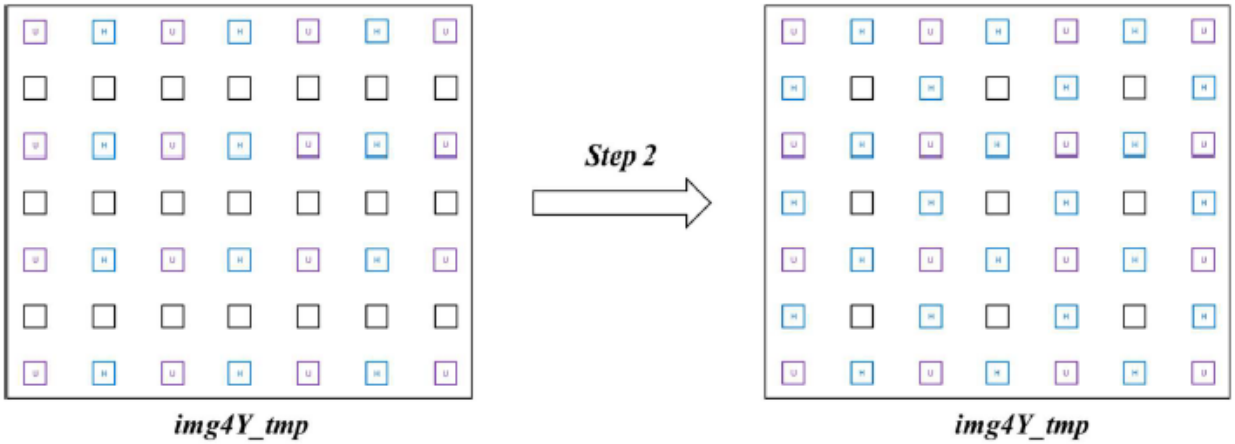
The blank (**black**) sub-pixel locations for the data structure ***img4Y_tmp*** (*i.e.,* the sub-pixels at (1) the unit horizontal and half unit vertical locations and (2) the half unit horizontal and half unit vertical locations) have not yet been interpolated.

59. In the second step, the ***oneforthpix()*** function generates the remaining ½ sub-pixels in **blue** (*i.e.,* the sub-pixels at (1) the unit horizontal and half unit vertical locations and (2) the half unit horizontal and half unit vertical locations) to

fill in the blank in **black** sub-pixel locations in the data structure ***img4Y_tmp***. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to c=(c+16)/32. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$."). The ***oneforthpix()*** function "vertically interpolates"[5] the unit pixels in **purple** generate intermediate interpolated values of ½ sub-pixels in **blue** located at the unit horizontal and half unit vertical locations, as illustrated below. TML6, 57-58

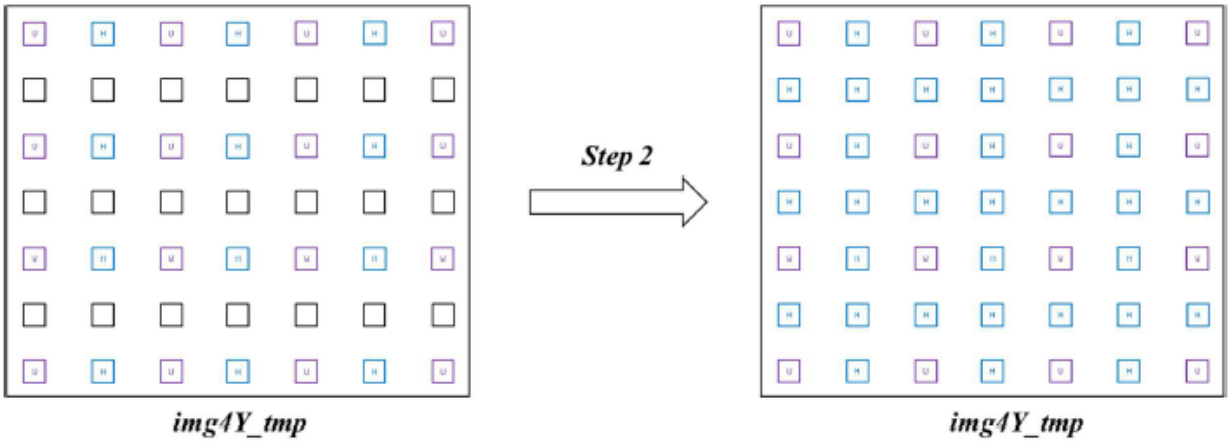
[5] Further details of how these sub-pixels are interpolated are discussed below.

(/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to $(c+512)/1024$. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").

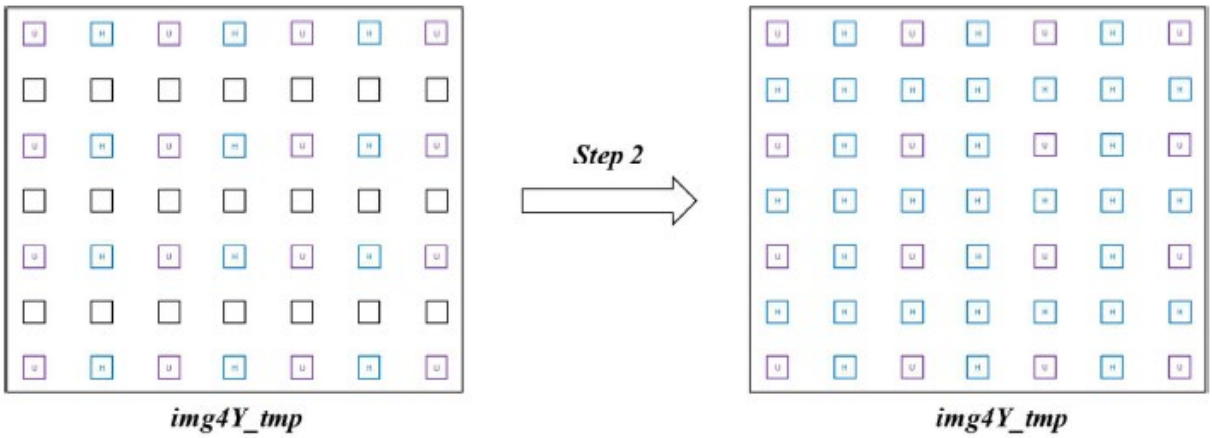


60. In the same loop, the ***oneforthpix()*** function performs "vertical interpolation" with the sub-pixels in **blue** located at the half unit horizontal and unit vertical locations to generate intermediate interpolated values of the ½ sub-pixels in **blue** located at the half unit horizontal and half unit vertical locations (the remaining black sub-pixels illustrated above). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to $(c+512)/1024$. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$."). These intermediate

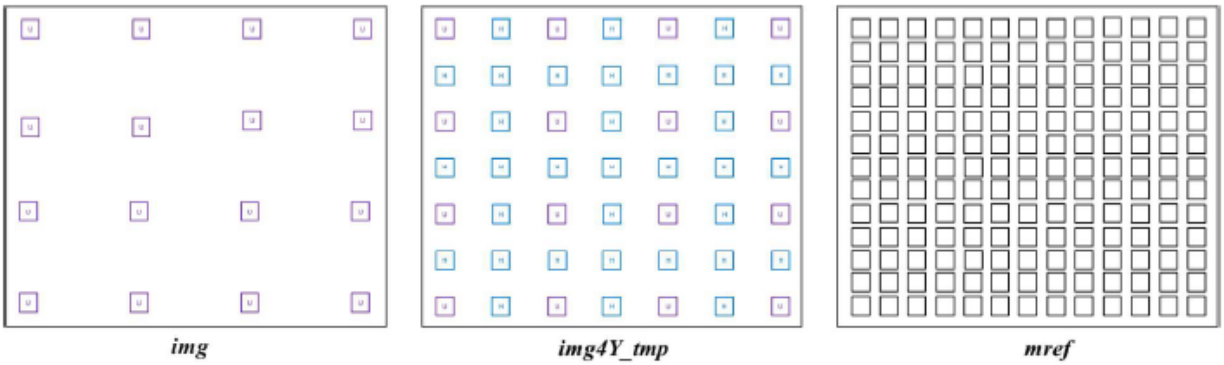
values for sub-pixels at half unit horizontal and half unit vertical are stored in the data structure ***img4Y_tmp***, as illustrated below.[6]



[6] While the above and below figures shows sub-pixels in **blue** located at the ½ unit horizontal and unit vertical locations and sub-pixels in **blue** located at the ½ unit horizontal and ½ unit vertical locations separately, they are in fact performed at the same time. That is, step 2 of the ***oneforthpix()*** function includes lines of code that interpolate ***both*** the sub-pixels in **blue** located at the ½ unit horizontal and unit vertical locations and sub-pixels in **blue** located at the ½ unit horizontal and ½ unit vertical locations.



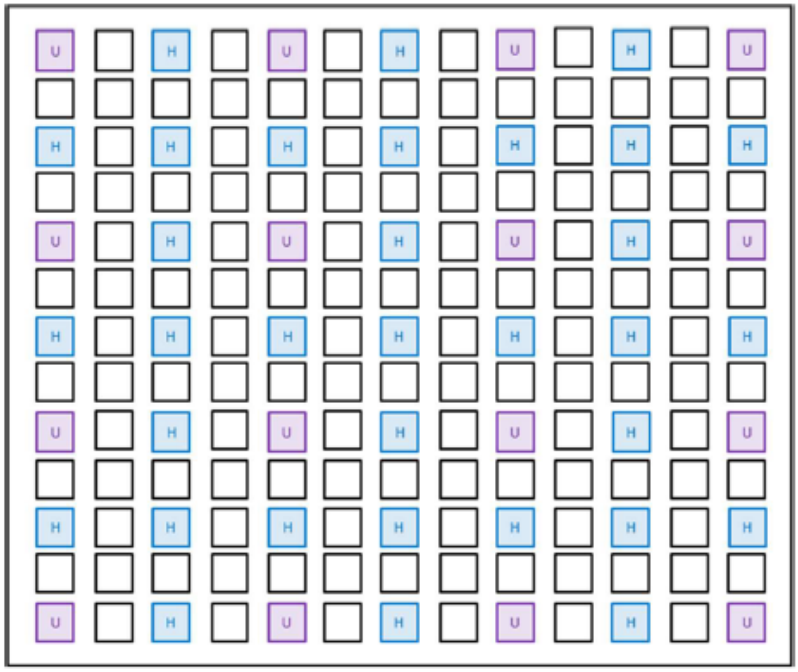
61. In addition, the second step uses the data structure ***mref***. As I discuss above, the data structure ***mref*** includes three additional rows and three additional columns between the respective rows and columns of pixels in **purple** in the data structure ***imgY***. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592). Put another way, the ***mref*** data structure includes an additional row and an additional column between respective rows and columns of the data structure ***img4Y_tmp***, as illustrated below. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).



As above, the blank boxes (**black**) in the data structure ***mref*** indicate empty data locations.

62. The second step of ***oneforthpix()*** also stores all of the ½ sub-pixels in **blue** in the data structure ***mref***, as illustrated below. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13c, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to $(c+512)/1024$. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$."). However, before the values of ½ sub-pixels in **blue** in the data structure ***img4Y_tmp*** are stored in the data structure ***mref***, they are normalized, rounded, and clipped so that the values are integer values lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63

("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to $(c+512)/1024$. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").

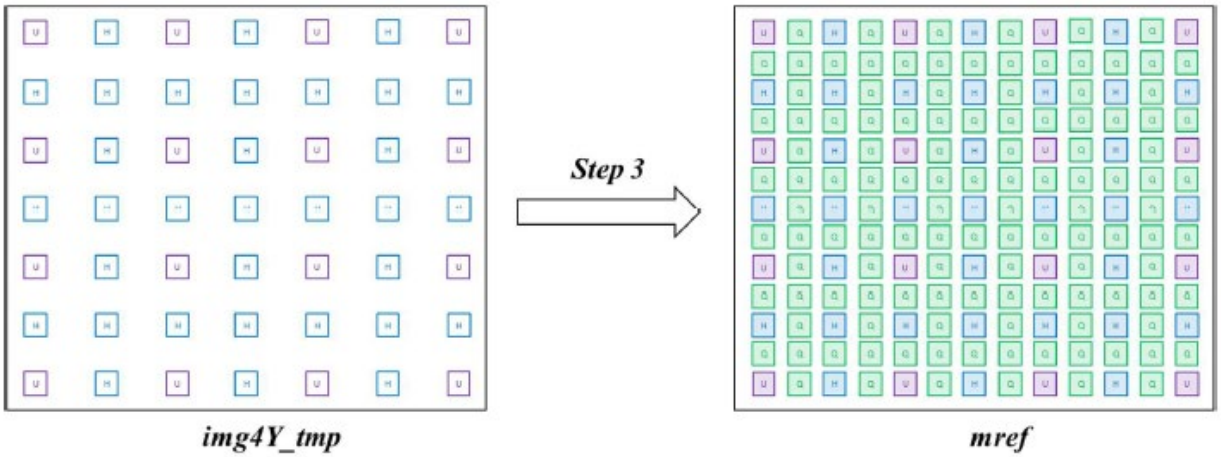


mref

As above, the blank boxes in black in the data structure ***mref*** indicate empty data locations, some of which are used for values of ¼ sub-pixels in the third step.

63. In the third step, the ***oneforthpix()*** function generates the ¼ sub-pixels in **green** and stores them in the data structure ***mref***, as illustrated below. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:45. As I discuss in further detail below, these ¼ sub-pixels in **green** are interpolated using combinations of unit pixels in **purple** and/or ½ sub-pixels in **blue**, and normalized, rounded, and clipped to integer values so that they lie within the dynamic range of 0 to 255 (*i.e.,* 8 bits), as I describe in further detail, below. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001,

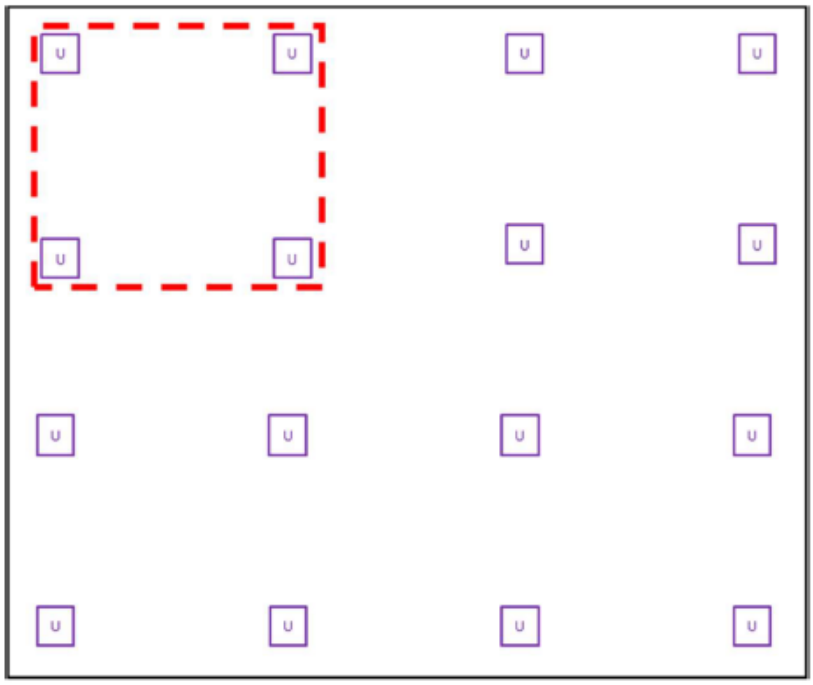
11:64-12:45. The result is that the data structure ***mref*** includes interpolated luminance values for all ½ and ¼ sub-pixels that are in the dynamic range of 0 to 255 (*i.e.,* 8 bits).



c) *Detailed Description of Interpolation of Pixels in the Oneforthpix() Function*

64. In the following paragraphs, I provide a more detailed explanation of how the ***oneforthpix()*** function performs the three main steps I introduced above to interpolate the sub-pixels for a frame of a video data stream. As I noted above, the process starts with data for the unit pixels in **purple** in the data structure ***imgY*** (illustrated below). *See* EX-1001, 11:35-45 ("The value for the sub-pixel at half unit horizontal and unit vertical location, that is ½ resolution sub-pixel b in FIG. 13*a*, is obtained by first calculating an intermediate value b using a six tap filter. The filter calculates b based upon the values of the 6 pixels (A_1 to A_6) situated in a row at unit horizontal locations and unit vertical locations symmetrically about b,

as shown in FIG. 13b, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value of b is then calculated as $b=(b+16)/32$ and is clipped to lie in the range $[0, 2^n-1]$. As before, the operator / denotes division with truncation.") .

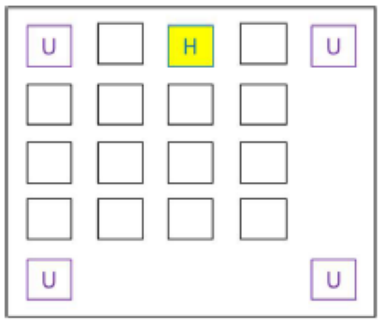


imgY

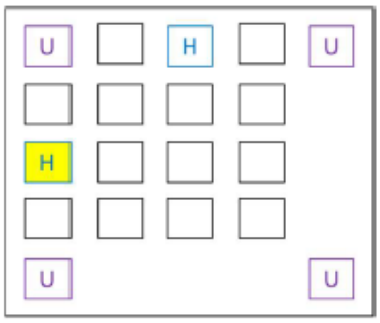
For the sake of simplicity, the following description will focus on the two rows and two columns of unit pixels in **purple** located within the dashed square (**red**) above. *See* EX-1001, 8:38-9:3, 11:25-13:18, Figures 13(a)-13(c). More specifically, as illustrated in the figures below, the following paragraphs will explain how the ***oneforthpix()*** function interpolates:

1. The highlighted ½ sub-pixel in **blue** located at the half unit horizontal and unit vertical locations.

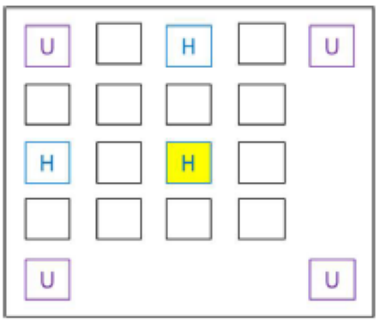
2. The highlighted ½ sub-pixel in **blue** located at the unit horizontal and half unit vertical locations.
3. The highlighted ½ sub-pixel in **blue** located at the half unit horizontal and half unit vertical locations.
4. The highlighted ¼ sub-pixels in **green** located at the quarter unit horizontal and unit vertical locations.
5. The highlighted ¼ sub-pixels in **green** located at the quarter unit horizontal and half unit vertical locations.
6. The highlighted ¼ sub-pixels in **green** located at the unit horizontal and quarter unit vertical locations.
7. The highlighted ¼ sub-pixels in **green** located at the half unit horizontal and quarter unit vertical locations.
8. The highlighted ¼ sub-pixels in **green** located at the quarter unit horizontal and quarter unit vertical locations.
9. The interpolated value of the highlighted ¼ sub-pixel in **green** located at a position, which the TML6 source code refers to as the "funny position," is replaced with a new interpolated value.



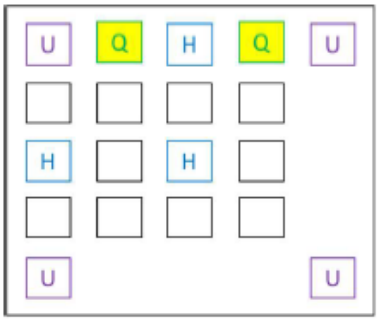
1. *½ unit horizontal unit vertical*



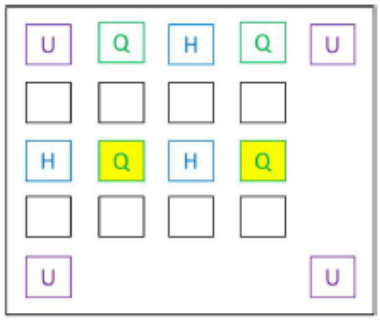
2. *unit horizontal ½ unit vertical*



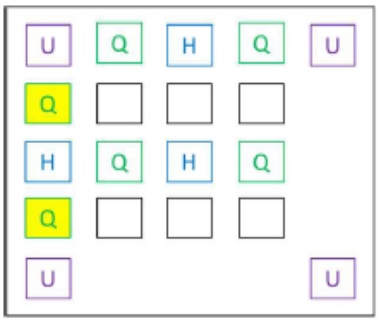
3. *½ unit horizontal ½ unit vertical*



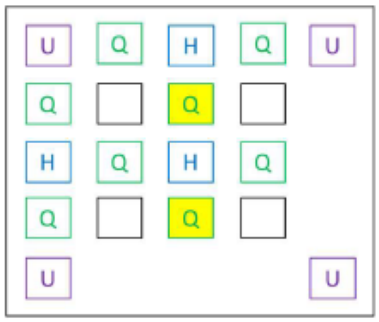
4. *¼ unit horizontal unit vertical*



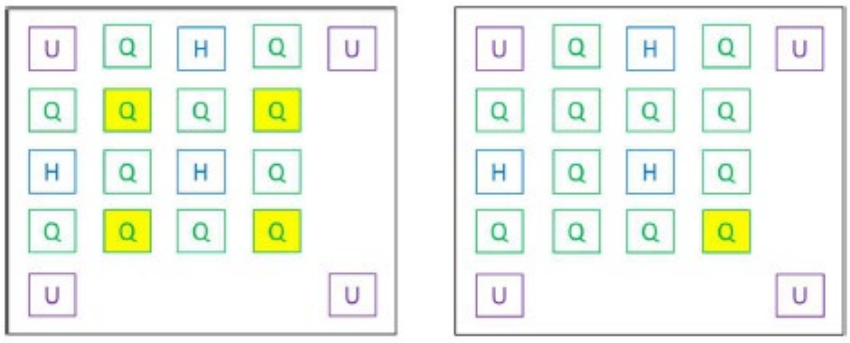
5. *¼ unit horizontal ½ unit vertical*



6. *unit horizontal ¼ unit vertical*



7. *½ unit horizontal ¼ unit vertical*

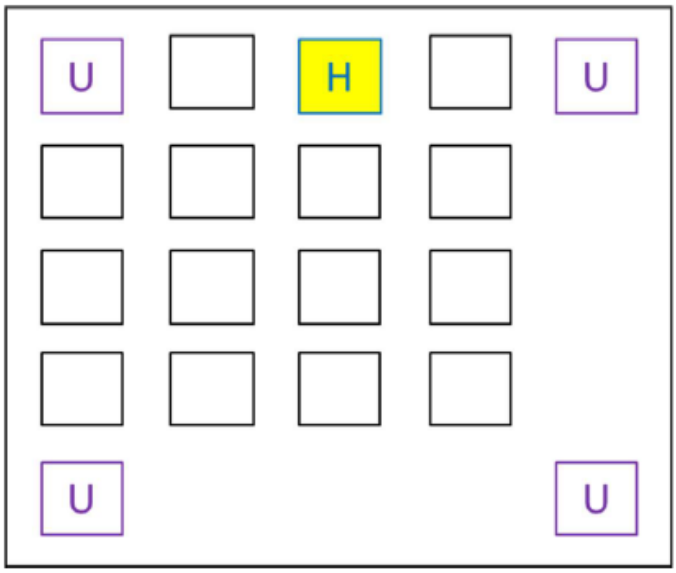


8. *¼ unit horizontal ¼ unit vertical*

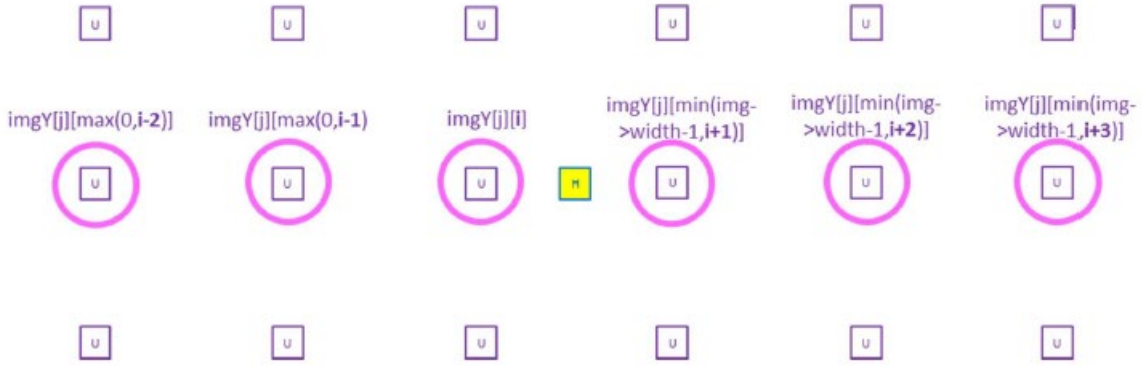
9. *"Funny Position"*

d) *Sub-Pixels at Half Unit Horizontal and Unit Vertical Locations*

65. The ½ sub-pixels lying in **blue** located at the half unit horizontal locations and unit vertical locations (such as the sub-pixel highlighted below) are interpolated in the first step of the ***oneforthpix()*** function. Then, they are normalized, rounded, and clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits), and are stored in the data structure ***mref*** in the second step of the ***oneforthpix()*** function. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 538-592); *see also* EX-1001, 11:35-45.



66. Namely, these highlighted ½ sub-pixels in **blue** are interpolated using a weighted sum "***is***" of the luminance values of the six nearest unit pixels in **purple** residing along the same horizontal row as the respective ½ sub-pixels in **blue** being interpolated. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 563-575); *see also* EX-1001, 11:35-45. For example, as illustrated below, to interpolate the highlighted ½ sub-pixel in **blue** at the half unit horizontal location and unit vertical location, the ***oneforthpix()*** function uses a weighted sum of the values of the three adjacent circled unit pixels in **purple** to the left of the highlighted sub-pixel in **blue** and the three adjacent circled unit pixels in **purple** to the right of the highlighted sub-pixel in **blue**. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 563-575); *see also* EX-1001, 11:35-45.



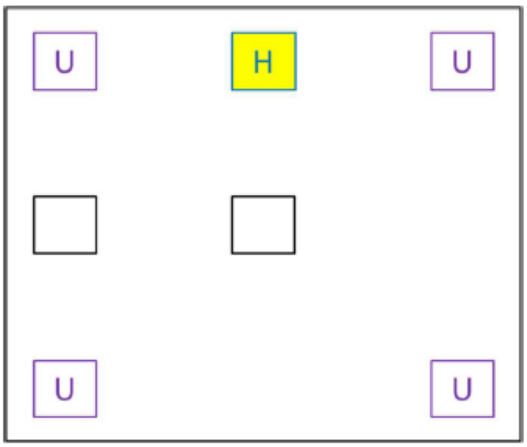

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

67. The values of the unit pixels in **purple** used to calculate the weighted sum for the highlighted sub-pixel in **blue** are retrieved from the data structure ***imgY***. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 568-570); *see also* EX-1001, 11:35-45 ("The value for the sub-pixel at half unit horizontal and unit vertical location, that is ½ resolution sub-pixel b in FIG. 13*a*, is obtained by first calculating an intermediate value b using a six tap filter. The filter calculates b based upon the values of the 6 pixels (A_1 to A_6) situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13*b*, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$.

The final value of b is then calculated as b=(b+16)/32 and is clipped to lie in the range [0, 2^n-1]. As before, the operator / denotes division with truncation.”). Moreover, the weights, *i.e.,* numbers by which each respective value in the sum is multiplied, given to the value of each unit pixel in **purple** for the weighted sum are stored in the data structure ***ONE_FOURTH_TAP***, which is defined in the file “***image.h***” in the ***lencod*** folder. TML6, 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); *see also* EX-1001, 11:35-45.

68. Then, the ***oneforthpix()*** function stores values[7] representing the weighted sums for these ½ sub-pixels in **blue** in the data structure ***img4Y_tmp*** without being normalized, rounded, or clipped to be integers lying within the range of 0 to 255 (*i.e.,* 8 bits), as shown below. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 572). Thus, these are intermediate values and not final interpolated values. *Id.*

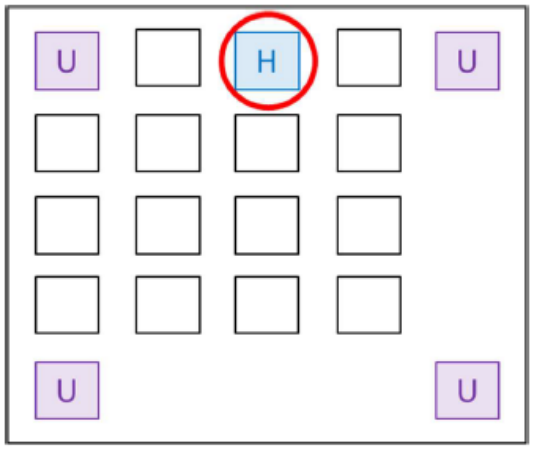
[7] These values are weighted sums themselves, simply with different weights as the value ***is*** is multiplied by 32 in line 572. *Id.*



img4Y_tmp

```
572                 img4Y_tmp[2*j][i2+1]=is*32;              /* 1/2 pix pos */
```

69. In addition, in the second step of the ***oneforthpix()*** function, the values for these sub-pixels in **blue** stored in the data structure ***img4Y_tmp*** are normalized, rounded, and clipped so that they are integers lying within the range of 0 to 255 (*i.e.,* 8-bits) and stored in the data structure ***mref***, as shown and circled below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 588); *see also* EX-1001, 11:35-45.



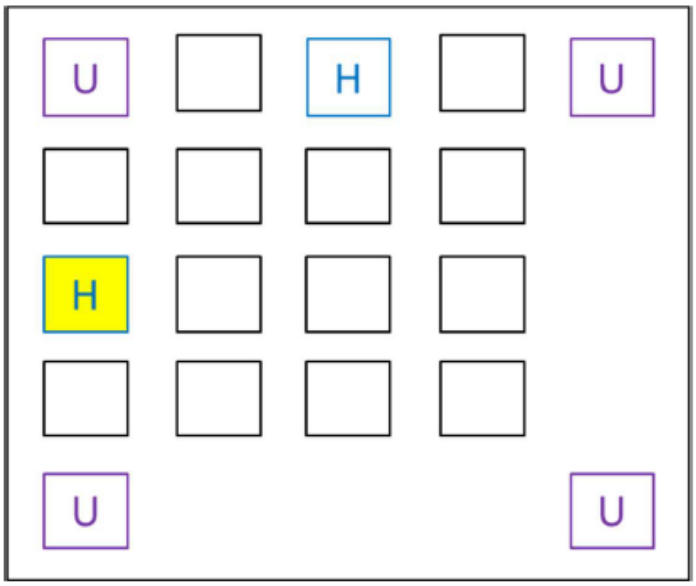
mref

```
588                 mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                    img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
```

e) *Sub-Pixels at Unit Horizontal and Half Unit Vertical Locations*

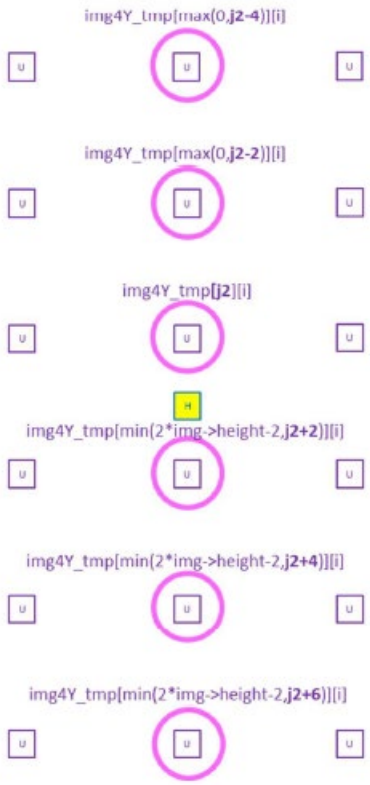
70. Afterwards, in the second main step, the ***oneforthpix()*** function interpolates the ½ sub-pixels in **blue** located at the unit horizontal locations and half unit vertical locations (such as the sub-pixel highlighted below). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13c, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the

½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to c=(c+16)/32. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").



71. These ½ sub-pixels in **blue** are interpolated using a weighted sum "***is***" of the values of the six nearest unit pixels in **purple** residing along the same vertical column as the respective sub-pixels in **blue** being interpolated. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63. For example, as illustrated below, to interpolate the highlighted ½ sub-pixel in **blue**

located at the unit horizontal location and half unit vertical location, the ***oneforthpix()*** function uses a weighted sum of the values of the three adjacent circled unit pixels in **purple** above the highlighted sub-pixel in **blue** and the three adjacent circled unit pixels in **purple** below the highlighted sub-pixel in **blue**. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63.

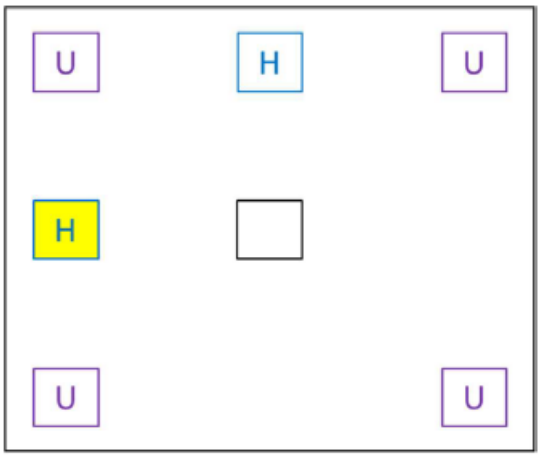


```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE FOURTH TAP[0][0]*(img4Y tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

72. Again, the values of the unit pixels in **purple** used to calculate the weighted sum for the highlighted sub-pixel in **blue** are retrieved from the data structure ***imgY***. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 583-586); *see also* EX-1001, 11:46-63. The weights given to the values of the unit pixels in **purple** are stored in the data structure ***ONE_FOURTH_TAP***, which is defined in the ***image.h*** file in the ***lencod*** folder.[8] TML6, 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); *see also* EX-1001, 11:46-63.

[8] Note that the value ***is*** also includes a factor of 1/32, which can be interpreted as either dividing a weighted sum by 32, or as being a weighted sum itself with the weights simply uniformly reduced by a factor of 32. *Id.*

73. Then, the ***oneforthpix()*** function stores the weighted sums for these ½ sub-pixels in **blue** to the data structure ***img4Y_tmp*** without being normalized, rounded, or clipped to be integers lying within the range of 0 to 255 (*i.e.,* 8 bits), as shown below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 587).

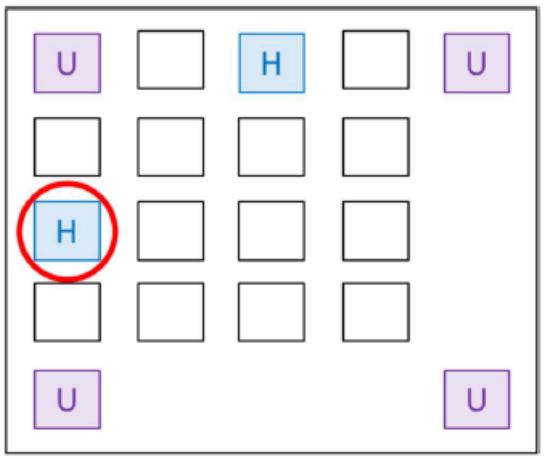


img4Y_tmp

```
587                img4Y_tmp[j2+1][i]=is;                /* 1/2 pix */
```

74. Also, in the second main step of ***oneforthpix()***, the weighted sums are normalized, rounded, and clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits), and stored in the data structure ***mref***, as shown and circled below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 589); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13c, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6)

situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").

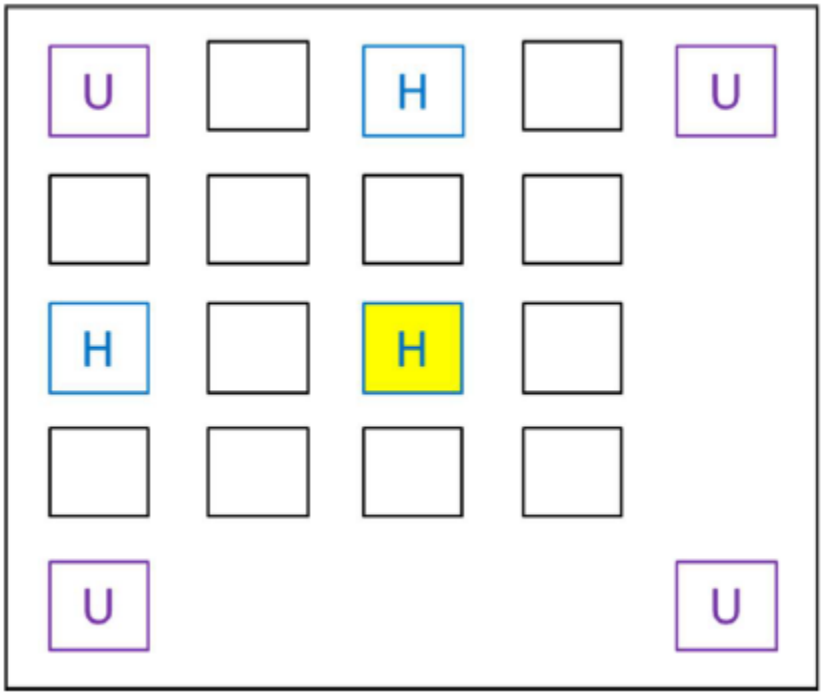


mref

```
589             mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                512)/1024))); /* 1/2 pix */
```

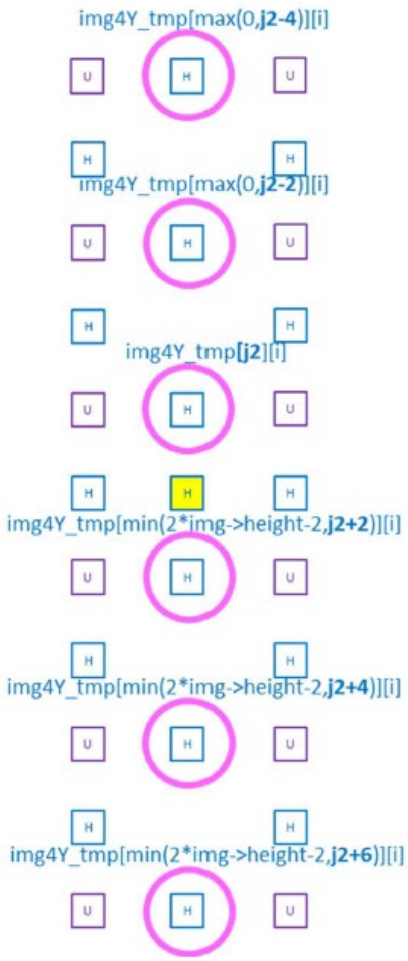
f) *Sub-pixels at Half Unit Horizontal and Half Unit Vertical Locations*

75. During the second main step, the ***oneforthpix()*** function also interpolates the ½ sub-pixels in **blue** located at the half unit horizontal locations and half unit vertical locations (such as the sub-pixel highlighted below). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to $(c+512)/1024$. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").



76. These ½ sub-pixels in **blue** are interpolated using a weighted sum "is" of values of the nearest six nearest ½ sub-pixels in **blue** residing along the same vertical column as the respective ½ sub-pixels in **blue** being interpolated. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63. For example, as I show below, to interpolate the highlighted ½ sub-pixel in **blue** (at the half unit horizontal location and half unit vertical location), the ***oneforthpix()*** function uses a weighted sum of the values of the three adjacent circled ½ sub-pixels in **blue** (located at the unit vertical and half unit horizontal locations) above the highlighted sub-pixel in **blue** and the three adjacent circled ½ sub-pixels in **blue** (located at the unit vertical and half unit horizontal locations)

below the highlighted sub-pixel in **blue**. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592); *see also* EX-1001, 11:46-63.



```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

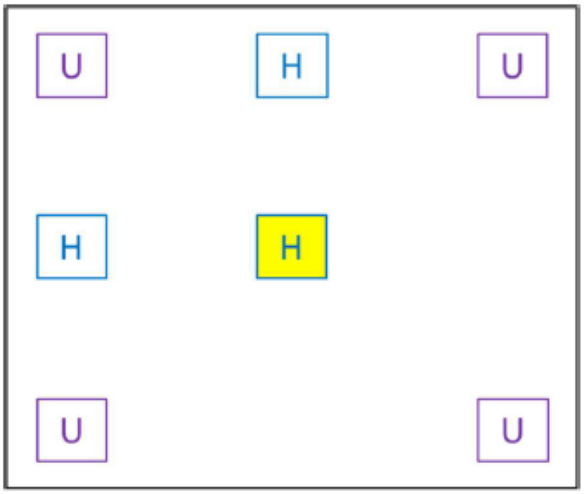
77. The values of the ½ sub-pixels in **blue** (located at the unit vertical and half unit horizontal locations) used to calculate the weighted sum are retrieved from the data structure ***img4Y_tmp***. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 583-585); *see also* EX-1001, 11:46-63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to

c=(c+16)/32. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range [0, 2^n-1].”). That is, they are based on intermediate values for these sub-pixels that have not been normalized, rounded, or clipped, as I discuss above. The weights given to these ½ sub-pixels are stored in the data structure ***ONE_FOURTH_TAP***, which is defined in the ***image.h*** file in the ***lencod*** folder.[9] *Id.*; 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); *see also* EX-1001, 11:46-63.

78. Furthermore, the weighted sums for the highlighted ½ sub-pixels in **blue** in the figure shown below at the half unit vertical and half unit horizontal locations are stored in the data structure ***img4Y_tmp*** without being normalized,

[9] As I discussed above with respect to the sub-pixels located at unit horizontal and half unit vertical, the value ***is*** also includes a factor of 1/32, which can be interpreted as either dividing a weighted sum by 32, or as being a weighted sum itself with the weights simply uniformly reduced by a factor of 32. *Id.*

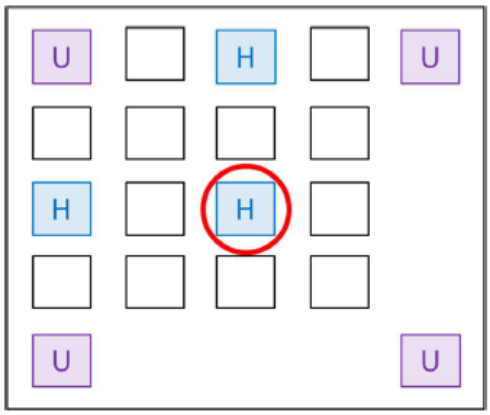
rounded, or clipped to lie within the range of 0 to 255 (*i.e.,* 8 bits). TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 587).



img4Y_tmp

```
587                img4Y_tmp[j2+1][i]=is;                /* 1/2 pix */
```

79. After the weighted sums for these ½ sub-pixels in **blue** are stored in the data structure ***img4Y_tmp***, they are normalized, rounded, and clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits) and stored in the data structure ***mref***, as shown and circled below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 589); *see also* EX-1001, 11:46-63.



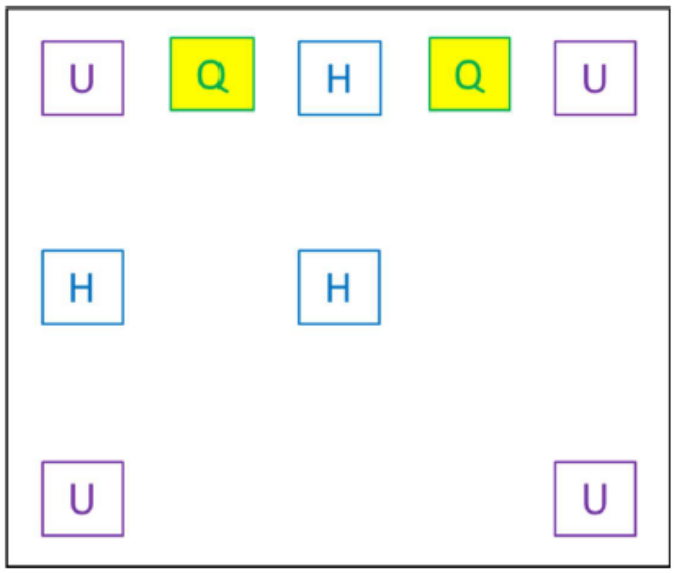
mref

```
589             mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                512)/1024))); /* 1/2 pix */
```

g) *Sub-Pixels at Quarter Unit Horizontal and Unit Vertical Locations*

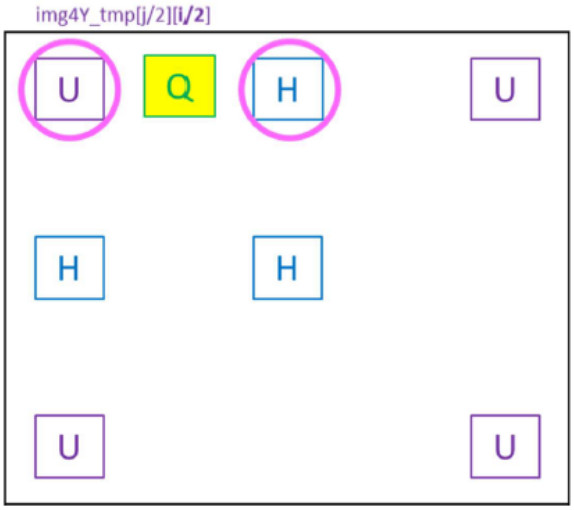
80. During the third main step, the ***oneforthpix()*** function interpolates the ¼ sub-pixels in **green** located at the quarter unit horizontal locations and unit vertical locations (such as the sub-pixels highlighted below). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14 ("Values for the ¼ resolution sub-pixels labelled d are calculated as follows. Values for ¼ resolution sub-pixels d located at quarter horizontal and unit vertical locations, are calculated from the value of the immediately neighbouring pixel at unit horizontal and unit vertical location (pixel A) and the intermediate value b calculated in step (1) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and unit vertical location (½ resolution sub-pixel b), according to d=(32A+b+32)/64. Values for ¼ resolution sub-pixels d located at quarter

horizontal and half vertical locations, are interpolated using the intermediate values c calculated for the immediately neighbouring ½ resolution sub-pixels c which lie at unit horizontal and half vertical location and half horizontal and half vertical locations respectively, according to $d=(32c_1, +c_2+1024)/2048$. Again operator / indicates division with truncation and the finally obtained ¼ resolution sub-pixel values d are clipped to lie in the range $[0, 2^n-1]$.").



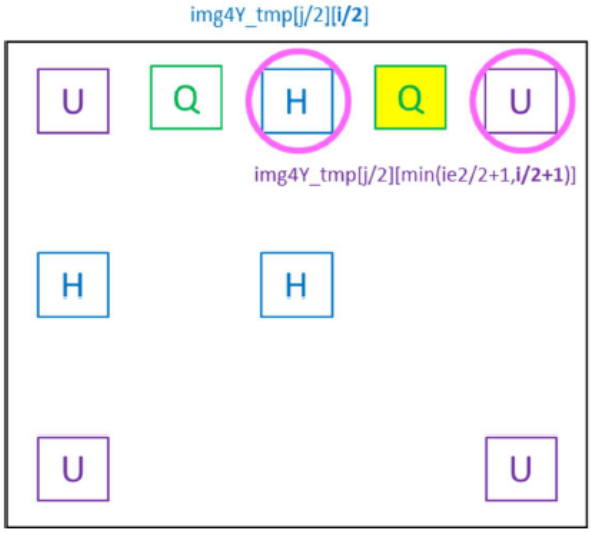
81. These ¼ sub-pixels in **green** are interpolated using a weighted average of values of the nearest unit pixel in **purple** and the nearest ½ sub-pixel in **blue** residing along the same horizontal row. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14. For example, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled unit pixel in **purple** and the adjacent circled ½ sub-pixel in **blue** located at

the unit vertical and half unit horizontal locations. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14.



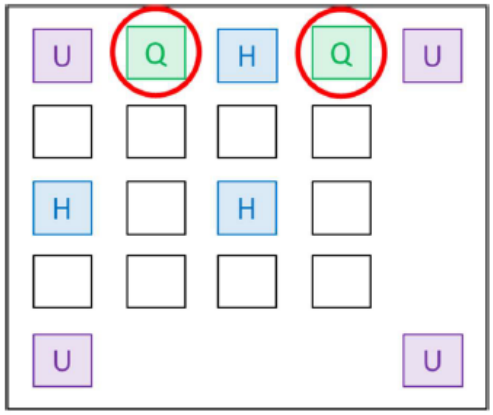

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
```

82. Similarly, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled unit pixel in **purple** and the adjacent circled ½ sub-pixel in **blue** located at the unit vertical and half unit horizontal locations). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
```

83. The values of the unit pixels in **purple** and the ½ sub-pixels in **blue** used to calculate the weighted averages of these ¼ sub-pixels in **green** are retrieved from the data structure ***img4Y_tmp***. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 600); *see also* EX-1001, 11:64-12:14. After the weighted averages are calculated, they are normalized, rounded, and clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits), and stored in the data structure ***mref***, as shown and circled below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 600); *see also* EX-1001, 11:64-12:14.



mref

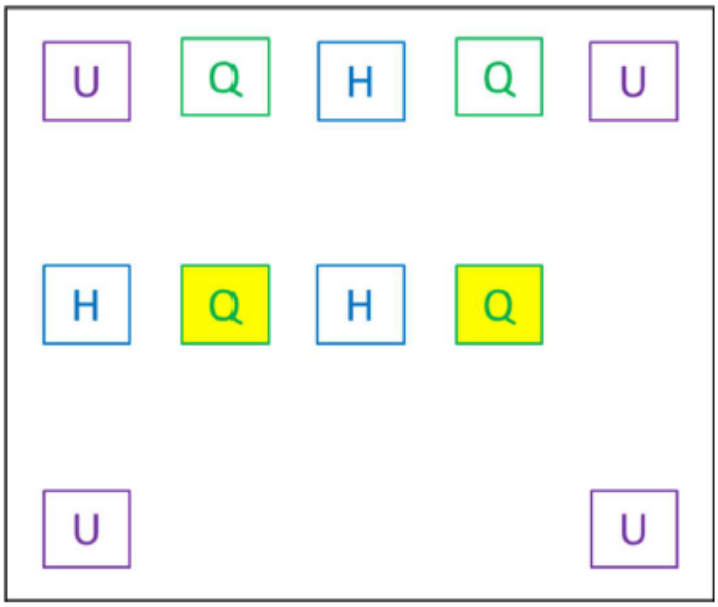
```
593      /* 1/4 pix */
594      /* luma */
595      ie2=(img->width-1)*4;
596      je2=(img->height-1)*4;
597
598      for (j=0;j<je2+4;j+=2)
599        for (i=0;i<ie2+3;i+=2)
600          mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
           ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
```

h) *Sub-pixels at Quarter Unit Horizontal and Half Unit Vertical Locations*

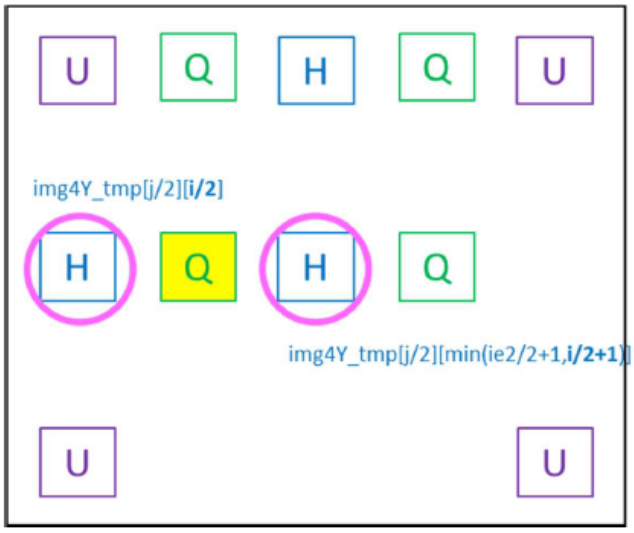
84. Also, during the third main step, the ***oneforthpix()*** function interpolates the ¼ sub-pixels in **green** located at the quarter unit horizontal locations and half unit vertical locations (such as the highlighted sub-pixels below).[10] TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14 ("Values for the ¼ resolution sub-pixels labelled d are calculated as follows. Values for ¼ resolution sub-pixels d located at quarter

[10] As discussed further, below, these sub-pixels are interpolated alongside sub-pixels at ¼ unit horizontal and unit vertical locations.

horizontal and unit vertical locations, are calculated from the value of the immediately neighbouring pixel at unit horizontal and unit vertical location (pixel A) and the intermediate value b calculated in step (1) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and unit vertical location (½ resolution sub-pixel b), according to d=(32A+b+32)/64. Values for ¼ resolution sub-pixels d located at quarter horizontal and half vertical locations, are interpolated using the intermediate values c calculated for the immediately neighbouring ½ resolution sub-pixels c which lie at unit horizontal and half vertical location and half horizontal and half vertical locations respectively, according to $d=(32c_1, +c_2+1024)/2048$. Again operator / indicates division with truncation and the finally obtained ¼ resolution sub-pixel values d are clipped to lie in the range $[0, 2^n-1]$.").

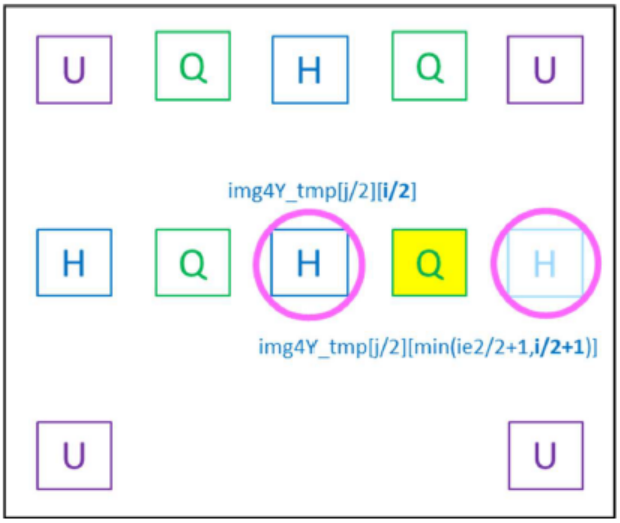


85. These ¼ sub-pixels in **green** are interpolated using a weighted average of the values of the nearest two ½ sub-pixels in **blue** residing along the same horizontal row. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14. For example, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled ½ sub-pixel in **blue** located at the half unit vertical and unit horizontal locations and the adjacent circled ½ sub-pixel in **blue** located at the half unit vertical and half unit horizontal locations. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
```

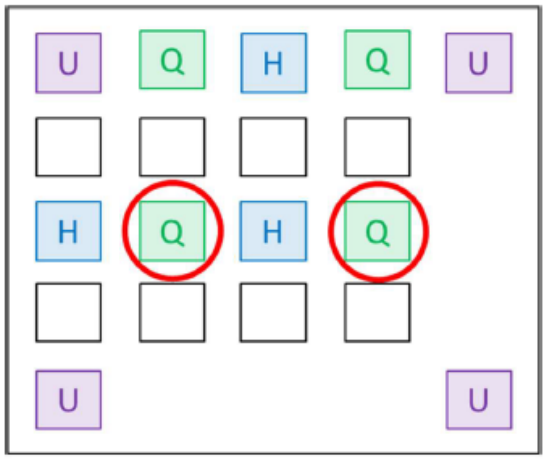
86. Similarly, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled ½ sub-pixel in **blue** located at the half unit vertical and half unit horizontal locations and the adjacent circled ½ sub-pixel in **blue** located at the half unit vertical and unit horizontal locations. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 11:64-12:14.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
```

87. The values of the ½ sub-pixels in **blue** used to calculate the weighted averages of these ¼ sub-pixels in **green** are retrieved from the data structure ***img4Y_tmp***. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 600); *see also* EX-1001, 11:64-12:14. Then, the weighted averages are normalized, rounded, and

clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.*, 8 bits), and stored in the data structure ***mref***, as shown and circled below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 600); *see also* EX-1001, 11:64-12:14.



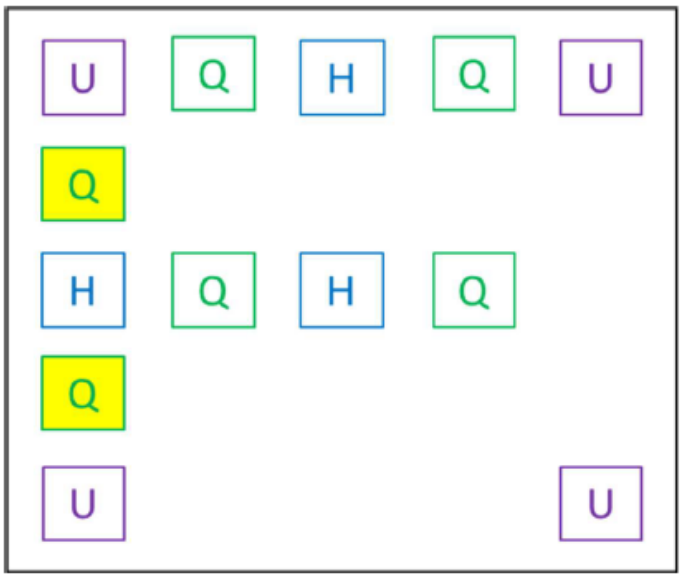
mref

```
593       /* 1/4 pix */
594       /* luma */
595       ie2=(img->width-1)*4;
596       je2=(img->height-1)*4;
597
598       for (j=0;j<je2+4;j+=2)
599         for (i=0;i<ie2+3;i+=2)
600           mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
            ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
```

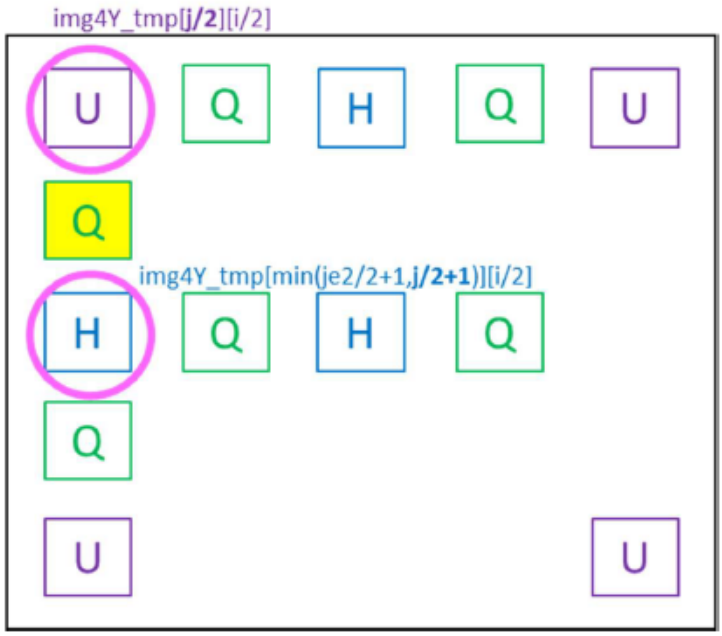
i) *Sub-pixels at Unit Horizontal and Quarter Unit Vertical Locations*

88. During the third main step, the ***oneforthpix()*** function interpolates the ¼ sub-pixels located at the unit horizontal locations and quarter unit vertical locations (such as the highlighted sub-pixels below). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33 ("Values for the ¼ resolution sub-pixels labelled e are calculated as follows.

Values for ¼ resolution sub-pixels e located at unit horizontal and quarter vertical locations are calculated from the value of the immediately neighbouring pixel at unit horizontal and unit vertical location (pixel A) and the intermediate value c calculated in step (2) for the immediately neighbouring ½ resolution sub-pixel at unit horizontal and half vertical location, according to e=(32A+c+32)/64. Values for ¼ resolution sub-pixels e located at half horizontal and quarter vertical locations are calculated from the intermediate value b calculated in step (1) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and unit vertical location and the intermediate value c calculated in step (2) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and half vertical location, according to e=(32b+c+1024)/2048. Once more, operator / indicates division with truncation and the finally obtained ¼ resolution sub-pixel values e are clipped to lie in the range [0, 2^n-1].").



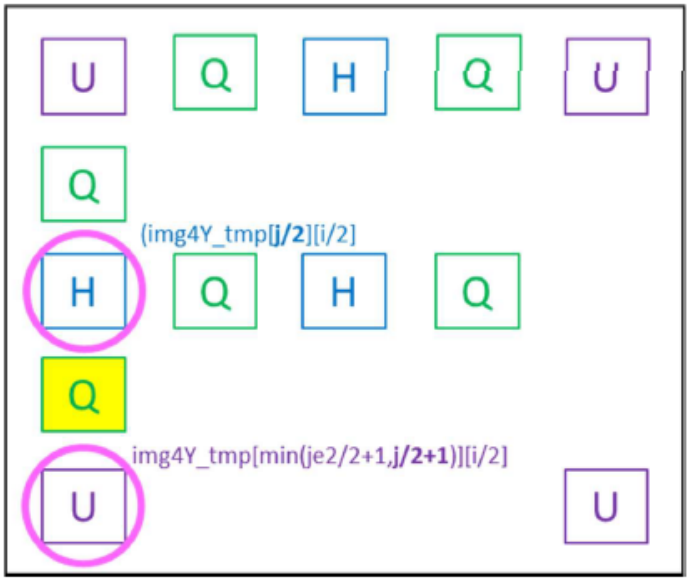
89. These ¼ sub-pixels in **green** are interpolated using a weighted average of values of the nearest unit pixel in **purple** and ½ sub-pixel in **blue** residing along the same vertical column. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33. For example, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled unit pixel in **purple** and the adjacent circled ½ sub-pixel in **blue** located at the unit horizontal and half unit vertical locations. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
```

90. Similarly, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses weighted average of the values of the adjacent circled ½ sub-pixel in **blue** located at the unit horizontal and half unit

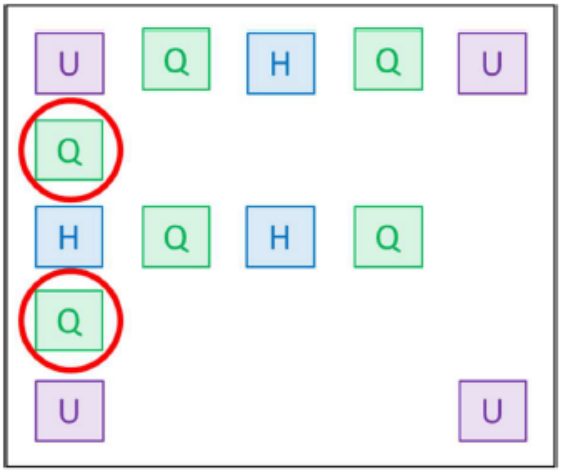
vertical locations and adjacent circled unit pixel in **purple**. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
```

91. The values of the unit pixels in **purple** and the ½ sub-pixels in **blue** used to calculate the weighted averages of these ¼ sub-pixels in **green** are retrieved from the data structure ***img4Y_tmp***. TML6, 58

(/tml6pre.zip/lencod/src/image.c, l. 607); *see also* EX-1001, 12:15-33. Similar to the cases I explain above, the weighted averages are normalized, rounded, and clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits), and stored in the data structure ***mref***, as shown and circled below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 607); *see also* EX-1001, 12:15-33.



mref

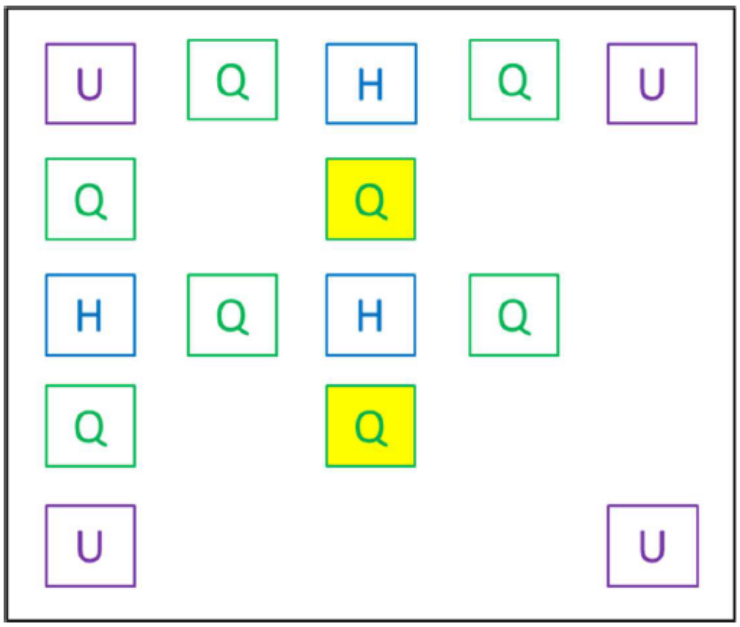
```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
```

j) *Sub-pixels at Half Unit Horizontal and Quarter Unit Vertical Locations*

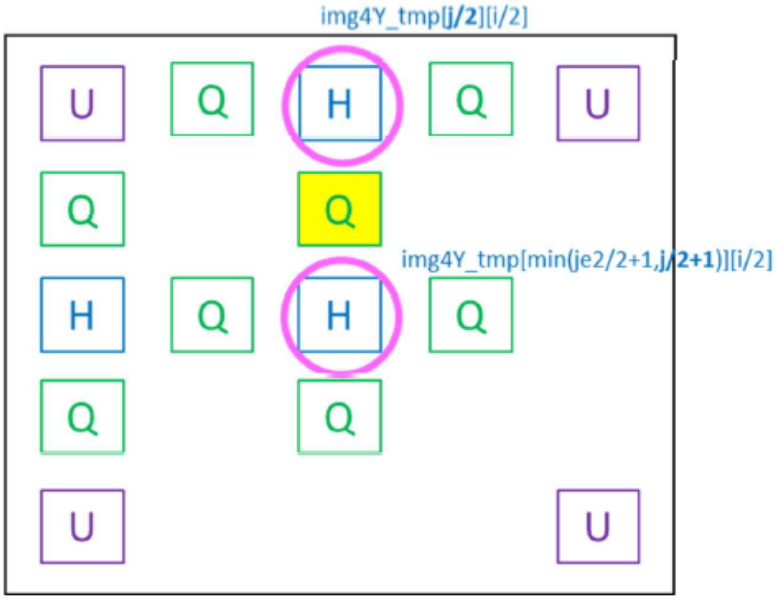
92. During the third main step, the ***oneforthpix()*** function interpolates the ¼ sub-pixels in **green** located at the half unit horizontal and quarter unit vertical locations (such as the highlighted sub-pixels below).[11] TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33 ("Values for the ¼ resolution sub-pixels labelled e are calculated as follows. Values for ¼ resolution sub-pixels e located at unit horizontal and quarter vertical locations are calculated from the value of the immediately neighbouring pixel at unit horizontal and unit vertical location (pixel A) and the intermediate value c calculated in step (2) for the immediately neighbouring ½ resolution sub-pixel at unit horizontal and half vertical location, according to e=(32A+c+32)/64. Values for ¼ resolution sub-pixels e located at half horizontal and quarter vertical locations are calculated from the intermediate value b calculated in step (1) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and unit vertical location and the intermediate value c calculated in step (2) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and half

[11] As discussed further, below, these sub-pixels are interpolated alongside sub-pixels at unit horizontal and ¼ unit vertical locations.

vertical location, according to e=(32b+c+1024)/2048. Once more, operator / indicates division with truncation and the finally obtained ¼ resolution sub-pixel values e are clipped to lie in the range [0, 2^n−1].").



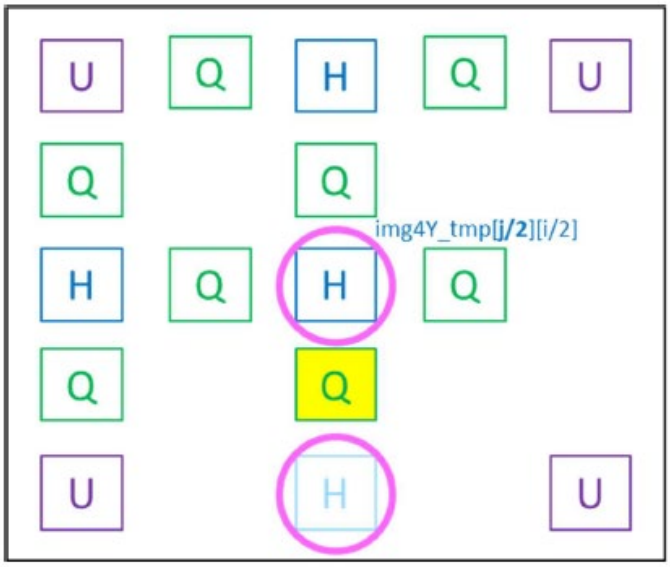
93. These ¼ sub-pixels in **green** are interpolated using a weighted average of values of the nearest two ½ sub-pixels in **blue** residing along the same vertical column. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33. For example, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled ½ sub-pixel in **blue** located at the unit vertical and half unit horizontal locations and the adjacent circled ½ sub-pixel in **blue** located at the half unit vertical and half unit horizontal locations. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
```

94. Similarly, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled ½ sub-pixel in **blue** located at the half unit vertical and half unit horizontal locations and the adjacent circled ½ sub-pixel in **blue** located at the unit

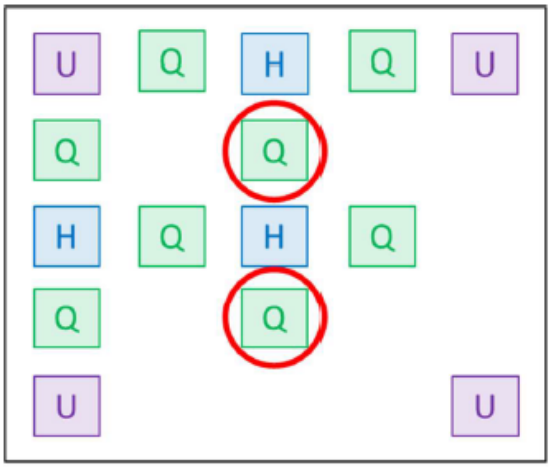
vertical and half unit horizontal locations. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
```

95. The values of the two ½ sub-pixels in **blue** used to calculate the weighted averages of these ¼ sub-pixels in **green** are retrieved from the data structure ***img4Y_tmp***. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 607); *see also* EX-1001, 12:15-33. As in the cases above, the weighted averages are normalized, rounded, and clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits), and stored in the data structure ***mref***, as shown and circled

below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, l. 607); *see also* EX-1001, 12:15-33.



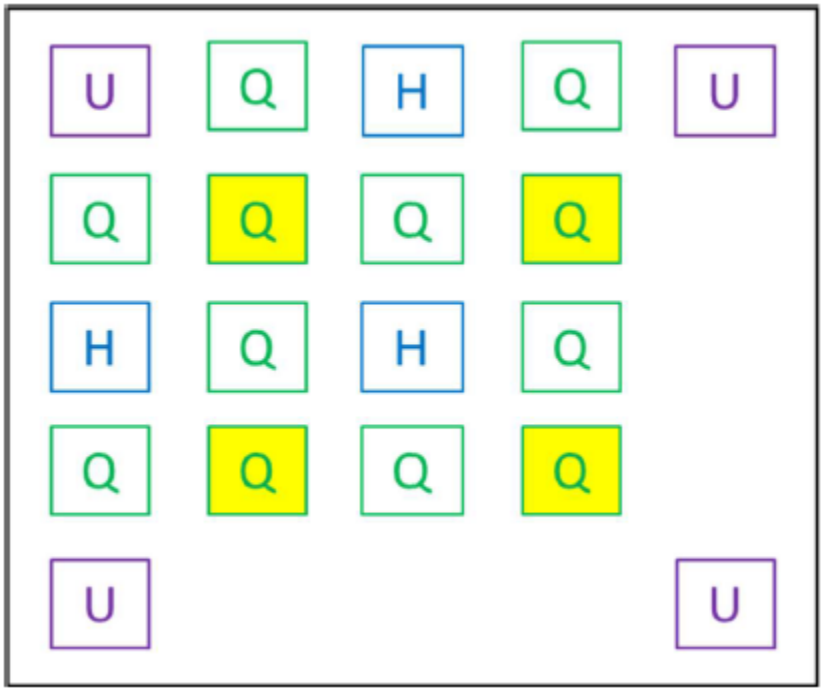
mref

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
```

k) *Sub-pixels at Quarter Unit Horizonal and Quarter Unit Vertical Locations*

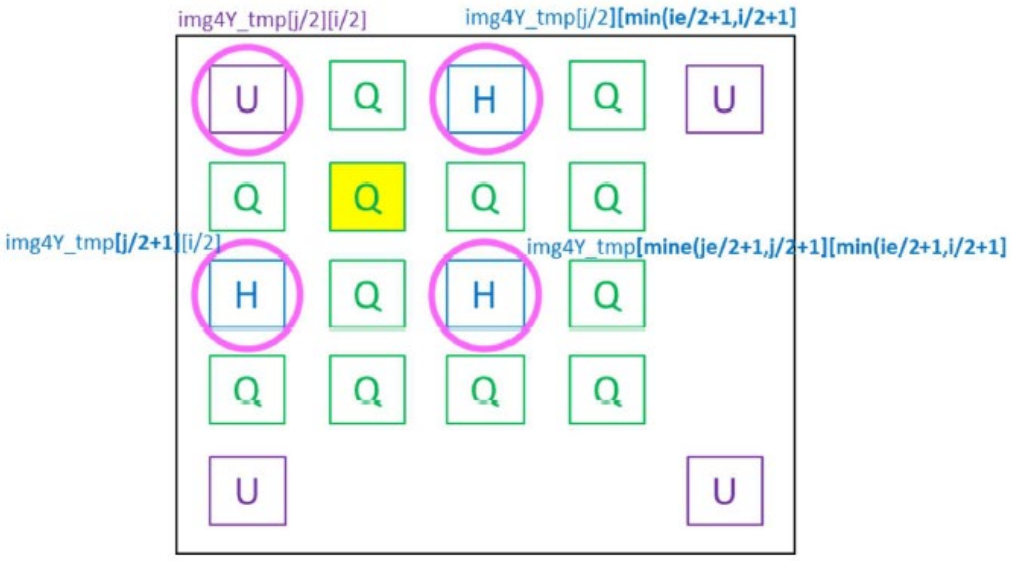
96. In step three, the ***oneforthpix()*** function also interpolates the ¼ sub-pixels in **green** located at the quarter unit horizontal locations and quarter unit

vertical locations (such as the highlighted sub-pixels below). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33 ("Values for the ¼ resolution sub-pixels labelled e are calculated as follows. Values for ¼ resolution sub-pixels e located at unit horizontal and quarter vertical locations are calculated from the value of the immediately neighbouring pixel at unit horizontal and unit vertical location (pixel A) and the intermediate value c calculated in step (2) for the immediately neighbouring ½ resolution sub-pixel at unit horizontal and half vertical location, according to e=(32A+c+32)/64. Values for ¼ resolution sub-pixels e located at half horizontal and quarter vertical locations are calculated from the intermediate value b calculated in step (1) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and unit vertical location and the intermediate value c calculated in step (2) for the immediately neighbouring ½ resolution sub-pixel at half horizontal and half vertical location, according to e=(32b+c+1024)/2048. Once more, operator / indicates division with truncation and the finally obtained ¼ resolution sub-pixel values e are clipped to lie in the range [0, 2^n-1].").



97. These ¼ sub-pixels in **green** are interpolated using a weighted average of the values of the closest unit pixel in **purple** and the three closest ½ sub-pixels in **blue**. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:15-33. For example, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled unit pixel in **purple** and the three adjacent circled ½ sub-pixels in **blue**. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:34-40 ("Values for ¼ resolution sub-pixels labelled g are computed using the value of the nearest original pixel A and the intermediate values of the three nearest neighbouring ½ resolution sub-pixels, according to

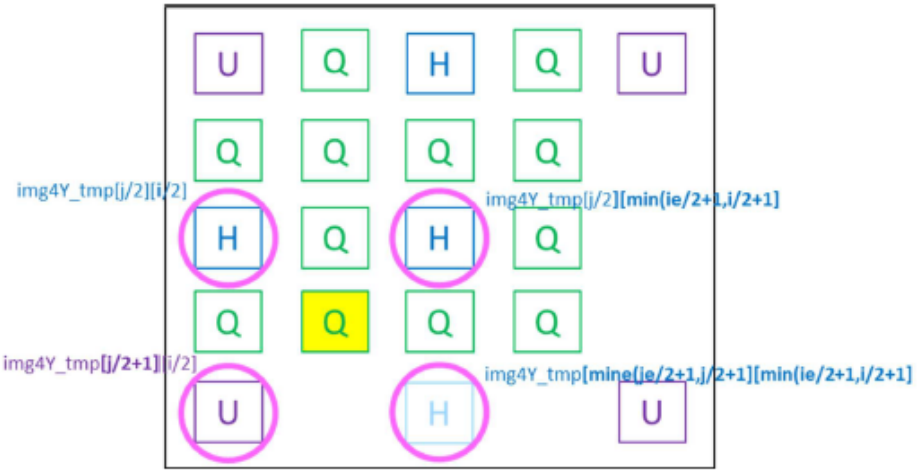
g=(1024A+32b+32c_1+c_2+2048)/4096. As before, operator / indicates division with truncation and the finally obtained for ¼ resolution sub-pixel values g are clipped to lie in the range [0, 2^n−1].").




```
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
            ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                  ][i/2                  ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                  ] +
612                       img4Y_tmp[j/2                  ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
```

98. As shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled unit pixel in **purple** and the three adjacent circled ½ sub-pixels in **blue**.

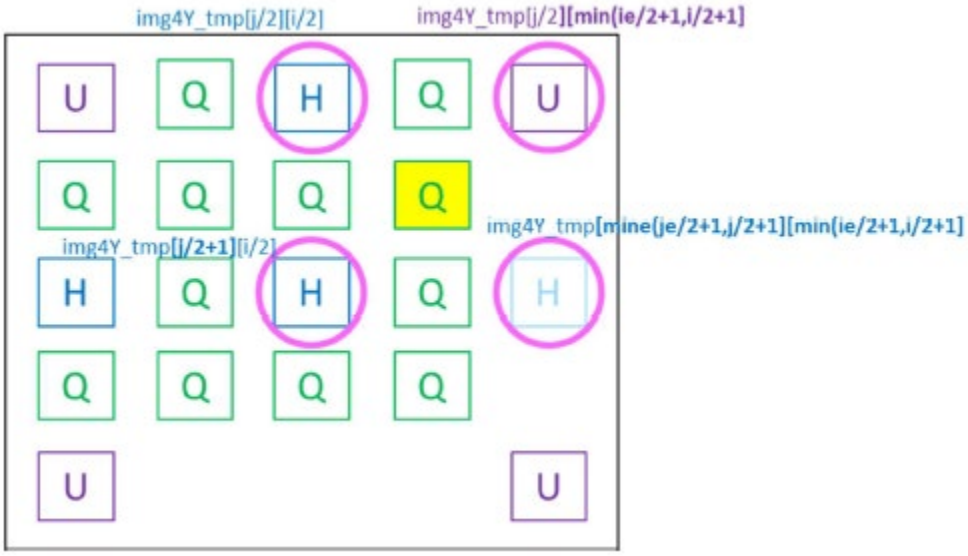
TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:34-40.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                ][i/2                ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                ] +
612                       img4Y_tmp[j/2                ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
```

99. As further shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled unit pixel in **purple** and the three adjacent circled ½ sub-pixels in

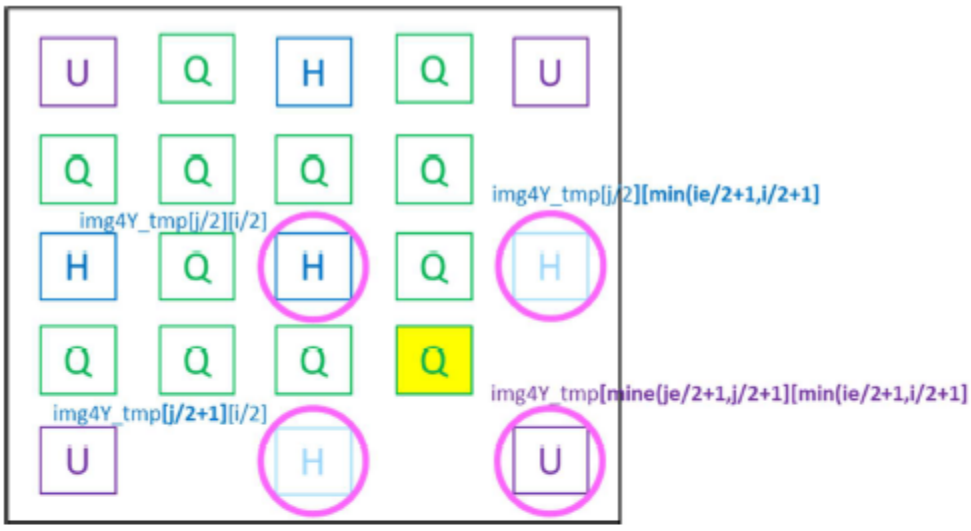
blue. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:34-40.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                ][i/2                ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                ] +
612                       img4Y_tmp[j/2                ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
```

100. As yet further shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the

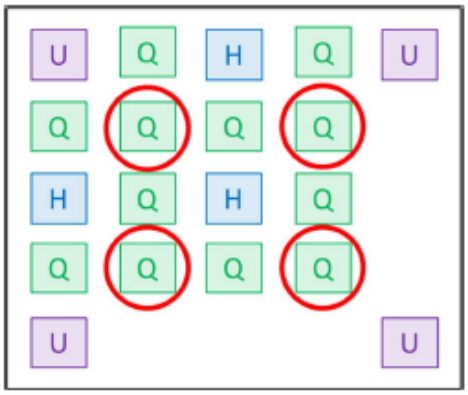
adjacent circled unit pixel in **purple** and the three adjacent circled ½ sub-pixels in **blue**. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:34-40. Later in the code, and described in the next sub-section of my declaration, the interpolated value for this ¼ sub-pixel in **green** in the data structure ***mref***, known as the "funny position," is overwritten with a value calculated in a different way. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 615-622); *see also* EX-1001, 12:41-45 ("The value for ¼ resolution sub-pixel f is interpolated by averaging the values of the 4 closest pixels at unit horizontal and vertical locations, according to $f=(A_1+A_2+A_3+A_4+2)/4$, where the locations of pixels A_1, A_2, A_3 and A_4 are the four nearest original pixels.").




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                    ][i/2                    ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                    ] +
612                       img4Y_tmp[j/2                    ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                          + 2048)/4096));
```

101. The values of the unit pixels in **purple** and the ½ sub-pixels in **blue** used to calculate the weighted averages for the ¼ sub-pixels in **green** (located at the quarter unit horizontal and quarter unit vertical locations) are retrieved from the data structure ***img4Y_tmp***. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 609-613]); *see also* EX-1001, 12:34-40 ("Values for ¼ resolution sub-pixels labelled g

are computed using the value of the nearest original pixel A and the intermediate values of the three nearest neighbouring ½ resolution sub-pixels, according to $g=(1024A+32b+32c_1+c_2+2048)/4096$. As before, operator / indicates division with truncation and the finally obtained for ¼ resolution sub-pixel values g are clipped to lie in the range $[0, 2^n-1]$.”). The weighted averages are normalized, rounded, and clipped so that they are integers lying within the dynamic range of 0 to 255 (*i.e.,* 8 bits), and stored in the data structure ***mref***, as shown and circled below. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 609-613); *see also* EX-1001, 12:34-40.



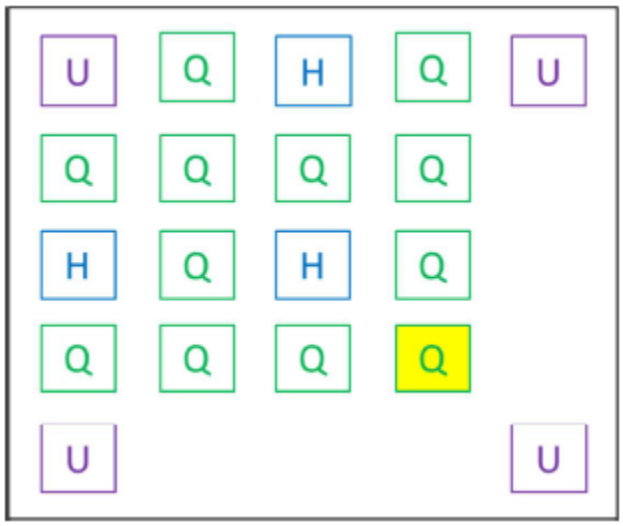
mref

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
            ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                    ][i/2                    ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                    ] +
612                       img4Y_tmp[j/2                    ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                          + 2048)/4096));
```

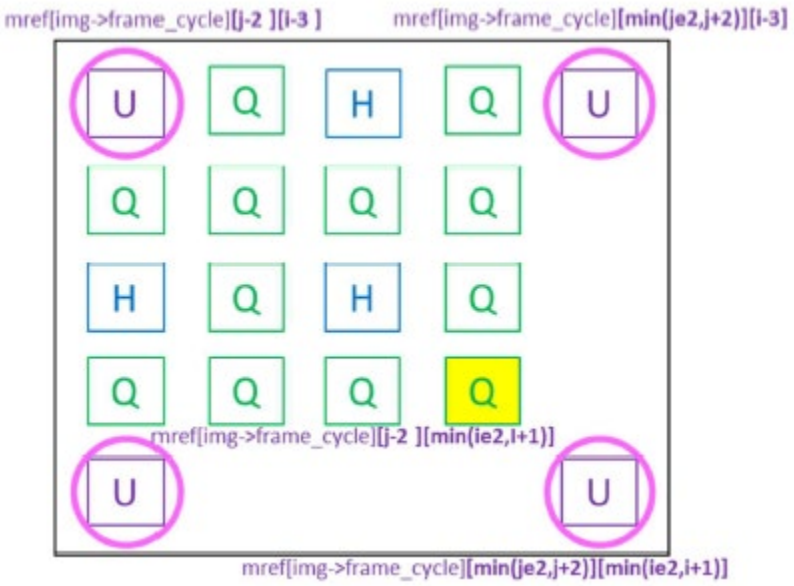
l) *Sub-pixels at Quarter Unit Horizontal and Quarter Unit Vertical "Funny Position" Locations*

102. Finally, in the third step, the ***oneforthpix()*** function also "re-interpolates" the ¼ sub-pixels in **green** located at the "funny positions" at the quarter unit horizontal and quarter unit vertical locations in the bottom right corner of the block of pixels that are being encoded (such as the highlighted sub-pixel below). TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-

1001, 12:41-45 ("The value for ¼ resolution sub-pixel f is interpolated by averaging the values of the 4 closest pixels at unit horizontal and vertical locations, according to $f=(A_1+A_2+A_3+A_4+2)/4$, where the locations of pixels A_1, A_2, A_3 and A_4 are the four nearest original pixels.").

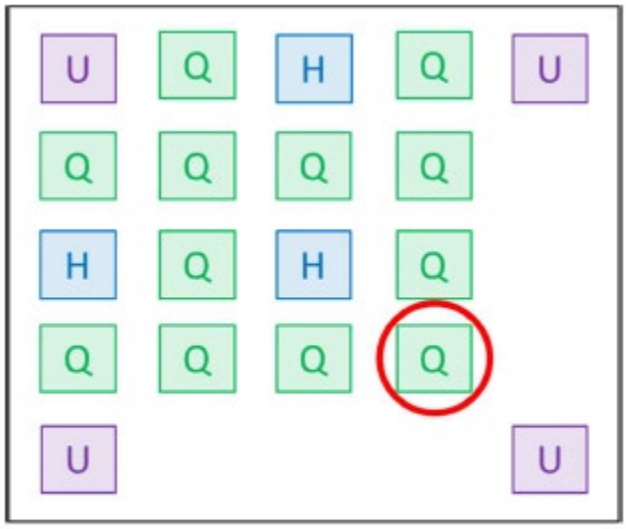


103. These ¼ sub-pixels in **green** are "re-interpolated" using a weighted average of the values of the four closest unit pixels in **purple**. *Id.* For example, as shown below, to interpolate the highlighted ¼ sub-pixel in **green**, the ***oneforthpix()*** function uses a weighted average of the values of the adjacent circled four unit pixels in **purple**. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625); *see also* EX-1001, 12:41-45.




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                ][i/2                ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                ] +
612                       img4Y_tmp[j/2                ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
614
615                   /* "funny posision" */
616                   if( ((i&3) == 3)&&(((j+1)&3) == 3))
617                   {
618                       mref[img->frame_cycle][j+1][i] = (mref[img->
                        frame_cycle][j-2          ][i-3          ] +
619                           mref[img->frame_cycle][min(je2,j+2)][i-3
                                ] +
620                           mref[img->frame_cycle][j-2          ][min(ie2,i
                            +1)] +
621                           mref[img->frame_cycle][min(je2,j+2)][min(ie2,i
                            +1)] + 2 )/4;
622                   }
```

104. The values of these four unit pixels in **purple** used to calculate the weighted averages for the ¼ sub-pixel in **green** in the "funny position" are retrieved from the data structure ***mref***. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 615-618); *see also* EX-1001, 12:41-45. Also, the weighted averages are normalized and rounded to be within the dynamic range of 0 to 255, and stored in the data structure ***mref***, as shown and circled below, over the previously interpolated ¼ sub-pixel in this location. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 618-621); *see also* EX-1001, 12:41-45.



mref

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                  ][i/2                  ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                  ] +
612                       img4Y_tmp[j/2                  ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                          + 2048)/4096));
614
615                   /* "funny posision" */
616                   if( ((i&3) == 3)&&(((j+1)&3) == 3))
617                   {
618                       mref[img->frame_cycle][j+1][i] = (mref[img->
                        frame_cycle][j-2          ][i-3          ] +
619                           mref[img->frame_cycle][min(je2,j+2)][i-3
                                    ] +
620                           mref[img->frame_cycle][j-2          ][min(ie2,i
                           +1)] +
621                           mref[img->frame_cycle][min(je2,j+2)][min(ie2,i
                           +1)] + 2 )/4;
622                   }
```

3. Motion Prediction During Decoding

105. The TML6 decoder also includes functions to perform:

- ***main()***: initializes decoding, calls functions to perform decoding, and writes output of the decoding. *See, e.g.,* TML6, 156-162 (/tml6pre.zip/lencod/src/lencod.c, ll. 75-434 ("main function for TML decoder")).

- ***decode_slice()***: called by the ***main()*** function and decodes a slice, *i.e.,* part of a frame of video data. *See, e.g.,* TML6, 137-140 (/tml6pre.zip/ldecod/src/image.c, ll. 29-177 (the decod_slice () function "decode[s] one slice")), 159-315 (/tml6pre.zip/ lencod/src/ldecod.c, ll. 79-95 (*main()* function calls the *decod_slice ()*)).
- ***macroblock()***: called by the ***decode_slice()*** function for decoding a macroblock of the frame slice. *See, e.g.,* TML6, 164-183 (/tml6pre.zip/lencod/src/macroblock.c, ll. 62-1091 (the *macroblock()* function "[d]ecode[s] one macroblock")), 139 (/tml6pre.zip/ lencod/src/image.c, ll. 148-150 (the *decode_slice()* function loops over frames and calls *macroblock()* function for decoding)).
- ***get_pix()***: called by the ***macroblock()*** function and interpolates specific sub-pixel positions needed based on motion vectors. *See, e.g.,* TML6, 150-154 (/tml6pre.zip/ldecod/src/macroblock.c, ll. 785-1022 (the *get_pix()* function performs "[d]irect interpolation of a specific subpel position")), 164-183 (/tml6pre.zip/ldecod/src macroblock.c, ll. 62-1091 (the *macroblock()* function calls the *get_pix*()function)).

- ***oneforthpix()***: interpolates sub-pixels in the decoder to ¼ pixel resolution. *See, e.g.,* TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632 (the ***oneforthpix()*** function "[u]psample[s] 4 times [*i.e.,* to ¼ pixel resolution] and store[s] in buffer [***mref***] for multiple reference frames")), 49 (tml6pre.zip/lencod/src/image.c, ll. 99-109 (the *encode_one_frame()* function calls the *oneforthpix()* function)).

4. TML6 Decoding Interpolation

106. As I discuss above, additional portions of the TML6 source code are stored in the ***ldecod*** folder. TML6. When the TML6 source code performs motion prediction to decode a video data stream, it executes the source code in a file called "***image.c***" contained in the ***ldecod*** folder. Within the ***image.c*** file, lines 784-1019 define the ***get_pixel()*** function, which is used to interpolate ½ and ¼ sub-pixels directly based on the unit pixels representing a frame when they are needed as a result of a motion vector. TML6, 150-154 (/tml6pre.zip/ldecod/src/image.c, ll. 784-1020); *see also* EX-1001, 12:46-55 ("In the decoder, sub-pixel values can be obtained directly by applying 6-tap filters in horizontal and vertical directions. In the case of ¼ sub-pixel resolution, referring to FIG. 13a, the filter coefficients applied to pixels and sub-pixels at unit vertical location are [0, 0, 64, 0, 0, 0] for a set of six pixels A, [1, −5, 52, 20, −5, 1] for a set of six sub-pixels d, [2, −10, 40,

40, −10, 2] for a set of six sub-pixels b, and [1, −5, 20, 52, −5, 1] for a set of six sub-pixels d. These filter coefficients are applied to respective sets of pixels or sub-pixels in the same row as the sub-pixel values being interpolated."). That is, each sub-pixel is calculated as a function of unit pixels, and not as a function of already interpolated sub-pixels, as I discuss above with respect to the encoder. TML6, 140-141 (/tml6pre.zip/ldecod/image.c, ll. 178-252); *see also* EX-1001, 12:46-55.

107. However, in an alternative implementation of the ***image.c*** file in the ***ldecod*** subfolder, an analogous similar function as the encoder, ***oneforthpix()***, is used to interpolate the sub-pixels in the decoder. TML6, 140-141 (/tml6pre.zip/ldecod/image.c, ll. 178-265). This function is also defined in the ***image.c*** file in the decoder subfolder, and is nearly identical to ***oneforthpix()*** as defined in ***image.c*** file in the encoder subfolder (and described above), but with differences for some sub-pixels. *Compare id., with id.,* 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 538-632).

108. The code used to call the ***oneforthpix()*** function in ***image.c*** in the decoder subfolder is commented out as an alternative implementation. *See, e.g.,* TML6, 138-139 (/tml6pre.zip/ldecod/image.c, ll. 75, 157). However, a POSITA would have readily understood how to use this alternative implementation because this code is fully functioning code. In other words, a POSITA would have readily understood to revert back to the implementation using ***oneforthpix()*** to interpolate

sub-pixels instead of the implementation using the direct interpolation in the ***get_pixel()*** embodiment, and to update decoder ***oneforthpix()*** function to interpolate sub-pixels using the same calculations as the encoder ***oneforthpix()*** function.

B. Fandrianto

109. I understand that Fandrianto published on January 14, 1997—more than four years before the '273 patent's alleged priority date—and is prior art to the '273 patent. Fandrianto, (45). Fandrianto is directed to "methods for motion vector … determination" for video compression. EX-1006, 1:7-10 ("The present invention relates generally to motion estimation, and more specifically to a programmable architecture and methods for motion vector and/or prediction error determination."), 1:12-18 ("Applications such as video telephone, digital television, and interactive multimedia using such digital storage technology as CD-ROM, digital audio tape, and magnetic disk require digital video coding, or video compression, to achieve the necessary high data transfer rates over relatively low bandwidth channels. Various standards have been proposed for video coding."), 1:43-60 ("Many video coding techniques include a predictive mode that realizes data compression between two different video frames by identifying how a frame is unlike a preceding frame. In predictive mode, the frame is represented in terms of a set of vectors of the displacement of respective groups of pixels in the frame

relative to their position in the preceding frame, known as motion vectors; and difference information representing the degree of difference between the displaced pixels and the corresponding pixels in the preceding frame. Because the amount of data in the set of motion vectors and difference information tends to be considerably less than the amount of data in the frame itself, the two frames are adequately represented by the considerably less data present in the preceding frame plus the motion vectors and difference information. When the frame is required in uncompressed form, it is reconstructed by applying the motion vectors and difference information to the preceding frame.").

110. In its "Background" section, Fandrianto refers to existing and proposed compression standards—including a predecessor ("H.261") to the H.26L standard described in the'273 patent. *Id.,* 1:12-41 ("Applications such as video telephone, digital television, and interactive multimedia using such digital storage technology as CD-ROM, digital audio tape, and magnetic disk require digital video coding, or video compression, to achieve the necessary high data transfer rates over relatively low bandwidth channels. Various standards have been proposed for video coding. A standard for the storage and transmission of still images has been adopted by the International Standards Organization ('ISO'), Joint Photographic Expert Group ('JPEG'); see 'JPEG Technical Specification, Revision 5,' JPEG-8-R5, January 1980. A standard for digital television broadcast coding at 30/45 Mb/s

is under consideration; see CCIR-CMTT/2, 'Digital Transmission of Component-Coded Television Signals at 30-34 Mb/s and 45 Mb/s Using the Discrete Cosine Transform,' Document CMTT/2-55. A standard for video telephony and video conferencing at 64 to 1920 kb/s has been adopted by the International Consultative Committee for Telephone and Telegraph ('CCITT'); see 'Draft Revision of Recommendation H.261,' Document 572, CCITT SG XV, Working Party XV/1, Spec. Grp. on Coding for Visual Telephony. A standard for storage applications below 1.5 Mb/s, which are similar to the applications targeted by the CCITT standard, is under consideration by the Moving Picture Experts Group ("MPEG") of the ISO. Video coding algorithms have been proposed as contributions to the standardization activity of ISO/MPEG; see Wong et al., 'MCPIC: A Video Coding Algorithm for Transmission and Storage Applications,' IEEE Communications Magazine, November 1990, pp. 24-32.").

111. Fandrianto then states that, "as various standards for digital video compression exist, and are proposed, a need has arisen" for improved motion estimation technology. *Id.,* 1:61-67 ("Because effective video coding requires the intimate integration of digital video compression technology, integrated circuit technology, and digital storage media, and as various standards for digital video compression exist and are proposed, a need has arisen for a flexible, high

performance, low implementation cost programmable architecture for motion estimation.").

112. Fandrianto addresses that need. *Id.,* 2:3-39 ("The present invention is advantageous in many respects. For example, the programmability aspect of the present invention enables support of future algorithms, and allows the addition of customer-proprietary optimizations and algorithms. These and other advantages are achieved in the present invention, which in one embodiment is an apparatus for performing an arithmetic operation on groups of pixels under program control having two memories and an arithmetic unit. One of the memories has a plurality of addressable locations N pixels in width and a read port, and N pixels from any one of the addressable locations are accessible in parallel on the read port during an address cycle. The other memory has a plurality of addressable locations greater than N pixels in width and a read port, and any N contiguous pixels from any one of the addressable locations are accessible in parallel on the read port during an address cycle. The arithmetic unit is connected to the two memory ports. In another embodiment, a memory is included that has a plurality of addressable locations greater than N pixels in width and two read ports, and any N contiguous pixels from any one of the addressable locations are accessible in parallel on each of the read ports during an address cycle. The arithmetic unit is connected to the two ports. In an embodiment of a pixel-group random access memory, a memory array

has a plurality of addressable locations N pixels in width and a read port, and N pixels from any one of the addressable locations and N pixels from an adjacent addressable location are accessible in parallel on the read port during an address cycle. A shifter has its input coupled to the read port, and provides N pixels on its output. In several method embodiments, groups of pixels are read from two memory ports, at least one of which is pixel-group random addressable, and used to determine sums of absolute differences, pixel differences, and pixel averages.").

113. Fandrianto specifically discloses methods of interpolating ½ and ¼ sub-pixels. As illustrated below, Fandrianto's Figure 5 shows unit pixels in **purple** and sub-pixels in both **blue** and **green** arranged into rows and columns:

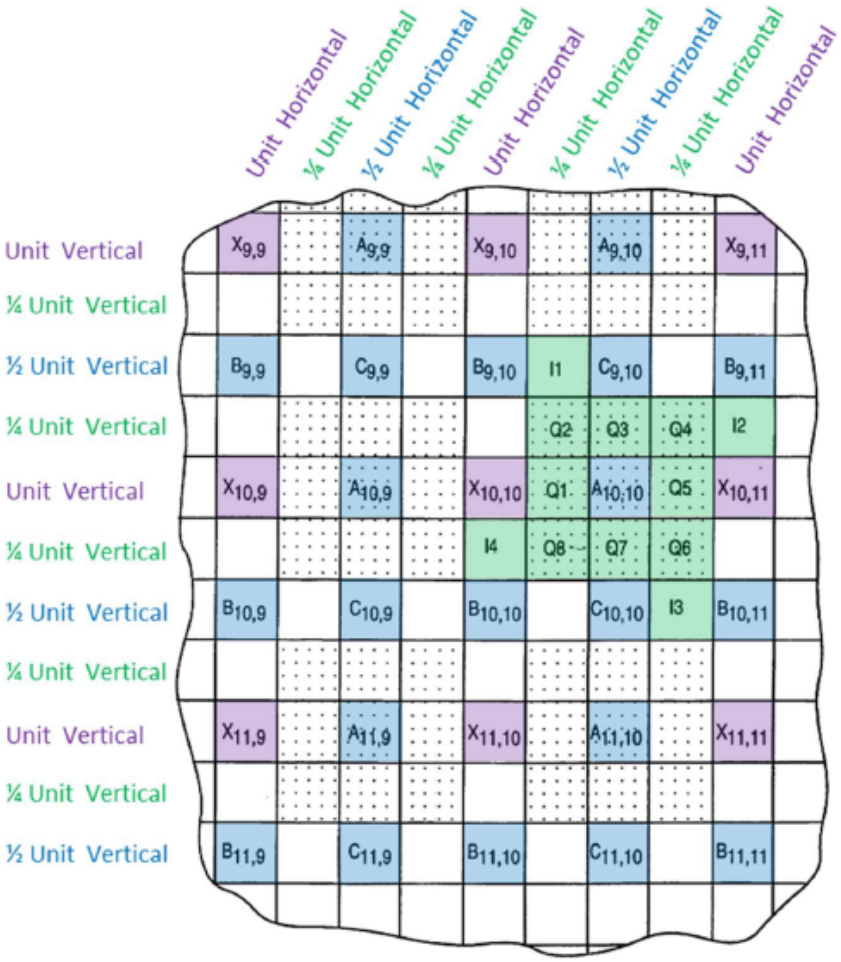


FIG. 5

Id., Figure. 5, 2:55-58 ("FIGS. 4 and 5 are pictorial representations of full pixel positions of a search matrix and their relationship to interpolated pixels at one-half and one-quarter pixel displacements."), 5:58-6:14 ("In an illustrative arrangement useful in the vision processor disclosed in the aforementioned patent document of Fandrianto et al. entitled 'Vision Processor,' the 16×16 pixels of a best match search block 71 from a motion vector search on an integral full pixel basis are shown as matrix elements $X_{1,1}$ through $X_{16,16}$, bounded within the region 402 in

FIG. 4 by a double line. Note that the full X block 70 includes $X_{r,c}$ (r=0, . . . , 18; c=0, . . . , 18) pixels. Pixels $X_{0,0}$ through $X_{0,17}$, $X_{0,0}$ through $X_{17,0}$, $X_{17,0}$ through $X_{17,17}$, and $X_{0,17}$ through $X_{17,17}$ are adjacent the best match search block 71 and are copied into X block 70 of the DPCM memory 130 to allow interpolation of fractional pixel positions about all of the pixels of the best match search block 71. Pixels $A_{r,c}$ (r=0, . . . ,17; c=0, . . . , 16) of the A block 72 are horizontally interpolated at half-pixel locations, from the X block 70. Pixels $B_{r,c}$ (r=0, . . . , 16; c=0, . . . , 17) of the B block 74 are vertically interpolated at half-pixel locations, from the X block 70. Pixels $C_{r,c}$ (r=0, . . . , 16; c=0, . . . , 16) are pixels interpolated at half-pixel locations, preferably vertically from the A matrix 72 or horizontally from the B matrix 74, but may also be interpolated diagonally from the X matrix 70. The D block 82 and the P block 80 stored in the DP memory 124 are each 16×16 pixels."), 8:20-28 ("The order shown in FIG. 5 about the pixel element $A_{10,10}$ is therefore illustrative. For purposes of illustration, assume that after half pixel estimation, the half pixel interpolated A block 72 is found to be the best match block. FIG. 5 shows in the highlighted areas about the elements $A_{r,c}$ in the interleaved search matrix 400 (see, for example, the numbered highlighted areas about element $A_{10,10}$) that eight blocks must be generated and compared with the image block 12."), 8:34-36 ("This frees up the A block 72 to hold the current quarter pixel interpolated block, the pixels of which for convenience are referred to

as $Q_{r,c}$.”). In particular, Figure 5 shows unit pixels “X” in **purple**, ½ sub-pixels “A,” “B,” and “C,” in **blue**, and ¼ sub-pixels Q in **green**. *Id.* Fandrianto refers to these pixels X in **purple** as the “X block” or “X matrix,” ½ sub-pixels “A,” “B,” and “C,” in **blue** as the “A block,” “B block,” and “C block,” respectively, or the “A matrix,” “B matrix,” and “C matrix,” respectively, and ¼ sub-pixels Q in **green** as the “quarter pixel interpolation block.” *Id.* Moreover, Fandrianto explains that ½ and ¼ sub-pixel values are interpolated as follows:

114. The values for ½ sub-pixels A in **blue** at the half unit horizontal and unit vertical locations are calculated by horizontally interpolating the two unit pixels X in **purple** horizontally adjacent to the sub-pixels A. This is illustrated below.

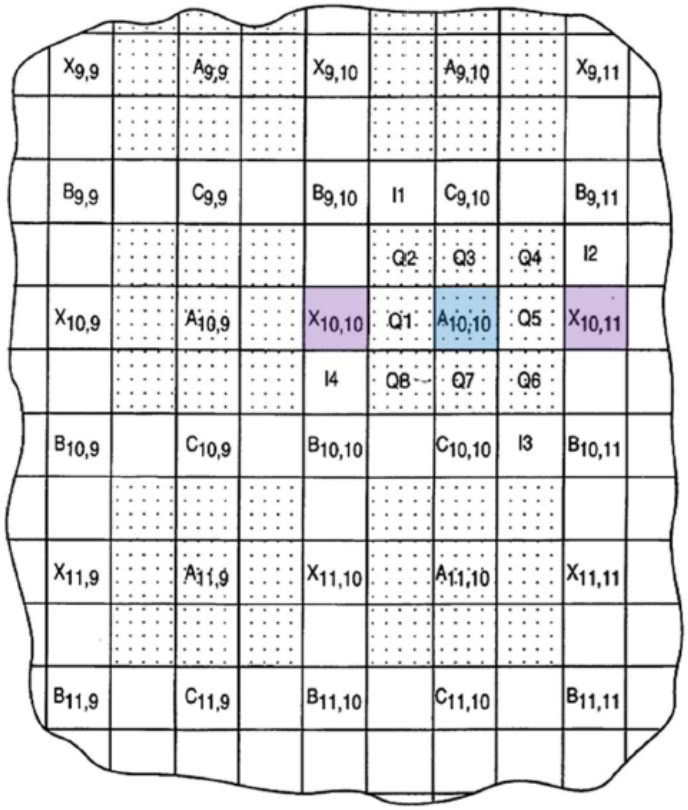


FIG. 5

Id., Figure 5, 6:18-34 ("The A block 72, the B block 74, and the C block 74 are interpolated as follows. The A block 72 is formed by progressive horizontal interpolations of the X block 70. A pixel group from a row of the X block 70 is addressed on both ports A and B of the DPCM memory 130. Accordingly, the same pixel group is loaded into both shifters 140 and 144. One of the pixel groups is shifted one pixel; for example, the pixel group in funnel shifter 144 is shifted one pixel, or eight bits, to the right. The unshifted output from funnel shifter 140 and the one-pixel right shifted output from the funnel shifter 144 are presented to respectively the A and B inputs of the ALU 154, which performs a divide by two

and a rounding off. The result is routed from the ALU 154 into appropriate address locations of the A block 74 in the DPCM memory 130. This process is continued until the entire horizontal interpolation of the X block 70 is complete and the entire A block 72 created.").

115. Similarly, as illustrated below, the values for the ½ sub-pixels B in **blue** at the unit horizontal and half unit vertical locations are calculated by vertically interpolating the two unit pixels X in **purple** vertically adjacent to the sub-pixels B.

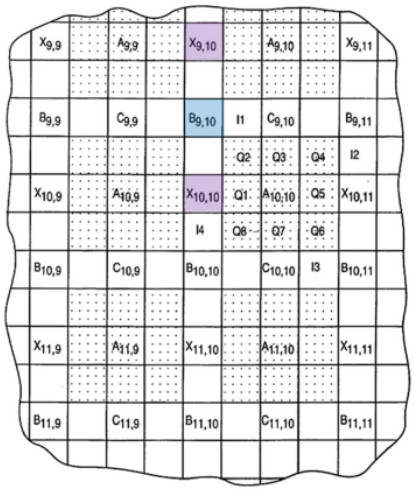


FIG. 5

Id., Figure 5, 6:35-48 ("The B block 74 is formed by progressive vertical interpolations of the X block 70. A pixel group from a row of the X block is addressed on port A of the DPCM memory 130, and a pixel group from an immediately adjacent row of the X block 70 having the same column locations is addressed on port B of the DPCM memory 130. The pixel groups on ports A and B

pass through funnel shifters 140 and 144 without being shifted, and are presented to respectively the A and B ports of the ALU 154. The ALU 154 performs a divide by two and a rounding off, and the result is routed into appropriate address locations of the B block of the DPCM memory 130. This process is continued until the entire vertical interpolation of the X block 70 is complete and the entire B block 74 created.").

116. Fandrianto also discloses, as illustrated in the figure below, that the values for the ½ sub-pixels "C" in **blue** at the half unit horizontal and half unit vertical locations are calculated in one of three ways:

(1) vertically interpolating the two ½ sub-pixels A in **blue** vertically adjacent to the sub-pixels C (illustrated as the **red** arrow),

(2) horizontally interpolating the two ½ sub-pixels B in **blue** horizontally adjacent to the sub-pixels C (illustrated as the **pink** arrow), or

(3) diagonally interpolating the two unit pixels X in **purple** diagonally adjacent to the sub-pixels C (illustrated as the **orange** arrow).

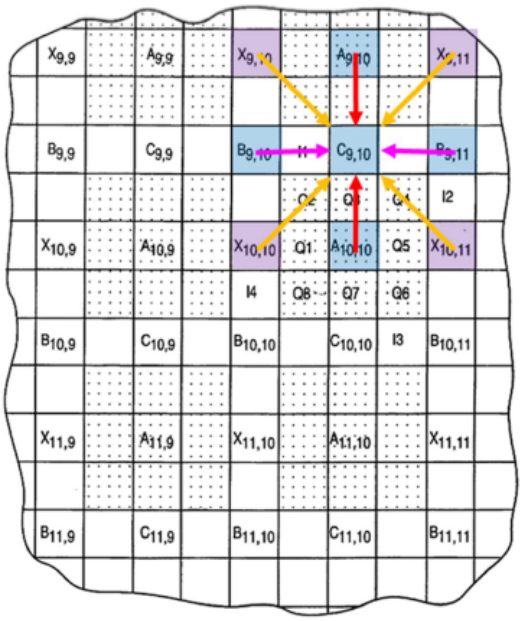


FIG. 5

Id., Figure 5, 6:49-7:2 ("The C block 76 is formed by progressive interpolation of preferably either the A block 72 vertically, or the B block 74 horizontally. Alternatively, progressive interpolation of the X block diagonally may be done. Horizontal and vertical interpolation are described above in the context of the A block 72 and the B block 74. In diagonal interpolation of the X block 70, one pixel group from the X block 70 is addressed on port A of the DPCM memory 130, and a pixel group from an immediately adjacent row of the X block 70 having the same column locations is addressed on port B of the DPCM memory 130. One of the pixel groups is shifted one pixel; for example, the pixel group in funnel shifter 144 is shifted one pixel, or eight bits, to the right. The unshifted output from funnel shifter 140 and the one-pixel right shifted output from the funnel shifter 144 are presented to respectively the A and B inputs of the ALU 154, which performs a

divide by two and a rounding off. The result is routed from the ALU 154 into appropriate address locations of the C block 76 in the DPCM memory 130. This process is continued until the entire horizontal interpolation of the X block 70 is complete and the entire C block 76 created.").

117. Finally, Fandrianto also discloses how values for the ¼ sub-pixels are interpolated. *Id.,* Figure 5, 7:27-10:58. For example, values for the ¼ sub-pixels Q2 in **green** at the quarter unit horizontal and quarter unit vertical location can be calculated by diagonally interpolating ***two*** ½ sub-pixels in **blue** diagonally adjacent to the sub-pixel Q2. This is illustrated below.

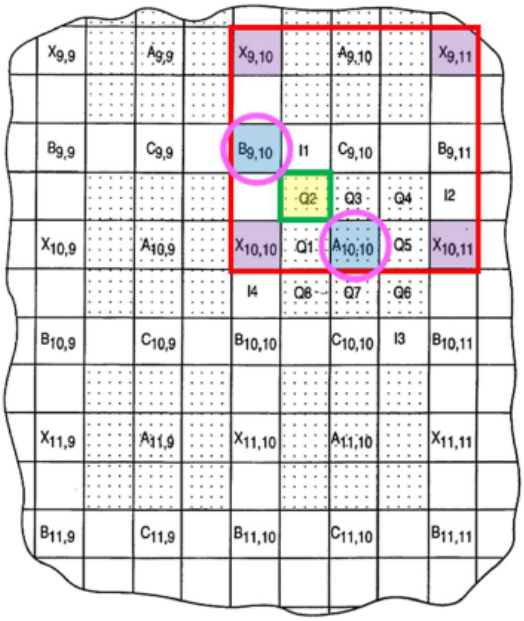


FIG. 5

Id., Figure 5, 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels Q1-Q8 may be practiced, if desired As a further

example, the Q2 pixels [**green**] may be derived from a ***diagonal interpolation*** of the data in the A block 72 [**blue**] and the B block 74 [**blue**].” (emphasis added)). Fandrianto likewise teaches interpolating the ¼ sub-pixels, including the highlighted Q2 sub-pixel (**green**), using a diagonal interpolation of the unit pixel (**purple**) and the diagonally opposed ½ sub-pixel at a half unit vertical and half unit horizontal location (**blue**).

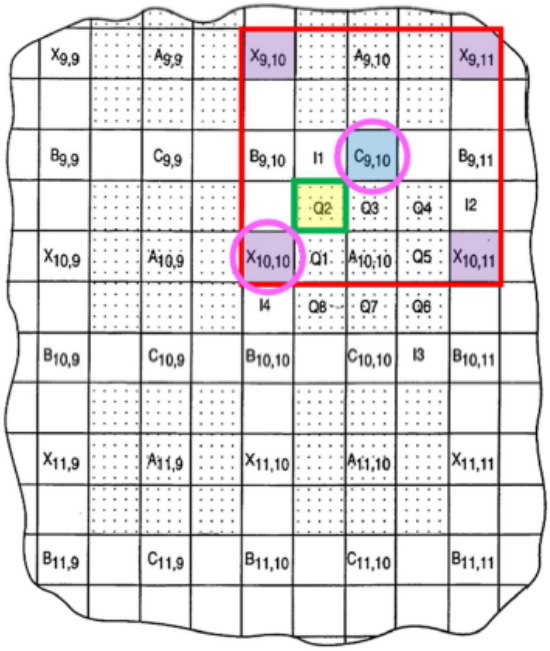


FIG. 5

Id., Figure 5, 5:26-32 (“For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less

accurate estimation operation."), 7:52-55 ("[W]hen diagonal interpolation is also used, all quarter pixel estimation search blocks are interpolated from the full [unit pixel] and half pixel estimation search blocks."), 8:58-9:32 ("The second quarter pixel estimation search block of Q2 pixels is generated from a vertical interpolation of the Q1 pixels with the results of a horizontal interpolation of the data in the B block 74 and the C block 76, with the results being stored in the A block 72 (the Q2 pixels progressively overwriting the Q1 pixels) for the absolute displaced block difference calculation, in accordance with the following expression. $\boldsymbol{Q2_{r,c}=(Q1_{r,c}+I1_{r,c})/2}$ where $\boldsymbol{I1_{r,c}=(B_{r-1,c}+C_{r-1,c})/2}$ Consider, for example, the interpolation of the Q2 pixel to the upper left of pixel $A_{10,10}$. A collection of pixels containing the pixel group $B_{9,8}$-$B_{9,15}$ in row nine of the B block 74 is addressed on, say, port A of the DPCM memory 130 and the pixel group $B_{9,8}$-$B_{9,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the A port of the ALU 154. A1 about the same time, a collection of pixels containing the pixel group $C_{9,8}$-$C_{9,15}$ in row nine of the C block 76 is addressed on, say, port B of the DPCM memory 130 and the pixel group $C_{9,8}$-$C_{9,15}$ is selected by shifter 144 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed to the DP memory 124, where it is stored as pixels $I1_{9,8}$-$I1_{9,15}$ in the ninth row of I block 84, a free 16x16 block of memory in

the DP memory 124. Next, the pixel group $I1_{9,8}$-$I1_{9,15}$ in row nine of the I block 84 is addressed on, say, port A of the DP memory 124 and presented through the multiplexer 152 to the A port of the ALU 154. At about the same time, a collection of pixels containing the pixel group $Q1_{10,8}$-$Q1_{10,15}$ in row ten of the A block 72 is addressed on, say, port A of the DPCM memory 130 and the pixel group $Q1_{10,8}$-$Q1_{10,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed back to the DPCM memory, where it is stored as pixels $Q1_{10,8}$-$Q1_{10,15}$ in the tenth row of the A block 72."), 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels Q1-Q8 may be practiced, if desired. For example, the Q2 pixels may be derived from a horizontal interpolation of the Q3 pixels with the results of a vertical interpolation of the data in the B block 74 and the X block 70. As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74.").

IX. DETAILED EXPLANATION OF GROUNDS

118. In the following sections, I demonstrate in detail how the prior art teaches each and every limitation of the claims of the '273 patent, and how those claims are rendered obvious by the prior art.

A. Ground 1: Claims 1, 4-17, 19-26, 28-29, 31, 33, 36-49, 51, 53-59, 61-62, 64-78, and 82-83 Are Obvious over TML6 in View of Fandrianto.

1. Motivation to Combine

119. A POSITA would have been motivated to combine the teachings of TML6 (disclosing an encoder and decoder with ½ and ¼ sub-pixel interpolation for motion prediction) and Fandrianto (disclosing a method of interpolating the ¼ sub-pixels residing at quarter unit horizontal and quarter unit vertical locations) by implementing Fandrianto's method for interpolating the ¼ sub-pixels residing at quarter unit horizontal and quarter unit vertical locations in the TML6 code. As I described in detail in the preceding sections, the two references are in the same field (*i.e.,* motion prediction video coding), address the same problem (*i.e.,* objects in video data moving from frame to frame by distances that do not match to the pixel locations precisely), and purport to solve this problem by interpolating sub-pixels at fractional locations (*i.e.,* ½ and ¼ sub-pixels) to improve motion prediction accuracy. *See* Fandrianto, 5:26-28 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation."); TML6, 16-24 (/tml6pre.zip/lencod/src/block.c, ll. 734-1180 ("motion_search" function)), 115-116 (/tml6pre.zip/lencod/src/macroblock.c, ll. 769-842 ("Motion vector search")), 57-59 (/tml6pre.zip/lencod/src/image.c ll. 538-632 (sub-pixel interpolation)); EX-1001, 8:38-9:3, 11:25-13:18 ("The sub-pixel

value interpolation scheme according to TML6, referred to herein as direct interpolation, will now be described. In the encoder the interpolation method according to TML6 works like the previously described TML5 interpolation method, except that maximum precision is retained throughout. This is achieved by using intermediate values which are neither rounded nor clipped. A step-by-step description of [the] interpolation method according to TML6 as applied in the encoder is given below with reference to FIGS. 13 *a*, 13 *b* and 13 *c*.").

120. First, Fandrianto teaches a variety of techniques that could have been used to interpolate ¼ sub-pixels, and that one could select different techniques depending on the particular features of the system implementing the interpolation. Fandrianto, 7:32-37 ("Various interpolation techniques may be used in the derivation of a given quarter pixel estimation block, depending on various factors such as the amount of memory available and the size of the memory ports and data buses."). Specifically, Fandrianto teaches, as an alternative to the vertical interpolation embodiment discussed at 8:58-9:32, that a diagonal interpolation averaging two ½ sub-pixels can be used for the ¼ sub-pixels located at ¼ unit horizontal and ¼ unit vertical locations ("Q2"). *Id.,* 10:51-58 ("permutations of the order of derivation of pixels Q1-Q8 may be practiced…. As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74"). Not only does Fandrianto explicitly suggest that this

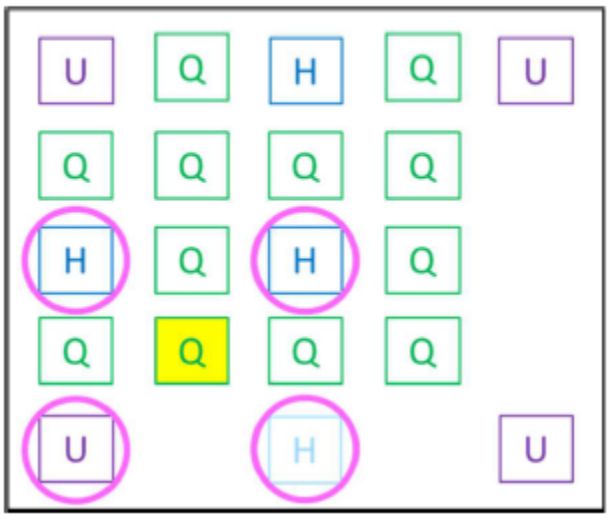
alternative can be practiced as an alternative to other interpolation methods for ¼ sub-pixels located at ¼ unit horizontal and ¼ unit vertical locations, but also, a POSITA would have been motivated to use Fandrianto's ¼ sub-pixel interpolation technique in TML6 to take advantage of the optimized and versatile software and hardware architecture for such an interpolation that Fandrianto discloses. *Id.,* 2:4-7 ("For example, the programmability aspect of the present invention enables support of future algorithms, and allows the addition of customer-proprietary optimizations and algorithms."), 3:55-4:6 ("An illustrative programmable architecture 100 for implementing motion vector searching is illustrated in FIG. 2. Rapid and efficient motion vector searching is accommodated by two high-speed, multi-ported register files in the datapath of the architecture 100: an image block, best match block memory conveniently referred to as DP memory 124, and a search memory conveniently referred to as DPCM memory 130. The memories 124 and 130 are configured in an advantageous manner based on the desired video information block size and on the critical operations required of the architecture 100 in executing certain widely accepted current standards and possible future standards. Other important data path elements for motion vector estimation include two funnel shifters 140 and 144, an arithmetic logic unit ("ALU") 154, and a tree adder 156. Shifter 140 is connected to port A of the DPCM memory 130, and shifter 144 is connected to port B of the DPCM memory 130. ALU 154 receives

pixel data items from shifters 140 and 144 and from the DP memory 124."), 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 7:26-37 ("Motion vector searching on a quarter-pixel basis is similar to motion vector searching on a full-pixel basis, except that an absolute block difference is determined from a comparison of the P block 80 with a memory block that contains pixels displaced one-quarter pixel from the best match block. Various interpolation techniques may be used in the derivation of a given quarter pixel estimation block, depending on various factors such as the amount of memory available and the size of the memory ports and data buses. The following technique is suitable for the architecture of FIG. 2, although other techniques may be used if desired."), Figure 2. *See also id.,* 7:52-7:55 ("[W]hen diagonal interpolation is also used, all quarter pixel estimation search blocks are interpolated from the full [unit pixel] and half pixel estimation search blocks.").

121. Second, using Fandrianto's method of interpolating the sub-pixels located at ¼ unit horizontal and ¼ unit vertical locations with the encoder and decoder of TML6 is a simple substitution of one known element (*i.e.,* Fandrianto's

¼ sub-pixel interpolation scheme) for another known element (*i.e.,* TML6's ¼ sub-pixel interpolation scheme) to obtain a predictable result (*i.e.,* interpolation ¼ sub-pixels). Both TML6 and Fandrianto disclose known methods of interpolating the sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations for use in motion prediction, using values of other sub-pixels and/or pixels. This is demonstrated in the figure below comparing the methods of TML6 and Fandrianto.

TML6



Fandrianto

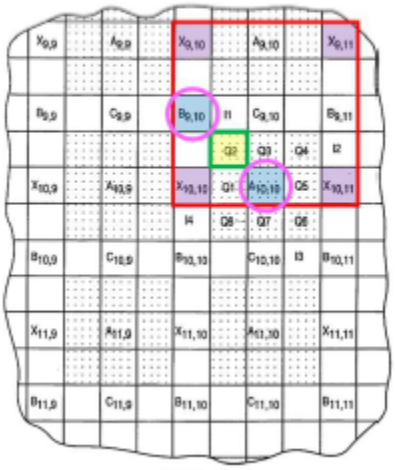


FIG. 5

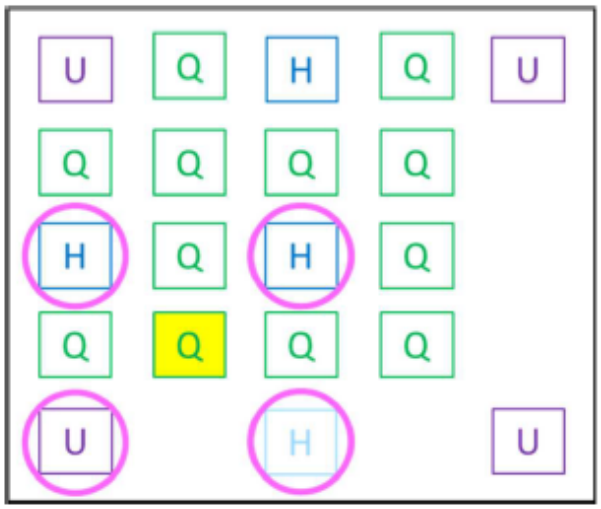
As illustrated on the left, TML6 discloses interpolating the highlighted sub-pixel located at ¼ unit horizontal and ¼ unit vertical location in **green**, along with other ¼ sub-pixels at other ¼ unit horizontal and ¼ unit vertical locations, based on the closest unit pixel in **purple** and the three closest ½ sub-pixels in **blue**. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-625). And as illustrated on the right, Fandrianto discloses interpolating the highlighted sub-pixels Q2 in **green** located

at ¼ unit horizontal and ¼ unit vertical location based on the two closest diagonally opposed ½ sub-pixels A and B in **blue**.[12] Fandrianto, Figure 5, 10:51-58 ("[T]he Q2 pixels may be derived from a ***diagonal interpolation*** of the data in the A block 72 [*i.e.,* a sub-pixel located at a ½ unit horizontal and unit vertical location] and the B block 74 [*i.e.,* a sub-pixel located at unit horizontal and ½ unit vertical location]." (emphasis added)).

122. Fandrianto also discloses interpolating ¼ sub-pixels using a weighted average of the value of a nearest-neighbouring pixel (**purple** and circled) and the value of the sub-pixel at the ½ unit vertical and ½ unit horizontal location (**blue** and circled), with weights equal to "1."

[12] While Figure 5 depicts one Q2 sub-pixel, Fandrianto clearly discloses that a "block of Q2 [sub-]pixels" are interpolated. Fandrianto, Figure 5, 8:58-9:32, 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels [¼ sub-pixels] Q1-Q8 may be practiced, if desired. For example, the Q2 pixels may be derived from a horizontal interpolation of the Q3 pixels with the results of a vertical interpolation of the data in the B block 74 and the X block 70. As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74.").

TML6



Fandrianto

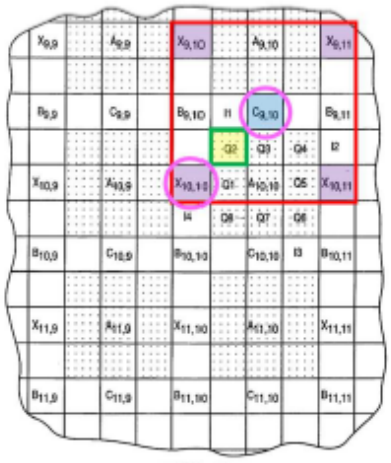


FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 7:52-55 ("[W]hen diagonal interpolation is also used, all quarter pixel estimation search blocks are interpolated from the full [unit pixel] and half pixel estimation search blocks."), 8:58-9:32 ("The second quarter pixel estimation search block of Q2 pixels is generated from a vertical interpolation of the Q1 pixels with the results of a horizontal interpolation of the data in the B block 74 and the C block 76, with the results being stored in the A block 72 (the Q2 pixels progressively overwriting the Q1 pixels) for the absolute

displaced block difference calculation, in accordance with the following expression. $\boldsymbol{Q2_{r,c}=(Q1_{r,c}+I1_{r,c})/2}$ where $\boldsymbol{I1_{r,c}=(B_{r-1,c}+C_{r-1,c})/2}$. Consider, for example, the interpolation of the Q2 pixel to the upper left of pixel $A_{10,10}$. A collection of pixels containing the pixel group $B_{9,8}$-$B_{9,15}$ in row nine of the B block 74 is addressed on, say, port A of the DPCM memory 130 and the pixel group $B_{9,8}$-$B_{9,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the A port of the ALU 154. A1 about the same time, a collection of pixels containing the pixel group $C_{9,8}$-$C_{9,15}$ in row nine of the C block 76 is addressed on, say, port B of the DPCM memory 130 and the pixel group $C_{9,8}$-$C_{9,15}$ is selected by shifter 144 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed to the DP memory 124, where it is stored as pixels $I1_{9,8}$-$I1_{9,15}$ in the ninth row of I block 84, a free 16x16 block of memory in the DP memory 124. Next, the pixel group $I1_{9,8}$-$I1_{9,15}$ in row nine of the I block 84 is addressed on, say, port A of the DP memory 124 and presented through the multiplexer 152 to the A port of the ALU 154. At about the same time, a collection of pixels containing the pixel group $Q1_{10,8}$-$Q1_{10,15}$ in row ten of the A block 72 is addressed on, say, port A of the DPCM memory 130 and the pixel group $Q1_{10,8}$-$Q1_{10,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and

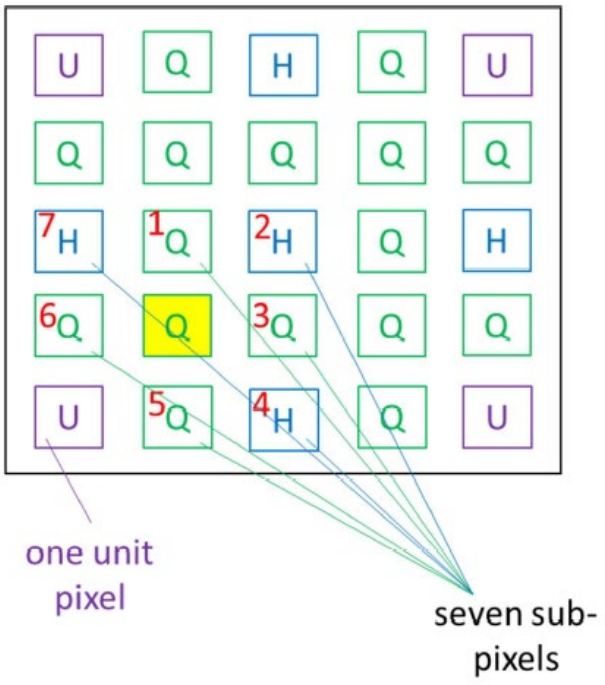
rounds the result. An eight pixel result at the output of the ALU 154 is routed back to the DPCM memory, where it is stored as pixels $Q1_{10,8}$-$Q1_{10,15}$ in the tenth row of the A block 72."), 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels Q1-Q8 may be practiced, if desired. For example, the Q2 pixels may be derived from a horizontal interpolation of the Q3 pixels with the results of a vertical interpolation of the data in the B block 74 and the X block 70. As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74.").

123. It therefore would have been a simple substitution to use the known interpolation method of Fandrianto instead of the interpolation method of TML6. Doing so would produce the predictable results of interpolating the value of sub-pixels residing at ¼ unit horizontal and ¼ unit vertical locations.

124. Third, for a POSITA, it would have been obvious to try Fandrianto's method to interpolate the sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations in TML6. As illustrated below, a POSITA would have recognized that there are a finite number of identifiable and predictable solutions to interpolate such sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations. In particular, both TML6 and Fandrianto perform interpolation for sub-pixels (within a block of pixels and sub-pixels) based on a finite number of pixels and sub-pixels available for the interpolation. *See, e.g.,* TML6, 57-59

(/tml6pre.zip/lencod/src/image.c, ll. 538-632), 49-50 (/tml6pre.zip/lencod/src/image.c, ll. 97-109 ("I- and P-frames:loop-filtered ***imgY***, imgUV -> ***mref[][][]***, mcef[][][][]" (emphasis added))), 110-112 (/tml6pre.zip/lencod/src/macroblock.c, ll. 480-600 (the *LumaResidualCoding_P()* function populates the ***imgY*** data structure with luminance data from the ***img*** data structure); Fandrianto, 3:16-37 ("The image block 12 may be any convenient size; 16×16 pixels "). A POSITA would not, however, look to the entire (yet still finite) block of pixels and sub-pixels to perform sub-pixel interpolation, as interpolation is based on the well-known concept that values for sub-pixels at fractional locations are typically similar to values of the nearby pixels and sub-pixels (rather than pixels and sub-pixels located far away from the sub-pixel that is being interpolated). *See, e.g.,* VCEG-M28 (describing 2-tap, 4-tap, and 6-tap filters). Indeed, this concept is at the core of the mathematical theory of interpolation, generally, which relates to estimating new data points based on surrounding known data points. *See, e.g.,* Britannica, 356 ("interpolation"). For example, commonly known filters for sub-pixel interpolation include 2-tap, 4-tap, and 6-tap filters, taking into account 2, 4, or 6 nearby pixels or sub-pixels ***in a row***, respectively. *See, e.g.,* VCEG-M28. But longer filters using values of pixels or sub-pixels that are further away (such as using 8 pixels or sub-pixels) were rare, if used at all.

125. As a result, a POSITA would have recognized that there are a finite number of pixels and sub-pixels near these sub-pixels located at a ¼ unit horizontal and ¼ unit vertical location available to reasonably use in the interpolation to produce accurate interpolation results. For example, the illustration below shows that there are only seven sub-pixels and one unit pixel immediately surrounding the highlighted ¼ sub-pixels located at quarter unit horizontal and quarter unit vertical locations. A POSITA would first look to these ***closest*** sub-pixels, as in TML6 and Fandrianto, to develop an interpolation scheme for the sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations.



126. Accordingly, a POSITA would have recognized that there are only a finite number of permutations of pixels and sub-pixels located near the highlighted

sub-pixel located at a ¼ unit horizontal and ¼ unit vertical location that could reasonably be used to interpolate the sub-pixel accurately. It therefore would have been obvious to try a well-known one of these finite options—*i.e.,* Fandrianto's interpolation method for such sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations—for use in the encoder and/or decoder of TML6 as an alternative to the method used by TML6 to interpolate this sub-pixel.

127. Finally, A POSITA would have understood that the alternative methods of interpolating the sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations is a design choice between finite options with well-understood tradeoffs—*i.e.,* improved accuracy and computational efficiency. Improving accuracy of the compression often increases the amount of computations that must be performed. Conversely, reducing the number of computations needed to compress videos often results in a less accurate compression. In fact, this tradeoff is well known and central to both interpolation and video compression, generally.

128. For instance, TML6's interpolation method is arguably more accurate because it takes into account more of the pixels or sub-pixels immediately surrounding the sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations being interpolated, but it is also more computationally complex because it relies on four values for the average rather than two. TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-625). By contrast, Fandrianto's

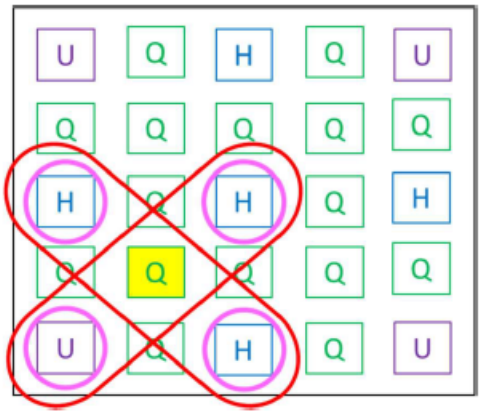
interpolation method may, in some instances, be less accurate because it does not use as many pixels or sub-pixels to interpolate the ¼ sub-pixel, but it also has the advantage of being less computationally intensive because it only relies on two values for the average rather than four. Fandrianto, Figure 5, 7:52-55 ("[W]hen diagonal interpolation is also used, all quarter pixel estimation search blocks are interpolated from the full [unit pixel] and half pixel estimation search blocks."), 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels [¼ sub-pixels] Q1-Q8 may be practiced, if desired. For example, the Q2 pixels may be derived from a horizontal interpolation of the Q3 pixels with the results of a vertical interpolation of the data in the B block 74 and the X block 70. As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74."). And as I detailed in the forgoing paragraphs, Fandrianto explicitly discloses optimizing interpolation methods based on available memory (*i.e.,* accuracy) and size of ports and data buses (*i.e.,* computational efficiency). Fandrianto, 7:32-37 ("Various interpolation techniques may be used in the derivation of a given quarter pixel estimation block, depending on various factors such as the amount of memory available and the size of the memory ports and data buses."). A POSITA thus would have been motivated to use the interpolation method of Fandrianto for the

sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations as a design choice in balancing considerations of accuracy and computational efficiency.

129. The combination of TML6 and Fandrianto would predictably result in the interpolation of ¼ sub-pixels located at ¼ unit horizontal and ¼ unit vertical locations in TML6 using Fandrianto's interpolation method, with a reasonable expectation of success. Both TML6 and Fandrianto disclose methods for interpolating these same sub-pixels located at a ¼ unit horizontal and ¼ unit vertical locations by using the same inputs (*i.e.,* neighbouring sub-pixels and/or pixels) and the same basic arithmetic. For any given sub-pixel located at a ¼ unit horizontal and ¼ unit vertical location, the interpolation method disclosed in Fandrianto would achieve the same result—whether implemented in hardware (as described in Fandrianto) or implemented in the TML6 code.

130. Also, making this combination is trivial and thus predictable. As I discuss above, both methods take an average of neighbouring diagonal sub-pixels. As illustrated in the figure below, the only difference is that TML6 discloses taking an average of both sets of two diagonally opposed sub-pixels (circled in **red**, on the left), but Fandrianto discloses taking an average of only one set of diagonally opposed sub-pixels (circled in **red**, on the right).

TML6



Fandrianto

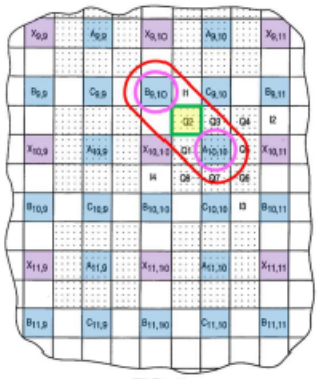


FIG. 5

Put another way, the average used in TML6 mathematically includes the average taught by Fandrianto. In mathematical terms, the single diagonal interpolation of Fandrianto can be written as:

$$Q = \frac{H+H}{2}.$$

With the values **H** being stored values for the sub-pixel in the A block and the sub-pixel in the B block. Furthermore, the average of TML6 can be represented as:

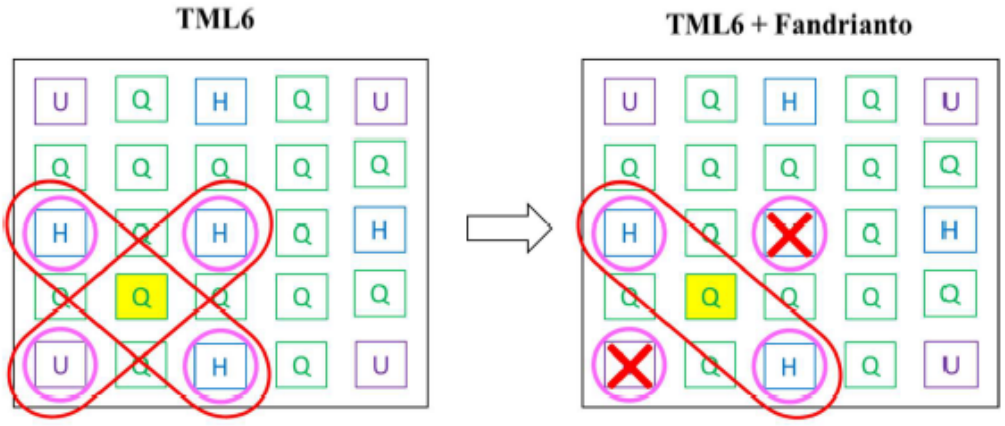
$$Q = \frac{H+H+H+U}{4},$$

with the values **H** and **U** being values stored for sub-pixels and the pixel in the ***imgY*** data structure. It is readily apparent that the TML6 average is mathematically equivalent to averaging two diagonal interpolations:

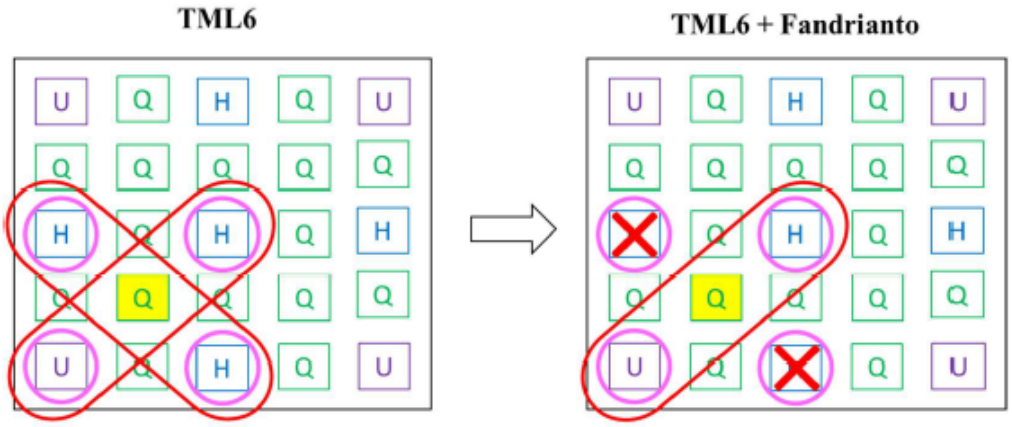
$$Q = \frac{\left(\frac{H+H}{2}\right)+\left(\frac{H+U}{2}\right)}{2}.$$

TML6 therefore mathematically includes the average taught by Fandrianto, as highlighted above, showing that TML6 teaches all of the values that are required as inputs for such a calculation (*i.e.,* the values for the two ½ sub-pixels used in the interpolation set forth in Fandrianto). And this is also true for the average with the unit pixel and sub-pixel located at a ½ unit horizontal and ½ unit vertical location.

131. Thus, implementing the method of interpolation disclosed in Fandrianto in the interpolation of TML6 would have been trivial for a POSITA, either by (1) using the interpolation hardware disclosed in, for example, Figure 2 of Fandrianto, for which TML6 already calculates all required inputs (which would achieve the added benefit of freeing up the processor running the TML6 code for other functions), or (2) by removing the unit pixel in **purple** and the ½ sub-pixel in **blue** located at half unit horizontal and half unit vertical locations from the average in the TML6 code, shown below for the highlighted sub-pixel located at a ¼ unit horizontal and ¼ unit vertical location. See TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-625).




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                ][i/2                ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                ] +
612                       img4Y_tmp[j/2                ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
```




```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img1Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                    ][i/2                    ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                    ] +
612                       img4Y_tmp[j/2                    ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 602-613). Accordingly, using Fandrianto's interpolation in TML6 would also predictably produce an interpolation result for the same sub-pixel, and a POSITA would have reasonably expected it to do so.

132. Likewise, a POSITA would have been motivated to replace TML6's method of interpolating all ¼ sub-pixels with the Fandrianto's method of doing so. Fandrianto discloses interpolating all ¼ sub-pixels using averages of values for pixels in the X-block and/or already interpolated ½ sub-pixels in the A-block, B-block, and/or C-block. Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 7:27-10:58 (detailing interpolation of ¼ sub-pixels Q1-Q8 and I1-I4). As I discussed above with respect to the sub-pixels located at a ¼ unit horizontal and ¼ unit vertical location, a POSITA would have been motivated to modify the TML6 code to implement the ¼ sub-pixel interpolation scheme of Fandrianto. *Id.* Alternatively, a POSITA would have been motivated to replace the ¼ sub-pixel interpolation in the TML6 ***oneforthpix()*** function shown below (TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-625)) with the technique implemented in Fandrianto's software and hardware. Fandrianto, Figures 2, 5, 5:26-32, 7:27-10:58.

TML-6 *oneforthpix()* ¼ Sub-Pixel Interpolation	Fandrianto ¼ Sub-Pixel Interpolation

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
            ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602         for (i=0;i<ie2+4;i++)
603         {
604             for (j=0;j<je2+3;j+=2)
605             {
606                 if( i%2 == 0 )
607                     mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                 else
609                     mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                     img4Y_tmp[j/2                ][i/2                ] +
611                     img4Y_tmp[min(je2/2+1,j/2+1)][i/2                ] +
612                     img4Y_tmp[j/2                ][min(ie2/2+1,i/2+1)] +
613                     img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
614
615                     /* "funny posision" */
616                 if( ((i&3) == 3)&&(((j+1)&3) == 3))
617                 {
618                     mref[img->frame_cycle][j+1][i] = (mref[img->
                        frame_cycle][j-2          ][i-3          ] +
619                         mref[img->frame_cycle][min(je2,j+2)][i-3
                                ] +
620                         mref[img->frame_cycle][j-2          ][min(ie2,i
                            +1)] +
621                         mref[img->frame_cycle][min(je2,j+2)][min(ie2,i
                            +1)] + 2 )/4;
622                 }
623             }
624         }
625
626         /*  Chroma: */
627         for (uv=0; uv < 2; uv++)
628             for (j=0; j < img->height_cr; j++)
629                 memcpy(mcef[img->frame_cycle][uv][j],imgUV[uv][j],img
                    ->width_cr); /* just copy 1/1 pix, interpolate
                    "online" */
630
631 }
```

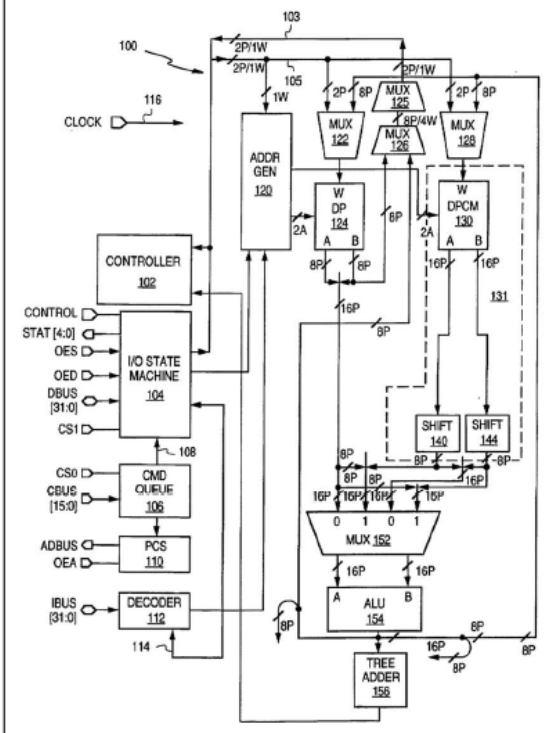


FIG. 2

A POSITA would have recognized that doing so would realize the benefits of the dedicated and optimized hardware for Fandrianto's ¼ sub-pixel interpolation technique rather than relying on a CPU to execute the ¼ sub-pixel interpolation scheme in the TML6 code. *Id.,* Figure 2, 2:4-7 ("[T]he programmability aspect of the present invention enables support of future algorithms, and allows the addition of customer-proprietary optimizations and algorithms."), 3:55-4:6 ("An illustrative programmable architecture 100 for implementing motion vector searching is illustrated in FIG. 2. Rapid and efficient motion vector searching is accommodated

by two high-speed, multi-ported register files in the datapath of the architecture 100: an image block, best match block memory conveniently referred to as DP memory 124, and a search memory conveniently referred to as DPCM memory 130. The memories 124 and 130 are configured in an advantageous manner based on the desired video information block size and on the critical operations required of the architecture 100 in executing certain widely accepted current standards and possible future standards. Other important data path elements for motion vector estimation include two funnel shifters 140 and 144, an arithmetic logic unit ("ALU") 154, and a tree adder 156. Shifter 140 is connected to port A of the DPCM memory 130, and shifter 144 is connected to port B of the DPCM memory 130. ALU 154 receives pixel data items from shifters 140 and 144 and from the DP memory 124."), 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 7:32-37 ("Various interpolation techniques may be used in the derivation of a given quarter pixel estimation block, depending on various factors such as the amount of memory available and the size of the memory ports and data buses."). Making this simple substitution of the known

method of Fandrianto for the known method of TLM6 would have the added benefit of freeing up memory in the CPU for other video compression functionality thereby speeding the compression. Furthermore, a POSITA would have expected that the combination would produce predictable results and would have had a reasonable likelihood of success because both TML6 and Fandrianto produce the end result of interpolated ¼ sub-pixels. And, as I discuss in further detail below, TML6 discloses that all of the values for unit pixels and ½ sub-pixels that are required to interpolate the ¼ sub-pixels addressed in Fandrianto are already stored in the ***imgY*** and ***mref*** data structures of TML6. That is, TML6 already has all of the inputs required for Fandrianto's ¼ sub-pixel interpolation scheme. Finally, using Fandrianto's ¼ sub-pixel interpolation scheme, rather than TML6's software, would increase the speed of interpolating the ¼ sub-pixels.

2. Claims 1, 4-17, 19-26, and 28-29

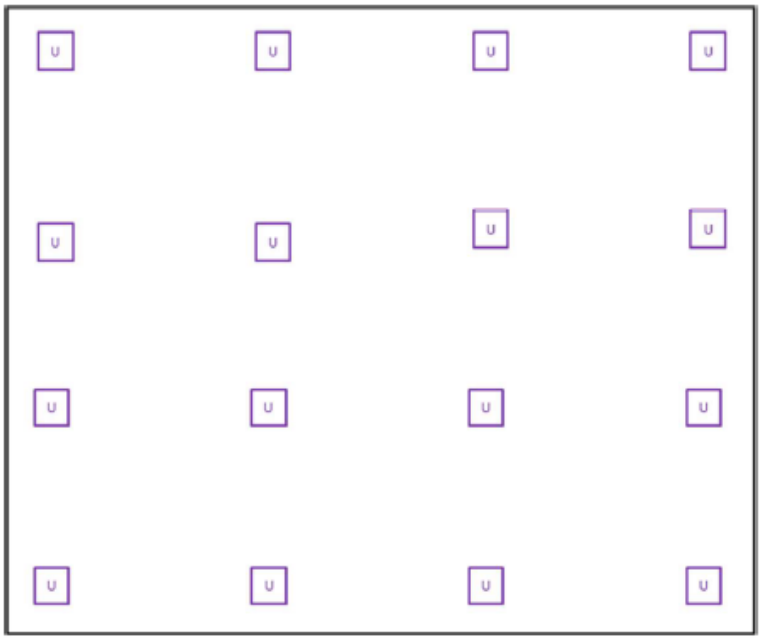
a) Independent Claim 1

(i) Element 1[pre][a]: "A method for sub-pixel value interpolation to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

133. I understand that this limitation forms part of the preamble of this claim and therefore may not be limiting. To the extent that the Board treats Element 1[pre][a] as limiting, the prior art teaches it.

134. For example, TML6 describes a method for sub-pixel value interpolation. TML6, 48 (/tml6pre.zip/lencod/src/image.c, l. 3) ("Code one image/slice"). TML6 includes a file ***image.c*** (in the src folder of the ***encoder*** folder) containing a function ***encode_one_frame()*** that "[e]ncodes one frame" of video data. *Id.,* 48-50 (/tml6pre.zip/lencod/src/image.c, ll. 33-162). This function ***encode_one_frame()*** calls the function ***oneforthpix()*** to interpolate sub-pixels for the frame of video data. *Id.,* 49 (/tml6pre.zip/lencod/src/image.c, ll. 102-109), 57-60 (/tml6pre.zip/lencod/src/image.c, ll. 538-719). Furthermore, explanation of how the ***oneforthpix()*** function works applies equally to the ***oneforthpix_2()*** function, because these functions interpolate sub-pixels in the same way.

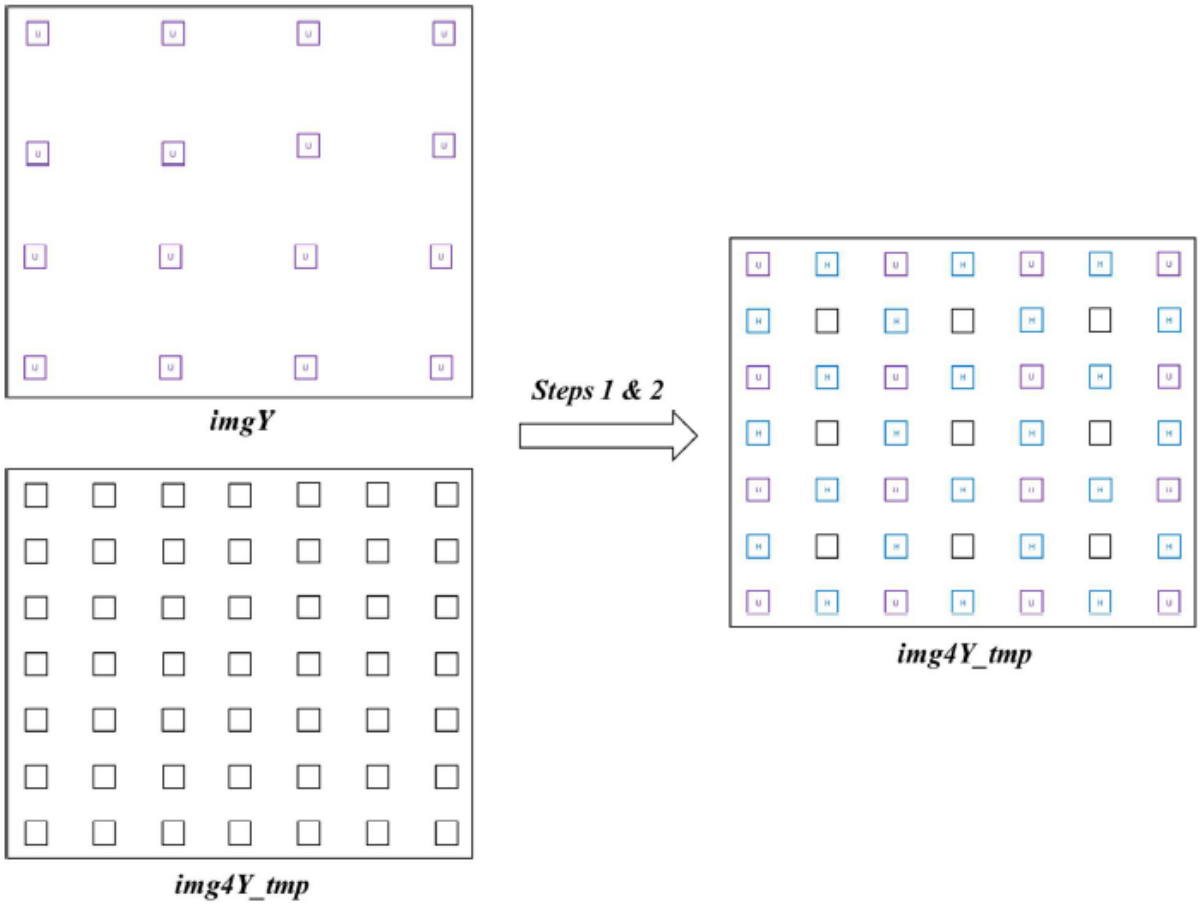
135. The ***imgY*** data structure stores data values representing the brightness—or "luminance"—of unit pixels "U" (**purple**) for a frame of video data as illustrated below.



imgY

TML6, 93 (/tml6pre.zip/lencod/src/lencod.c, ll. 1338-1344), 49-50 (/tml6pre.zip/lencod/src/image.c, ll. 97-109), 110-112 (/tml6pre.zip/lencod/src/macroblock.c, ll. 480-600), 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 563-632), 131 (/tml6pre.zip/lencod/src/global.h, l. 149). Then, as I explain in further detail below for Elements 1[c]-1[e], the ***oneforthpix()*** function performs three primary steps to interpolate sub-pixel values based on the stored unit pixel values in the ***imgY*** data structure. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 553-625).

136. As I note below, in the initial two steps, the ***oneforthpix()*** function generates a temporary data structure—***img4Y_tmp***—interpolates values for some of the ½ sub-pixels (**blue**) based on the values of the unit pixels (**purple**), and stores these values in the ***img4Y_tmp*** data structure.

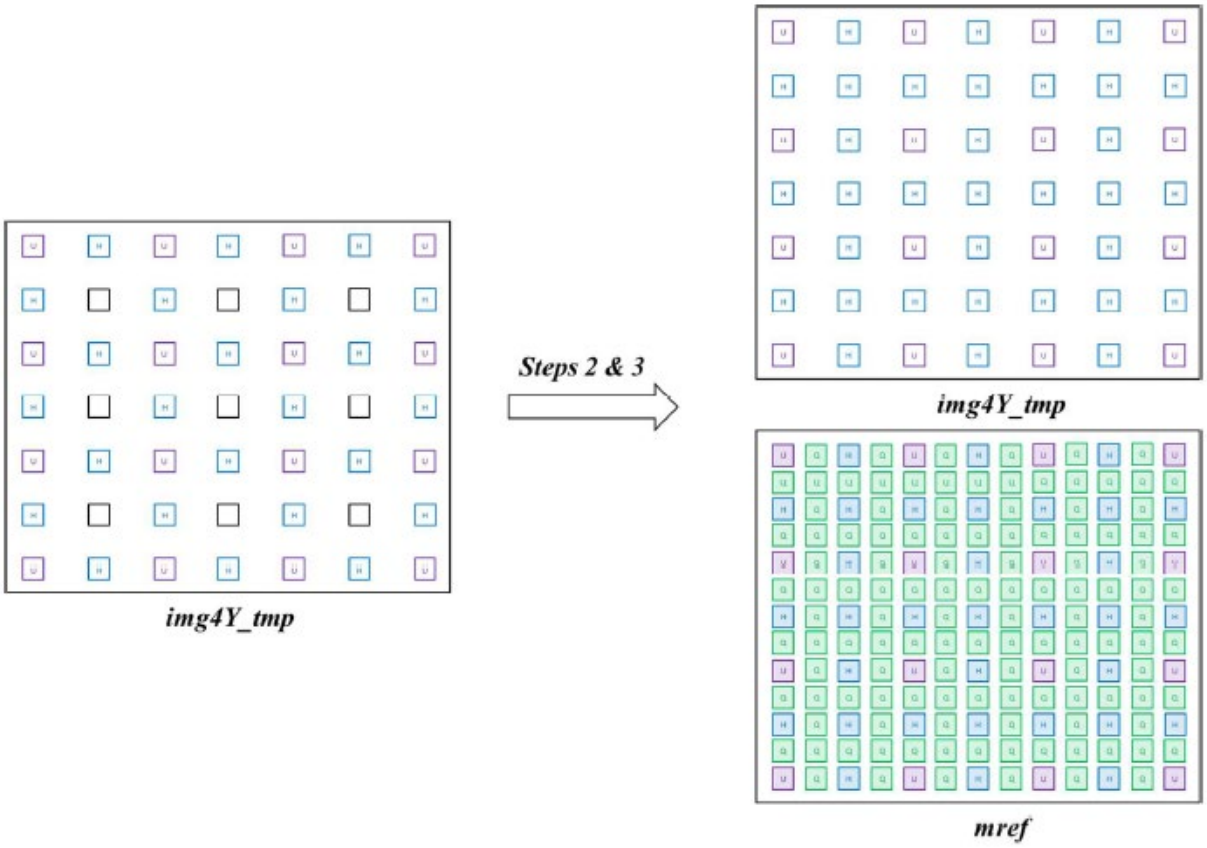


```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 563-592).

137. As I further describe below, in steps two and three, the ***oneforthpix()*** function interpolates values for additional ½ sub-pixels (**blue**) (some based on unit pixel values and others based on previously-interpolated ½ sub-pixels values) and stores these values in the data structure ***img4Y_tmp*** (below upper right). The ***oneforthpix()*** function uses the values for the ½ sub-pixels (**blue**) and unit pixels

(**purple**) in the data structure ***img4Y_tmp*** (below upper right) to interpolate values for ¼ sub-pixels (**green**) and stores the ¼ and ½ sub-pixel values (**green**) and (**blue**), in the data structure ***mref*** (below lower right).



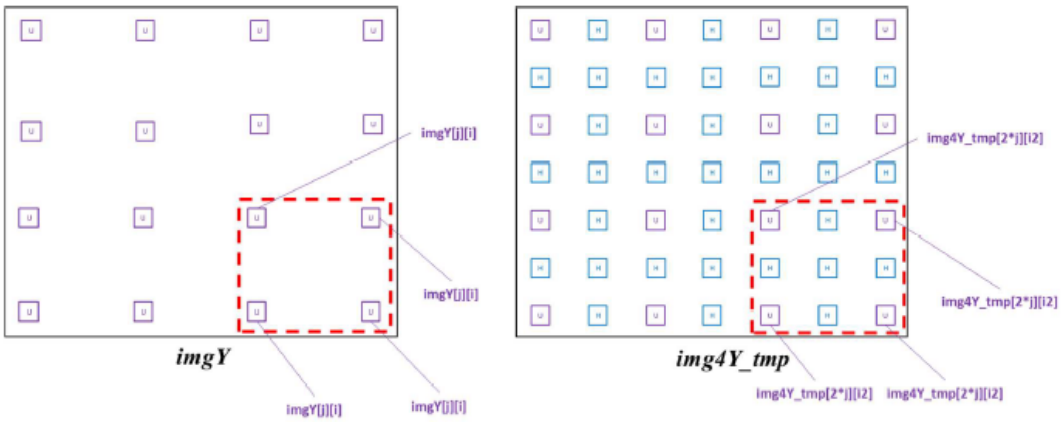
```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
            ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                          )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                          )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                  ][i/2                  ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                  ] +
612                       img4Y_tmp[j/2                  ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                          + 2048)/4096));
```

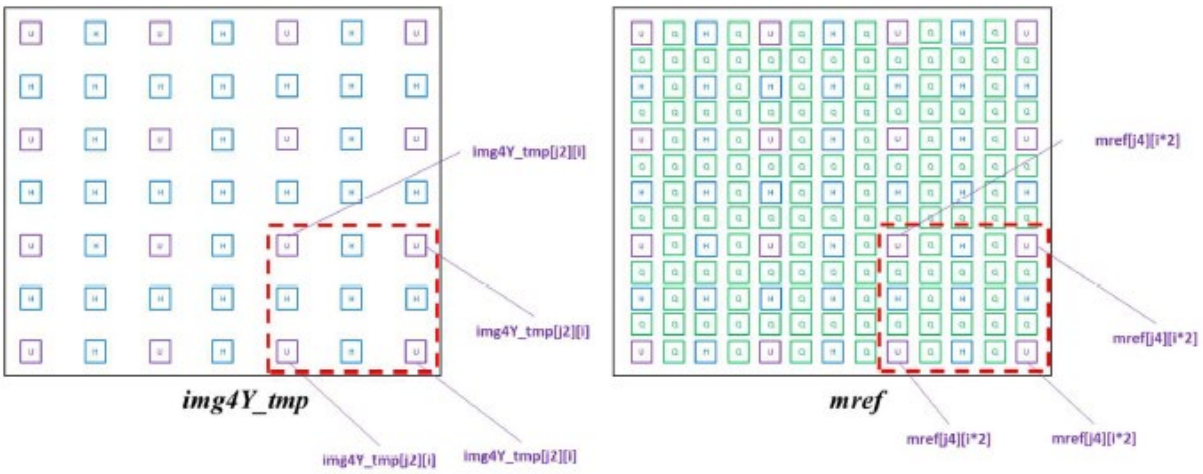
TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625).

138. As I have explained above, and shown below in more detail, the data structures ***img4Y_tmp*** and ***mref*** include ½ sub-pixels (**blue**) and/or ¼ sub-pixels (**green**) that are located within a rectangular bounded region (**red**) that is defined by four corner unit pixels (**purple**) and that has no intermediate pixels between the corners. The unit pixels (**purple**) and sub-pixels (**blue** and **green**) are, additionally, arranged in rows and columns.

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3))));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```



```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```



TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 554-592). The pixels are configured for display. TML6, 1-2 (/tml6pre.zip/lencod/readme.txt ("picture formats")). Similarly, if a motion vector points to a sub-pixel, the sub-pixels are also configured for display. TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632), 16-24 (/tml6pre.zip/lencod/src/block.c, ll. 734-1180 ("motion_search"

function)), 115-116 (/tml6pre.zip/lencod/src/macroblock.c, ll. 769-842 ("Motion vector search")).

139. The '273 patent explains, and thus confirms my assessment, that the TML6 source code performs a method for interpolating pixels and sub-pixels to determine values for the sub-pixels. EX-1001, Figures 13(a)-13(c), 8:38-9:36 ("[I]nterpolation schemes have been developed as part of the work ongoing in the ITU-Telecommunications Standardization Sector, Study Group 16, Video Coding Experts Group (VCEG), Question[] 6 … These approaches were proposed for incorporation into ITU-T recommendation H.26L and have been implemented in test models (TML) for the purposes of evaluation and further development…. The test model corresponding to …. Question 6 is known as Test Model 6 (TML6)…."), 11:25-13:18 (describing TML6's interpolation of ½ and ¼ sub-pixels).

140. In its description of TML6 with reference to Figure 13(a) (reproduced below), the '273 patent further notes that TML6 interpolates ½ sub-pixels "b" and "c" (**blue**) and ¼ sub-pixels "d" to "g" (**green**) and that these sub-pixels are located within a rectangular bounded region (**red**) defined by four corner pixels "A" (**purple**) with no intermediate pixels between the corners:

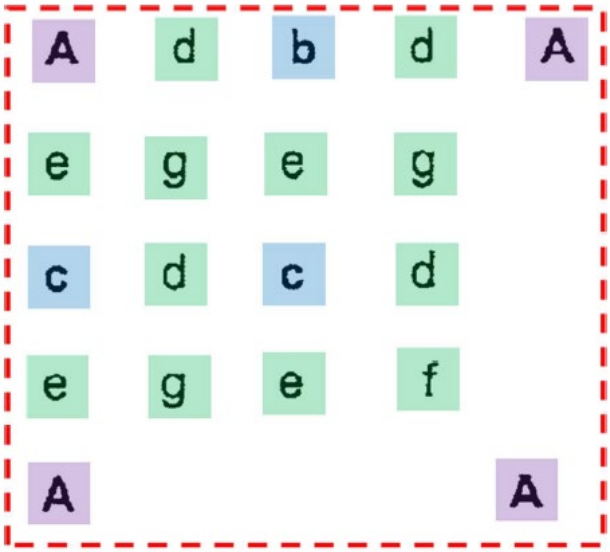


Fig. 13(a)

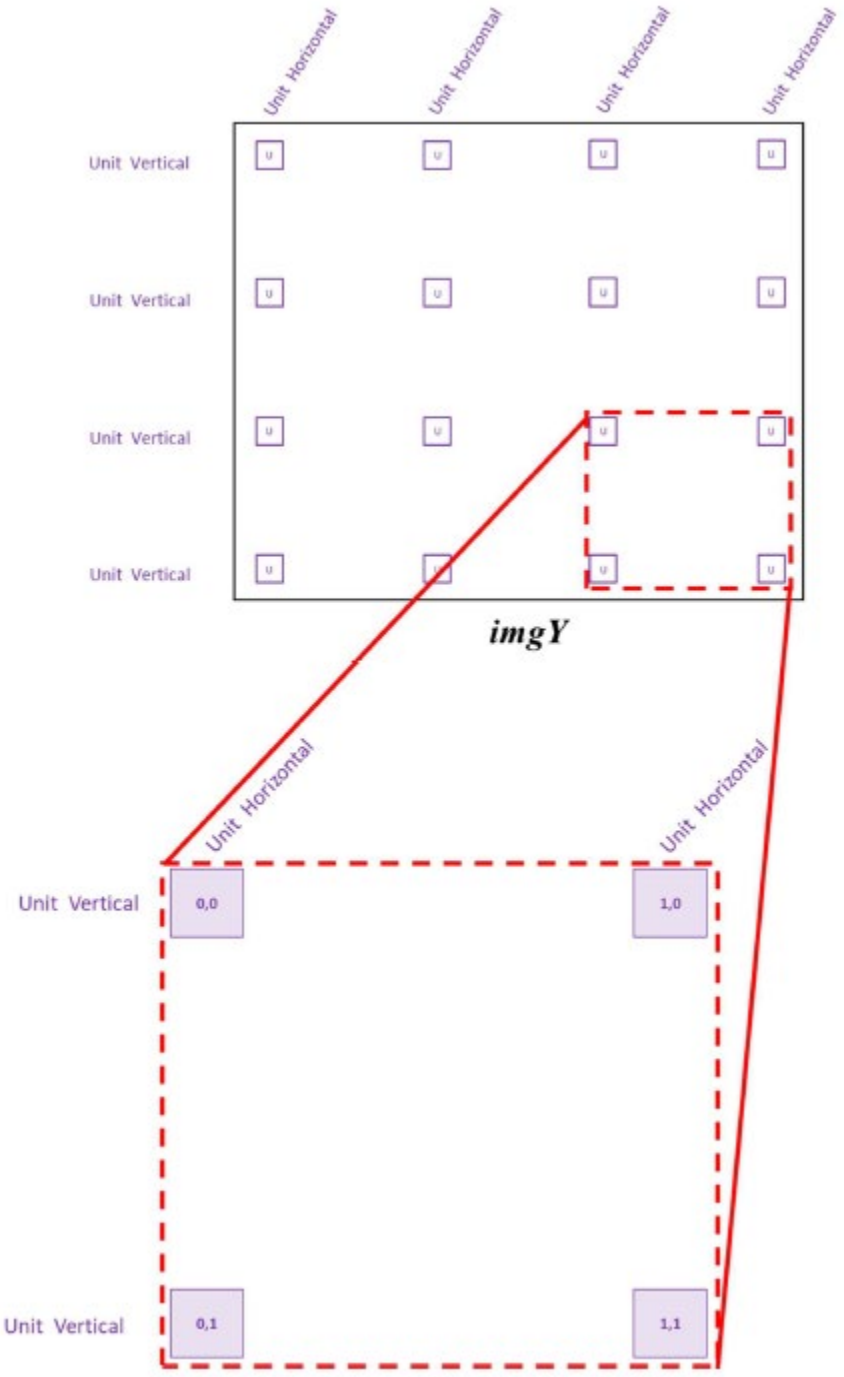
EX-1001, Figure 13(a), 8:38-9:27 ("[S]eparate notation, defined in FIG. 13a, will be used in the discussion of the sub-pixel value interpolation scheme used in TML6…. [T]he letter A represents the location of pixels in the image data representing a frame of a video sequence…."), 11:25-13:18 (describing TML6's interpolation of ½ sub-pixels b and c, and ¼ sub-pixels d-g using "pixels A" with reference to Figure 13(a)). The figure above illustrates that the unit pixels "A" (**purple**) and sub-pixels "b" to "g" (**blue** and **green**) are arranged in rows and columns. I agree all of these things were known in the prior art.

(ii) Element 1[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the method comprising causing an apparatus to:"

141. I understand that this limitation forms part of the preamble of this claim and therefore may not be limiting. To the extent that the Board treats Element 1[pre][b] as limiting, the prior art teaches it.

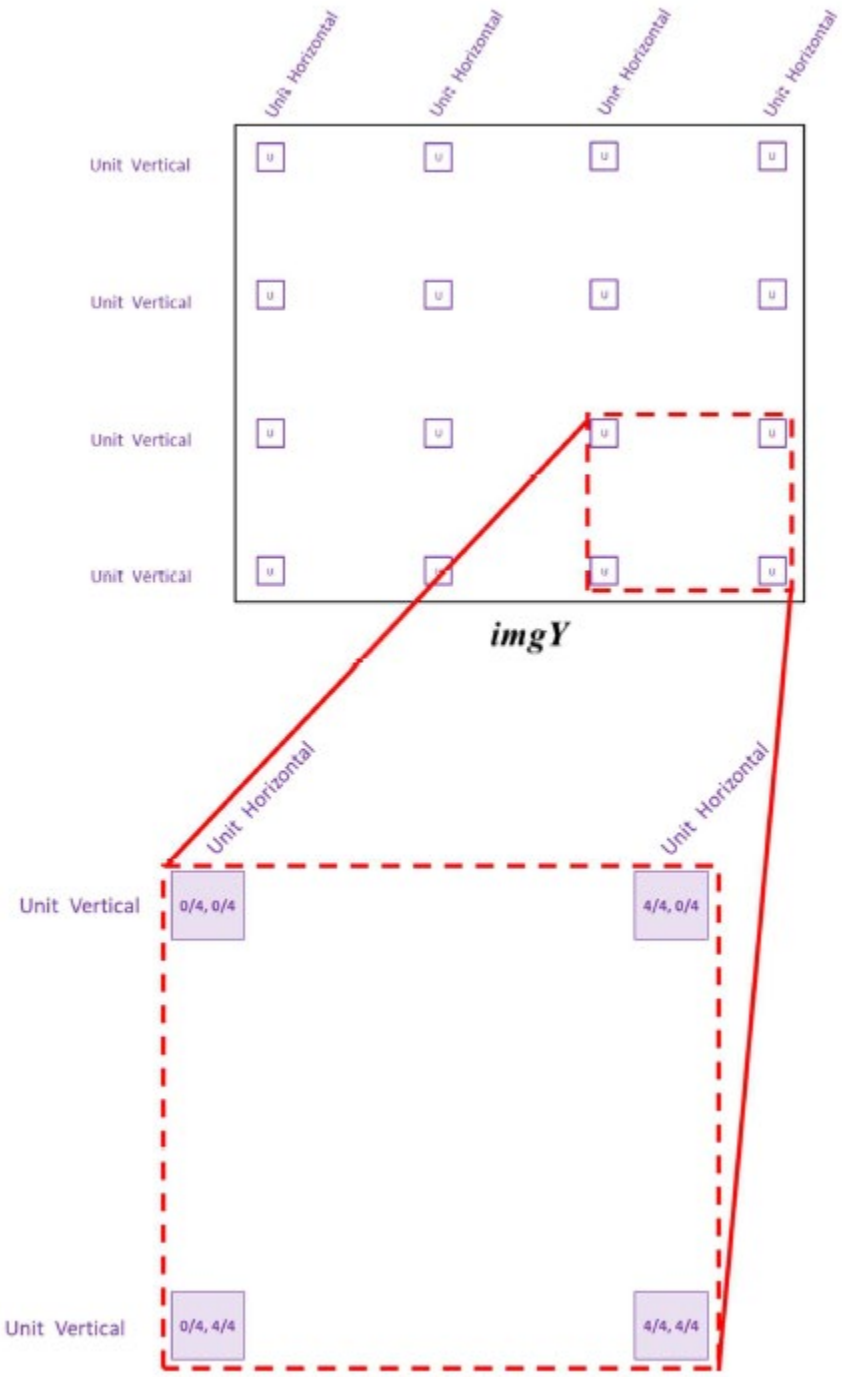
142. TML6 explains that pixel and sub-pixel locations are representable mathematically within the rectangular bounded region.

143. The following is described above for Element 1[pre][a]. The data structure ***imgY*** stores values for unit pixels U (**purple**). And, as illustrated below, four of the pixels U (**purple**) are located in the rectangular bounded region (**red**).



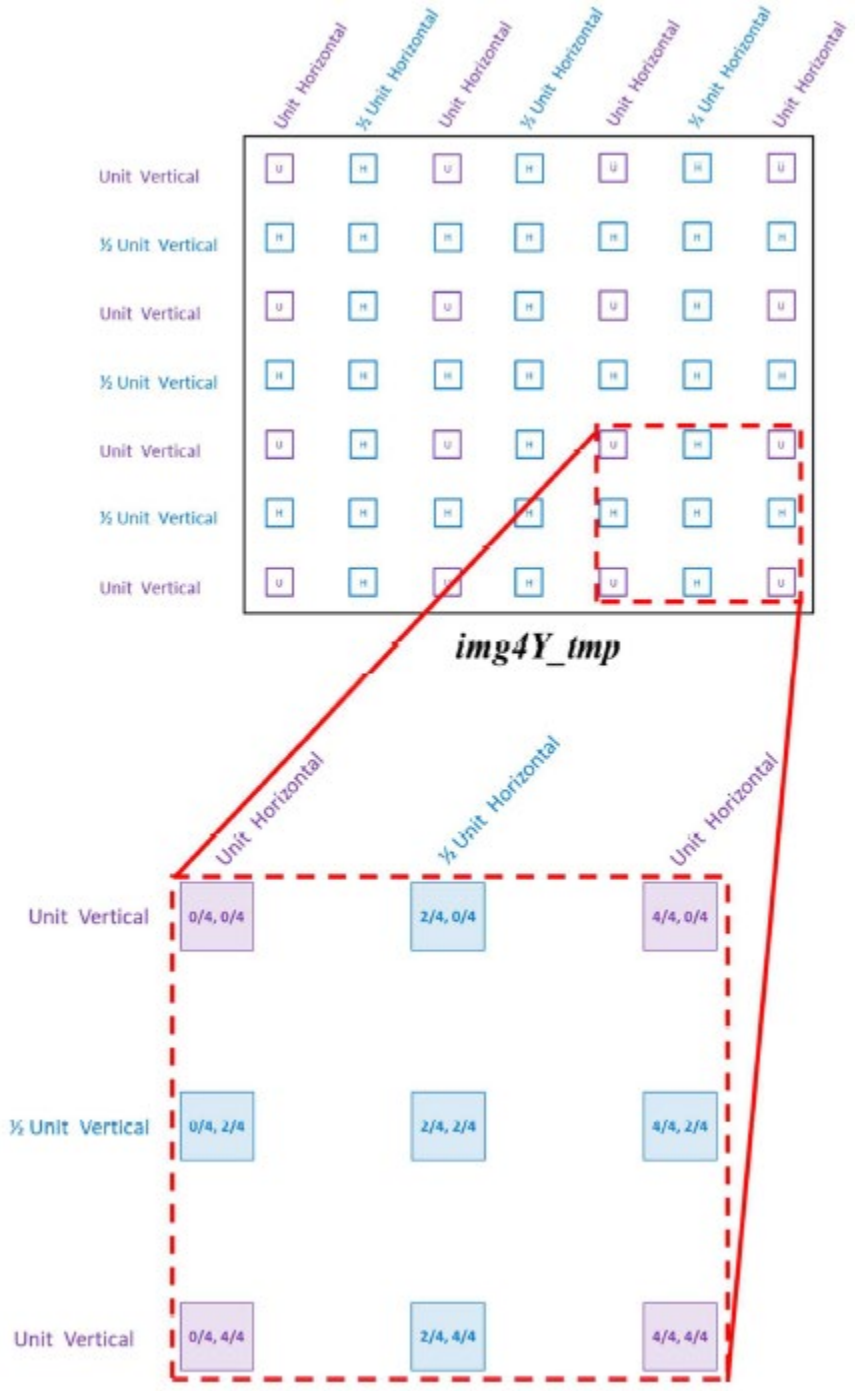
Within this bounded region (**red**), the four pixels U (**purple**) can be represented mathematically using the Cartesian co-ordinate system—the upper left pixel U has the co-ordinates (0,0), the upper right pixel U has the co-ordinates (1,0), the lower

left pixel U has the co-ordinates (0,1), and the lower right pixel U has the co-ordinates (1,1). These co-ordinates can also be represented with the notation “0/4” (which equals 0) and “4/4” (which equals 1), as illustrated below.



See Element 1[pre][a], *supra*.

144. Additionally, the ***oneforthpix()*** function interpolates ½ sub-pixels (**blue**) based on the values of the unit pixels U (**purple**) in the data structure ***imgY*** and stores these pixel values in the data structure ***img4Y_tmp***. *See* Element 1[pre][a], *supra*. Since the ½ sub-pixels (**blue**) are located halfway between the unit pixels U (**purple**) in the vertical direction, or the horizontal direction, or both, they can be expressed as x,y co-ordinates 0/4, 2/4 (= 1/2), and/or 4/4 as illustrated below.



145. Likewise, the ***oneforthpix()*** function interpolates ¼ sub-pixels (**green**) based on the unit pixels U (**purple**) and the previously-interpolated ½ sub-pixels (**blue**) and stores the pixel and sub-pixel values in the data structure ***mref***, as

noted above. *See* Element 1[pre][a], *supra*. Since the ¼ sub-pixels (**green**) are located a quarter-way between the unit pixels U (**purple**) in the vertical direction, or the horizontal direction, or both, they can be expressed with the x,y co-ordinates, as illustrated below.



See Element 1[pre][a], *supra*.

146. TML6 thus discloses, as shown above, that the locations of the pixels (**purple**) and sub-pixels (**blue** and **green**) can be represented mathematically within the rectangular bounded region (**red**) using the co-ordinate notation $K/2^N$ for the "horizontal co-ordinates" and notation $L/2^N$ for the "vertical co-ordinates." The horizontal co-ordinates include: 0/4 (K=0; N=2), 1/4 (K=1; N=2), 2/4 (K=2; N=2), 3/4 (K=3; N=2), and 4/4 (K=4; N=2). The vertical co-ordinates include: 0/4 (L=0; N=2), 1/4 (L=1; N=2), 2/4 (L=2; N=2), 3/4 (L=3; N=2), and 4/4 (L=4; N=2).

147. As further noted above, K and L are also integers having respective values between zero and 2^N (*i.e.,* 2^2=4). Although claim 1 states that K and L are "positive integers having respective values between zero and 2^N," it also states that K and L can equal zero (Elements 1[a]-1[c], infra). Therefore, while zero is not a positive integer, for my analysis, I assume that K and L can equal zero, as set forth in the claim.

148. Furthermore, the value of "N" is equal to 2, as noted above, which is a positive integer greater than one. And TML6 discloses interpolating up to ¼ sub-pixels, and thus, discloses that that N corresponds to a particular degree of sub-pixel value interpolation (*i.e.,* interpolating ¼ ($1/2^N$; N=2) sub-pixels). Moreover, TML6 is source code that a POSITA would have understood is designed to execute on a processor, such as a GPU. *See generally*, TML6. Thus, TML6 teaches causing an apparatus to perform the interpolating steps, as discussed below with respect to

Elements 1[a]-1[c], because it would cause the processor to execute the commands that perform the interpolation. *Id.*

149. In its description of TML6 with reference to Figure 13(a) (reproduced below), the '273 patent explains that, in TML6, the locations of the pixels (**purple**), ½ sub-pixels (**blue**), and ¼ sub-pixels (**green**) can be represented mathematically within the rectangular bounded region (red) using the co-ordinates $K/2^N$, $L/2^N$. This is illustrated below. K and L are integers that have values between zero and four (*i.e.*, 2^N; N=2), and N equals two, which is an integer greater than one and that represents a particular degree of sub-pixel value interpolation (*i.e.,* interpolating ¼ ($1/2^N$; N=2) sub-pixels).

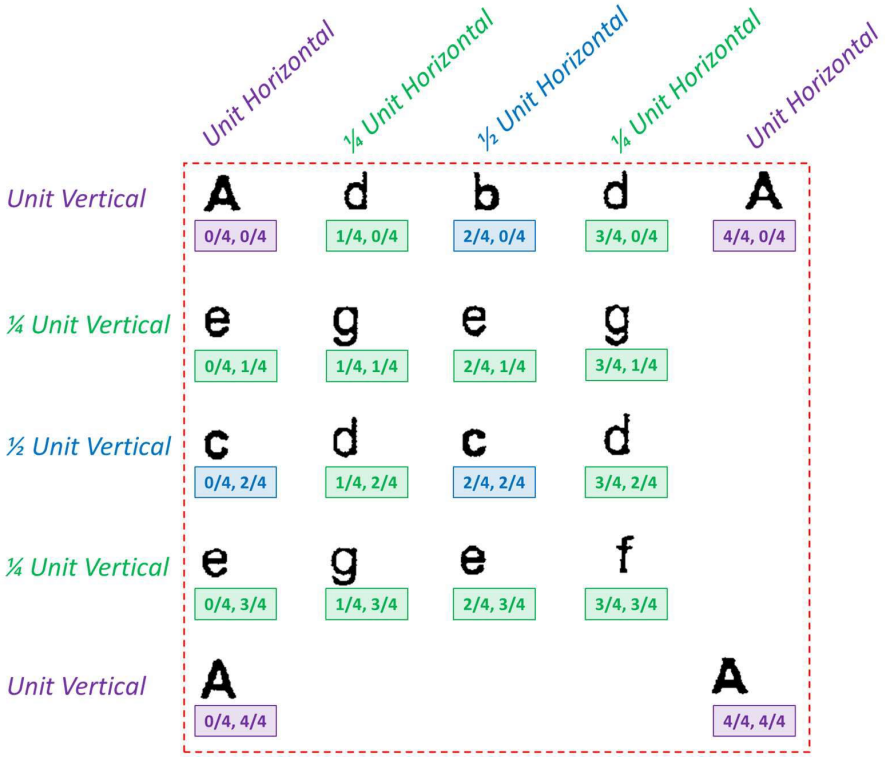


Fig. 13(a)

EX-1001, Figures 13(a)-13(c), 8:38-9:53 ("A separate notation, defined in FIG. 13*a*, will be used in the discussion of the sub-pixel value interpolation scheme used in TML6… [T]he letter A represents the location of pixels in the image data representing a frame of a video sequence…. The term 'unit horizontal location' is used to describe the location of any sub-pixel that is constructed in a column of the original image data….'[H]alf horizontal location' is used to describe the location of any sub-pixel that is constructed in a column that lies at half pixel resolution. Sub-pixels b, c, and e…fall into this category….'[Q]uarter horizontal location' refers to any sub-pixel that is constructed in a column which lies at quarter-pixel

resolution, such as…sub-pixels d and g in FIG. 13*a*….”), 11:25-13:18 (describing TML6’s interpolation of ½ and ¼ sub-pixels). I agree all these things were known in the prior art.

(iii) Element 1[a]: “interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel”

150. The combination of TML6 and Fandrianto discloses Element 1[a]. Both prior art references disclose interpolating a sub-pixel value with odd values of K and L (e.g., K=1; L=1 or 3). For example, TML6’s ***oneforthpix()*** function interpolates the highlighted ¼ sub-pixels (**green**)[13] located at the co-ordinates (¼,¼) (*i.e.,* $K/2^N$, $L/2^N$ (K=1, L=1, N=2)) (“¼,¼ sub-pixel”), as illustrated below:

[13] The illustrations in Element 1[a] and elsewhere in my declaration show sub-pixels within one rectangular bounded region of TML6 and Fandrianto. However, corresponding sub-pixels in other rectangular bounded regions are interpolated in



TML6, 58 (/tml6pre.zip/lencod/src/image.c ll. 593-625); Element 1[pre][a], *supra*. In my analysis that follow, the pixels and sub-pixels positioned at co-ordinates (x,y) are referred to as "x,y pixels" and "x,y sub-pixels."

151. In the above illustration, the highlighted ¼,¼ sub-pixel (**green**) is interpolated according to a weighted average of the four values of the pixel (**purple**) and the three ½ sub-pixels (**blue**) referenced in Element 1[a] (and circled

---

the same way. TML6, 39 (/tml6pre.zip/lencod/src/lencod.c, ll. 1338-1344), 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632); Fandrianto, Figure 5, 5:26-32, 8:58-9:32, 10:51-58.

above), with weights equal to "1." Specifically, the highlighted ¼,¼ sub-pixel (**green**) is interpolated using a weighted average of the following: (1) The "nearest-neighbouring pixel" (*i.e.,* the 0/4,0/4 pixel); (2) The 2/4,2/4 sub-pixel "situated at the co-ordinates ½,½" (*i.e.,* ½,½=2/4,2/4)); (3) The 0/4,2/4 sub-pixel "having co-ordinates with even values of K and L, including zero" (*i.e.,* co-ordinates $K/2^N$, $L/2^N$, where K=0; L=2; N=2) and "diagonally-opposed" to the 2/4,0/4 sub-pixel; and (4) The 2/4,0/4 sub-pixel "having co-ordinates with even values of K and L, including zero" (*i.e.,* co-ordinates $K/2^N$, $L/2^N$, where K=2; L=0; N=2) and "diagonally-opposed" to the 0/4,2/4 sub-pixel.

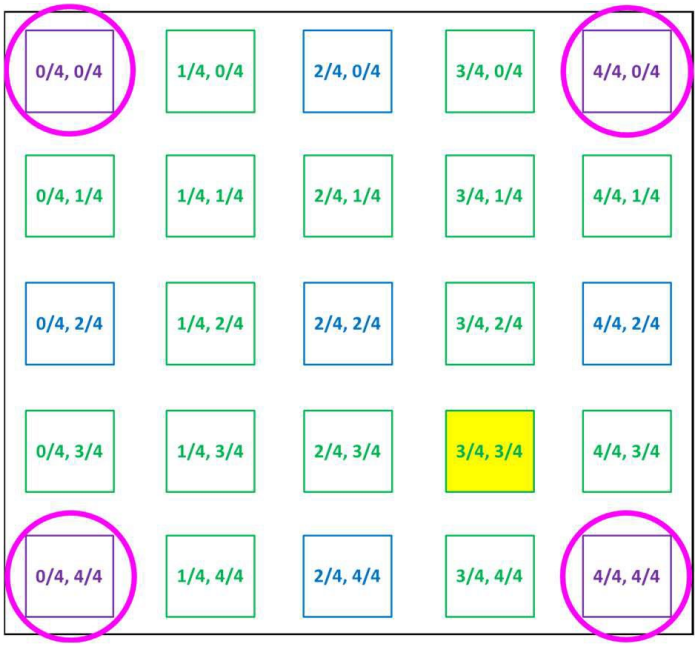
```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                    ][i/2                    ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                    ] +
612                       img4Y_tmp[j/2                    ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-625). The sub-pixels described above are located within a quadrant (**orange**) of the rectangular bounded region. This quadrant is defined by the sub-pixel having co-ordinates ½,½ (*i.e.,* 2/4,2/4 sub-pixel (**blue** and circled)) and the nearest-neighbouring pixel (*i.e.,* 0/4,0/4 pixel (**purple** and circled)). I note that (1/2,1/2) and (2/4,2/4) refer to the same sub-pixel in half- and quarter-pixel resolution, respectively, because they represent equivalent positions within their respective resolution grids.

152. Similarly, as illustrated below, in TML6, the ***oneforthpix()*** function interpolates the highlighted ¼,¾ sub-pixel (**green**) on the right and located at the co-ordinates (¼,¾) (*i.e.,* $K/2^N$, $L/2^N$ (K=1, L=3, N=2)), and the highlighted ¾,¼ sub-pixel (**green**) on the left and located at the co-ordinates (¾,¼) (*i.e.,* $K/2^N$, $L/2^N$ (K=3, L=1, N=2)) in the same fashion as the ¼,¼ sub-pixel, using a weighted average of the four circled values including the nearest-neighbouring pixel (**purple**) and the three sub-pixels (**blue**). These highlighted sub-pixels (**green**), circled pixels (**purple**), and circled sub-pixels (**blue**) in the figures below likewise correspond to the "nearest-neighbouring pixel" (*i.e.,* 4/4,0/4 or 0/4,4/4 pixel), "the sub-pixel situated at the co-ordinates ½,½" (*i.e.,* 2/4,2/4 sub-pixel), and the "pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero" (*i.e.,* 2/4,0/4 and 4/4,2/4 sub-pixels or 0/4,2/4 and 2/4,4/4 sub-pixels) recited in Element 1[a] for the reasons I discuss above.



TML6, 58-59 (/tml6pre.zip/lencod/src/image.c, ll. 593-632). Finally, the highlighted ¾,¾ sub-pixel (**green**) is interpolated using the nearest four unit pixels (**purple**).



```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                  ][i/2                  ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                  ] +
612                       img4Y_tmp[j/2                  ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                          + 2048)/4096));
614
615                   /* "funny posision" */
616                   if( ((i&3) == 3)&&(((j+1)&3) == 3))
617                   {
618                       mref[img->frame_cycle][j+1][i] = (mref[img->
                        frame_cycle][j-2          ][i-3          ] +
619                           mref[img->frame_cycle][min(je2,j+2)][i-3
                                ] +
620                           mref[img->frame_cycle][j-2          ][min(ie2,i
                            +1)] +
621                           mref[img->frame_cycle][min(je2,j+2)][min(ie2,i
                            +1)] + 2 )/4;
622                   }
623               }
624           }
625
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-624).

153. The ’273 patent explains, with reference to Figure 13(a), that TML6 interpolates the value for the ¼,¼ sub-pixel “g” (**green**) by using a weighted average of the same unit pixel A (**purple**) and three ½ sub-pixels b, c, and c (**blue**).

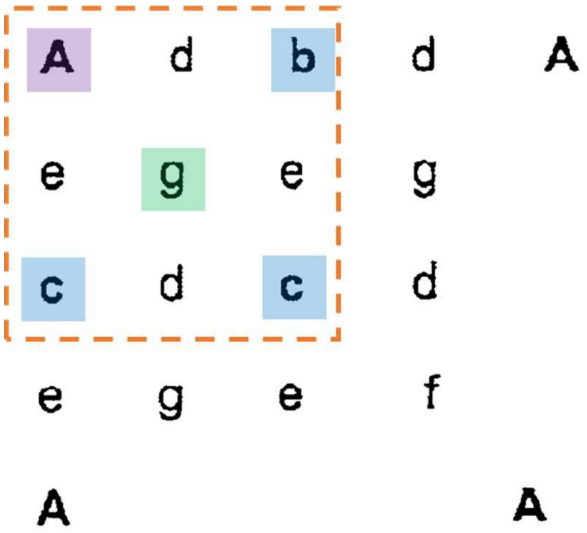


Fig. 13(a)

EX-1001, Figure 13(a), 12:34-40 ("Values for … g are computed using the value of the nearest original pixel A and the intermediate values of the three nearest neighbouring ½ resolution sub-pixels [b, c, and c], according to g=(1024A+32b+ $32c_1+c_2$+2048)/4096"), 12:41-45 ("¼ resolution sub-pixel f is interpolated by averaging the values of the 4 closest pixels….").

154. As I describe above, TML6 interpolates the highlighted ¼,¼, ¼,¾, and ¾,¼ sub-pixels in the figures shown above by using a weighted average of (1) "the value of a nearest-neighbouring pixel," (2) "the value of the sub-pixel situated at co-ordinates ½,½," and (3-4) "the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero." I do note that, per one interpretation of the claim, TML6 potentially may not teach

interpolating the ¼,¼, ¼,¾, ¾,¼ or ¾,¾ sub-pixel according to a predetermined choice of both (1) a weighted average the values of a pair of the nearest-neighbouring pixel and the sub-pixel located at co-ordinates ½,½ and (2) a weighted average of the values of the pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero. If the claim is interpreted in this manner, and if the Board determines that TML6 does not suggest this feature, the combined teachings of TML6 and Fandrianto do, as I describe below.

155. For example, as shown below in Figure 5, Fandrianto shows four unit pixels X (**purple**) within a rectangular bounded region (**red**) and discloses interpolating a value for the highlighted ¼,¾ sub-pixel Q2 (**green**).

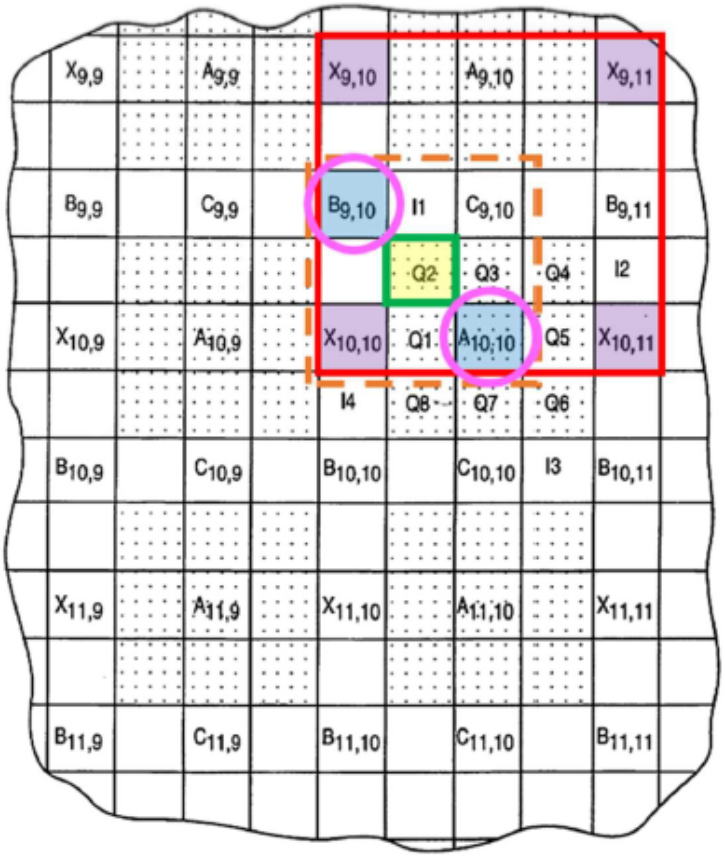


FIG. 5

Fandrianto, Figure 5, 5:26-32, 8:58-9:32. Fandrianto explains that "a great many permutations of the order of derivation of [¼ sub-]pixels Q1-Q8 [*i.e.*, including sub-pixel Q2, highlighted above] may be practiced, if desired." Fandrianto, 8:36-10:58, 10:51-58. This refers to the fact that, with reference to Figure 5, Fandrianto describes deriving quarter pixels Q1-Q8 surrounding sub-pixel $A_{10,10}$ in sequential order. Fandrianto clarifies, however, that "the order in which [Q1-Q8] are generated and compared, and the selection of obsolete full and half pixel estimation search blocks to be overwritten are…discretionary. **The order shown**

in FIG. 5…is therefore illustrative." Fandrianto, 8:15-21. Fandrianto therefore teaches that the ¼ sub-pixel Q2 in **green** is interpolated by diagonal interpolation of the two ½ sub-pixels A and B (**blue** and circled) diagonally adjacent to the sub-pixel Q2 in **green**.

156. I will now explain in detail the meaning of Fandrianto's statement that "[t]he order shown in FIG. 5…is therefore illustrative" and its relevance to Fandrianto's disclosure of diagonal interpolation. With reference to Figure 5 (below left), Fandrianto describes deriving highlighted quarter pixels Q1-Q8 surrounding sub-pixel $A_{10,10}$ in sequential order using the two-memory apparatus of Figure 3B (below right) comprising DPCM memory 130 and DP memory 124. DPCM memory 130 is a "search memory" in which "[t]he search window…[for] a previously processed frame" is stored. DP memory 124 is "used…to store…temporarily needed blocks," e.g., "an input or image block" in P block 80 (pink) for "successively compar[ing]…with the search blocks in the search window" stored in DPCM memory 130.

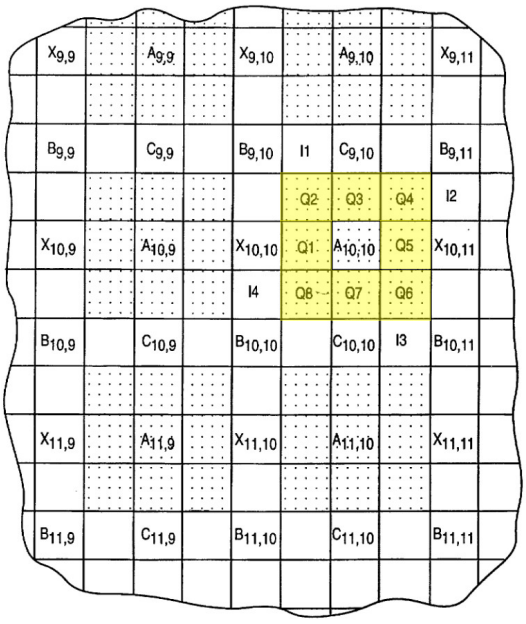


FIG. 5

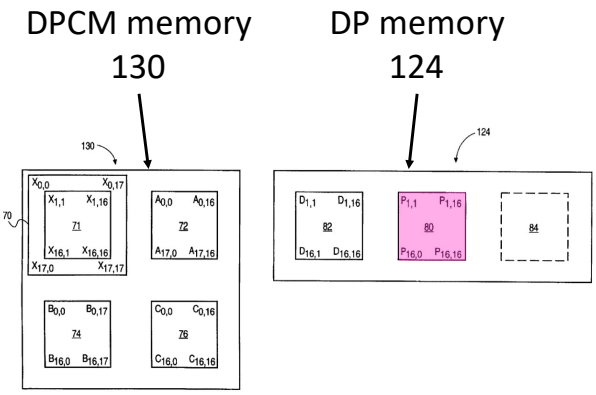


FIG. 3B

Fandrianto, Figure 5, Figure 3B, Abstract, 3:16-43, 3:57-66, 4:18-5:5, 5:26-32, 8:16-10:58, 12:9-15. Though Fandrianto focuses for simplicity's sake on quarter pixels Q1-Q8 around sub-pixel $A_{10,10}$, its technique is block-based and calculates all the "highlighted areas about the elements A" in Figure 5, as described in detail below. *See id.*, 8:24-28.

157. First, pixel block $X_{0,0}$-$X_{17,17}$ (**purple**)[14] is stored in "X" block 70 (**purple**) of DPCM memory 130.

[14] In Fandrianto, the subscript notation "$X_{r,c}$" indicates a range of pixel positions within a two-dimensional array "X" where the first subscript "r" denotes the row and the second subscript "c" denotes the column. *See* Fandrianto, 5:51-55.

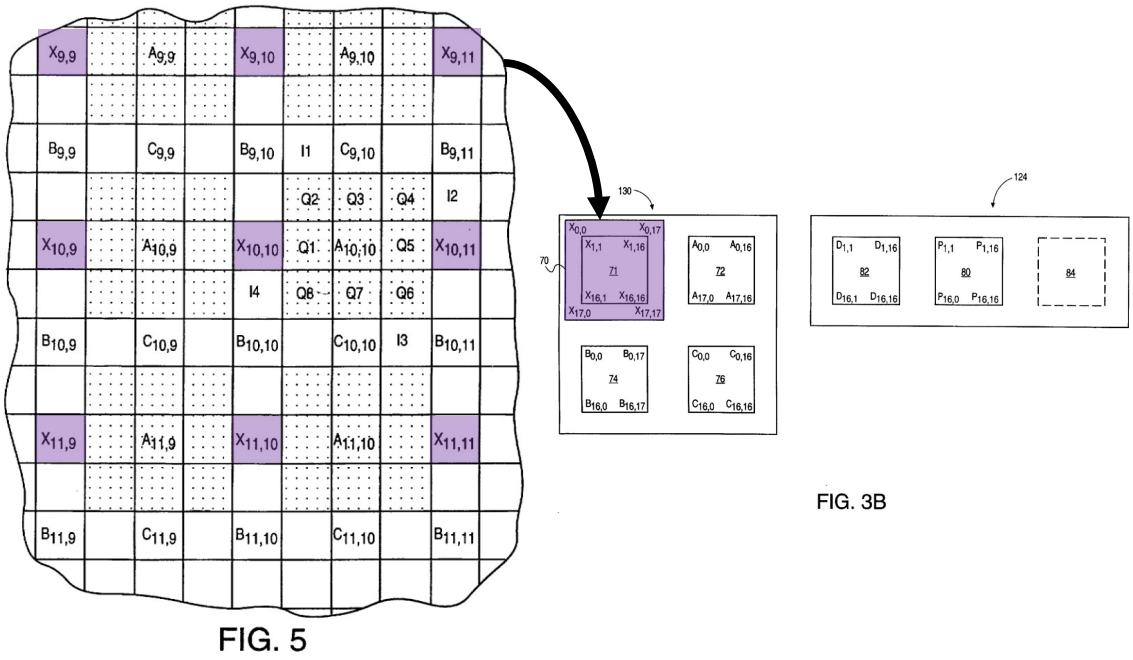


FIG. 5

FIG. 3B

Fandrianto, Figure 5, Figure 3B, Abstract, 5:37-41. Next, "'A' block 72 [red], 'B' block 74 [blue], and 'C' block 76 [**green**] are interpolated from…X block 70[.]"

The interpolated sub-pixels are annotated using like colors below.

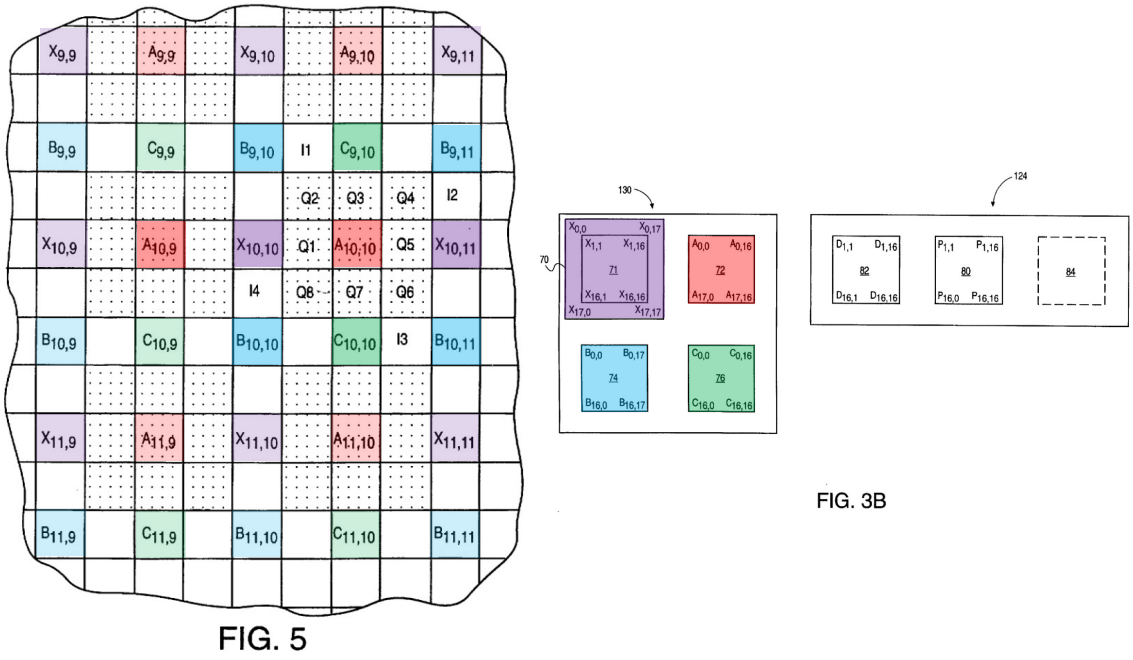


FIG. 5

FIG. 3B

Fandrianto, Figure 5, Figure 3B, 5:41-44, 6:18-7:2. Fandrianto then describes an iterative process for generating quarter pixels Q1-Q8 in sequential order. *Id.*, 8:29-10:50. Each quarter pixel is calculated based on the data of the preceding quarter pixel and is stored in A block 72, progressively overwriting the preceding quarter pixel data. *Id*.

158. Fandrianto notes "that a great many permutations of the order of derivation of pixels Q1-Q8 may be practiced[.]" Fandrianto, 10:51-53; *see also id.*, 8:15-21 ("[T]he order in which [Q1-Q8] are generated and compared, and the selection of obsolete full and half pixel estimation search blocks to be overwritten are…discretionary. The order shown in FIG. 5…is therefore illustrative."). In fact, I note that Fandrianto in 10:51-58 provides an explicit example in which the Q2 sub-pixels are derived from a horizontal interpolation of the Q3 sub-pixels and the results of a vertical interpolation of the data in the B block and the X block, which a POSITA would have understood to mean that Q3 is interpolated before Q2, despite the fact that Q3 obviously comes numerically after Q2.

159. As illustrated below, Fandrianto teaches that "the Q2 pixels [yellow] may be derived from a **diagonal interpolation** of the data in the A block 72 [red] and the B block 74 [**blue**]." Because Fandrianto previously established that both the order in which Q1-Q8 are generated and compared and the selection of obsolete full and half pixel estimation search blocks to be overwritten are

discretionary, A block 72 and B block 74 contain values for A pixels [red] and B pixels [**blue**], respectively.

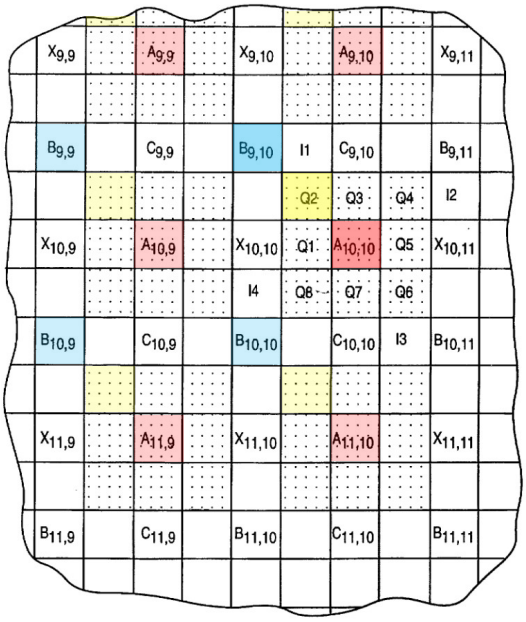


FIG. 5

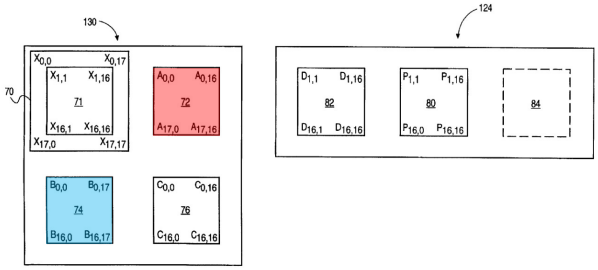


FIG. 3B

Fandrianto, Figures 5, 3B, 10:51-58.

160. This ¼,¾ sub-pixel Q2 (**green**) further has co-ordinates $K/2^N$, $L/2^N$ (K=1, L=3, N=2) with odd values of K and L. Moreover, this sub-pixel Q2 is interpolated by using a weighted average of the values of a pair of diagonally-opposed sub-pixels (*i.e.,* the 0/4,2/4 sub-pixel B and the 2/4,4/4 sub-pixel A (both **blue** and circled)). The 0/4,2/4 sub-pixel B and the 2/4,4/4 sub-pixel A have co-ordinates $K/2^N$, $L/2^N$ (K=0 or 2, L=2 or 4, N=2) with even values of K and L, including zero. Fandrianto, 8:58-9:32 ("The [] quarter pixel estimation search block of *Q2* pixels is generated … in accordance with the following

expression $Q2_{r,c}=(Q1_{r,c}+I1_{r,c})/2$ [*i.e.,* a weighted average (with weights equaling '1') for a 'vertical interpolation' embodiment using sub-pixels Q1 and I1]" (emphasis added)), 10:51-58 (in another embodiment, "the Q2 pixels may be derived from a *diagonal interpolation* of the data in the A block … and the B block [*i.e.,* a weighted average $Q2_{r,c}=(A_{r1,c}+B_{r,c})/2$ using sub-pixels A and B (both **blue** and circled)]" (emphasis added)), 8:36-10:58. As illustrated above, these diagonally-opposed 0/4,2/4 sub-pixel B and 2/4,4/4 sub-pixel A are located within a quadrant (**orange** box) of the rectangular bounded region (**red** box). This quadrant is defined by the sub-pixel having co-ordinates ½,½ (*i.e.,* the 2/4,2/4 sub-pixel $C_{9,10}$) and the nearest-neighbouring pixel (*i.e.,* 0/4,4/4 pixel $X_{10,10}$). *Id.*

161. Fandrianto also teaches alternatively interpolating the highlighted ¼,¾ sub-pixel Q2 (**green**) using a weighted average of the value of a nearest-neighbouring pixel (*i.e.,* 0/4,4/4 sub-pixel X) (**purple** and circled) and the value of the sub-pixel C located at co-ordinates ½,½ (**blue** and circled), as shown below.

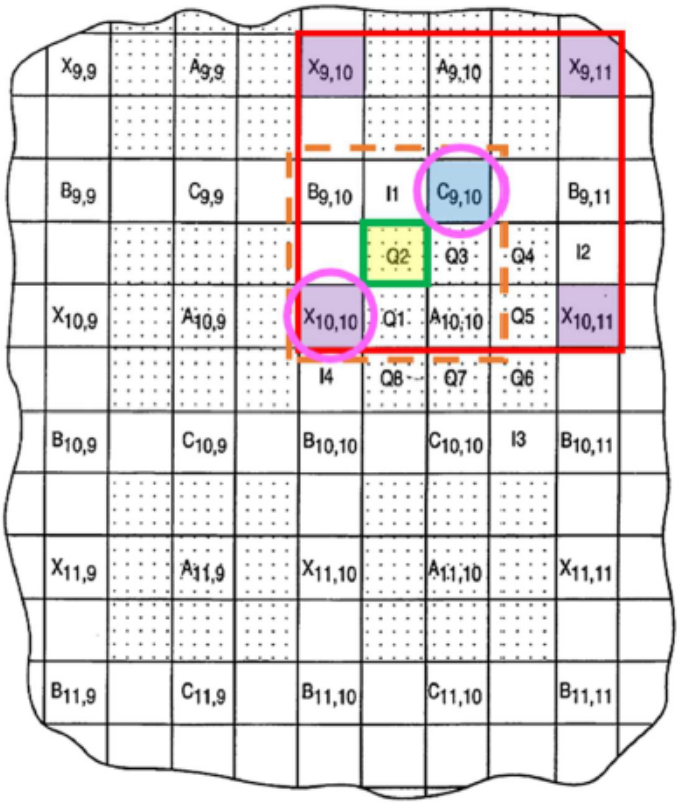


FIG. 5

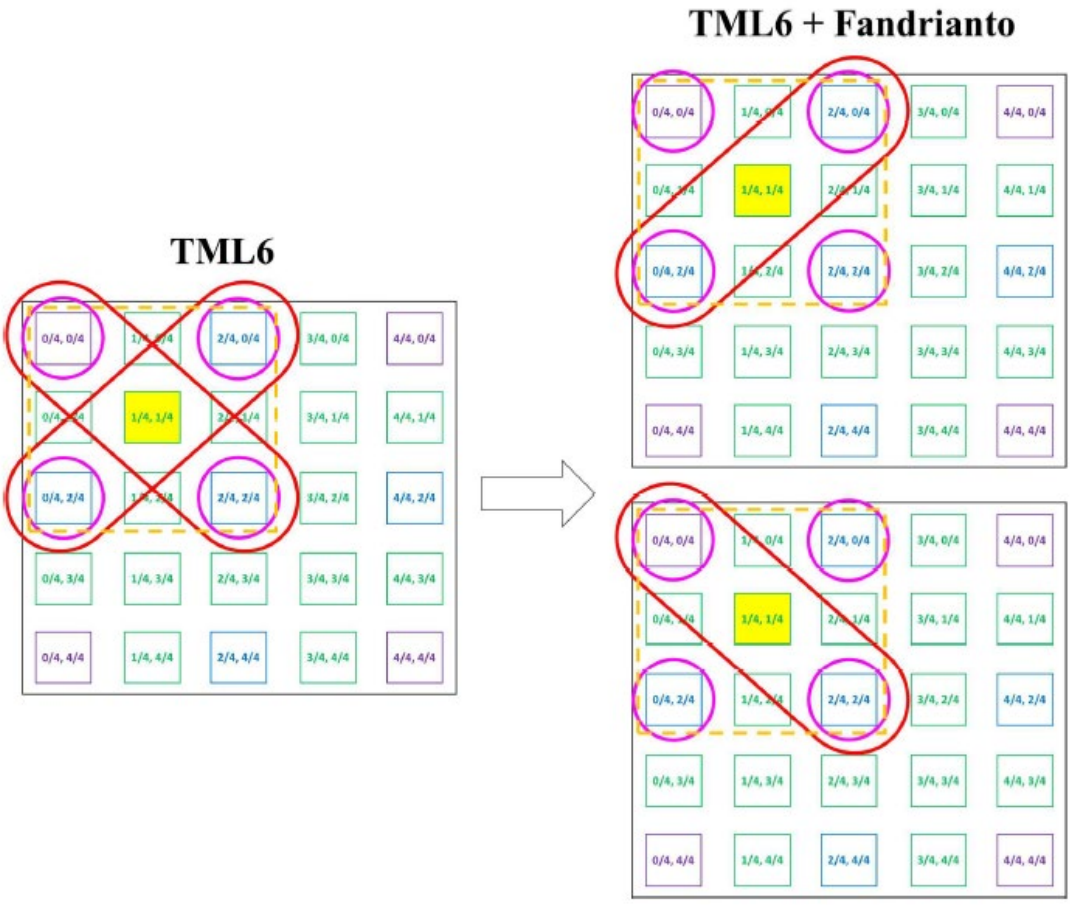
Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 7:52-55 ("[W]hen ***diagonal interpolation*** is also used, all quarter pixel estimation search blocks [including sub-pixel Q2 (**green**)] are interpolated from the full [unit pixel X (**purple**)] and half pixel [C (**blue**)] estimation search blocks."), 8:58-9:32 ("The second quarter pixel estimation search

block of Q2 pixels is generated from a vertical interpolation of the Q1 pixels with the results of a horizontal interpolation of the data in the B block 74 and the C block 76, with the results being stored in the A block 72 (the Q2 pixels progressively overwriting the Q1 pixels) for the absolute displaced block difference calculation, in accordance with the following expression. $\boldsymbol{Q2_{r,c}=(Q1_{r,c}+I1_{r,c})/2}$ where $\boldsymbol{I1_{r,c}=(B_{r\text{-}1,c}+C_{r\text{-}1,c})/2}$. Consider, for example, the interpolation of the Q2 pixel to the upper left of pixel $A_{10,10}$. A collection of pixels containing the pixel group $B_{9,8}$-$B_{9,15}$ in row nine of the B block 74 is addressed on, say, port A of the DPCM memory 130 and the pixel group $B_{9,8}$-$B_{9,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the A port of the ALU 154. At about the same time, a collection of pixels containing the pixel group $C_{9,8}$-$C_{9,15}$ in row nine of the C block 76 is addressed on, say, port B of the DPCM memory 130 and the pixel group $C_{9,8}$-$C_{9,15}$ is selected by shifter 144 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed to the DP memory 124, where it is stored as pixels $I1_{9,8}$-$I1_{9,15}$ in the ninth row of I block 84, a free 16x16 block of memory in the DP memory 124. Next, the pixel group $I1_{9,8}$-$I1_{9,15}$ in row nine of the I block 84 is addressed on, say, port A of the DP memory 124 and presented through the multiplexer 152 to the A port of the ALU 154. At about the same time, a collection

of pixels containing the pixel group $Q1_{10,8}$-$Q1_{10,15}$ in row ten of the A block 72 is addressed on, say, port A of the DPCM memory 130 and the pixel group $Q1_{10,8}$-$Q1_{10,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed back to the DPCM memory, where it is stored as pixels $Q1_{10,8}$-$Q1_{10,15}$ in the tenth row of the A block 72."), 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels Q1-Q8 may be practiced, if desired As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74."). Fandrianto, therefore, additionally discloses that the ¼,¾ sub-pixel Q2 is diagonally interpolated using the nearest-neighbouring pixel $X_{10,10}$ and the sub-pixel $C_{9,10}$ with co-ordinates ½,½ as recited in Element 1[a]. *Id.* As with the other disclosed interpolations, this interpolation is a weighted average of two values of the pixel $X_{10,10}$ and sub-pixel $C_{9,10}$, with weights equal to "1." *Id.,* Figure 5, 6:18-7:2, 8:36-10:50.

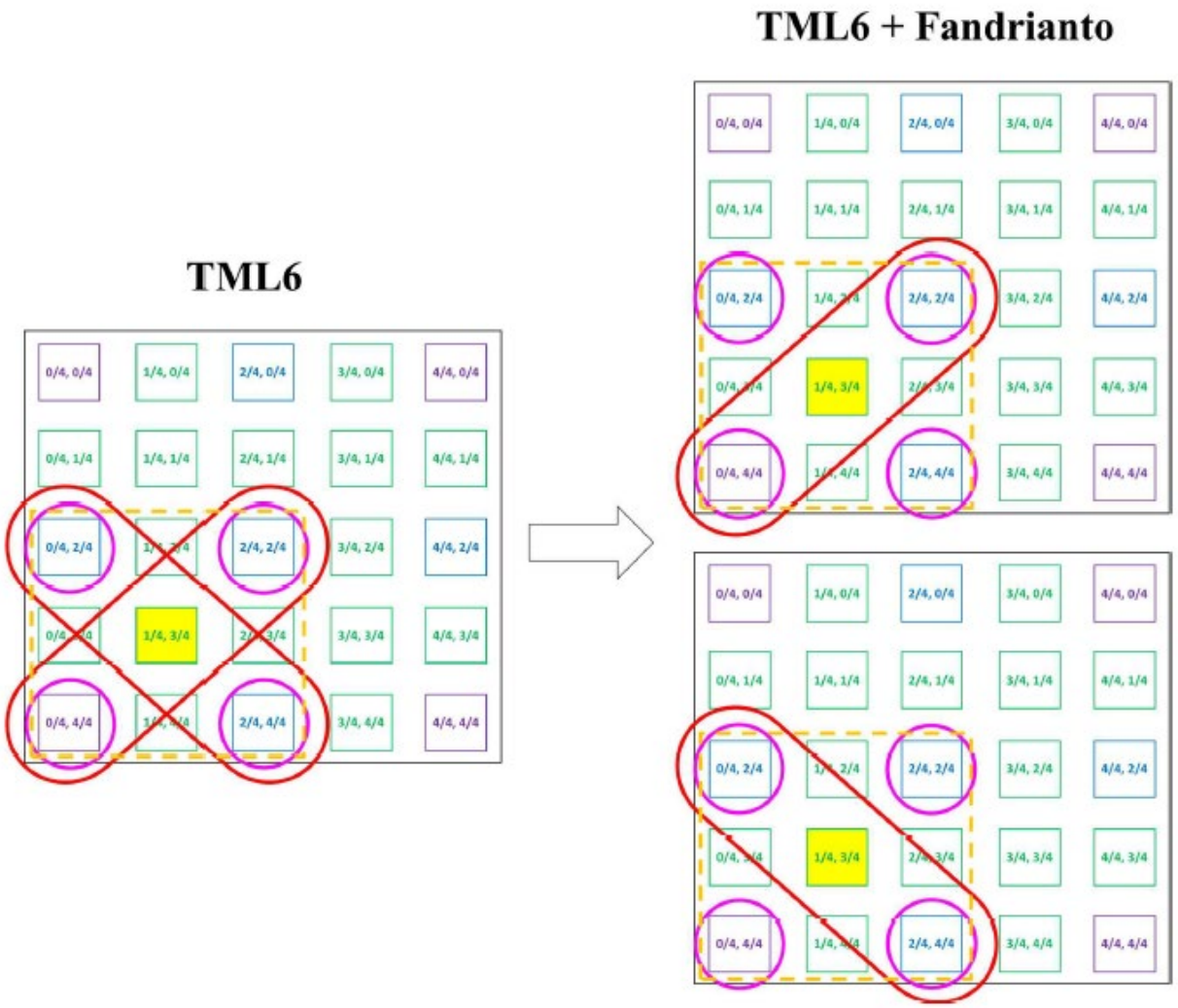
162. As I explain in Section IX.A.1 (motivation to combine), a POSITA would have been motivated to use the interpolation method, as set forth in Fandrianto, for interpolating the ¼,¾ sub-pixel Q2 (**green**) to also interpolate the ¼,¼ sub-pixel, ¼,¾ sub-pixel, ¾,¼ sub-pixel and/or ¾,¾ sub-pixel in TML6. For example, a POSITA would have known how to modify the code of TML6 to use

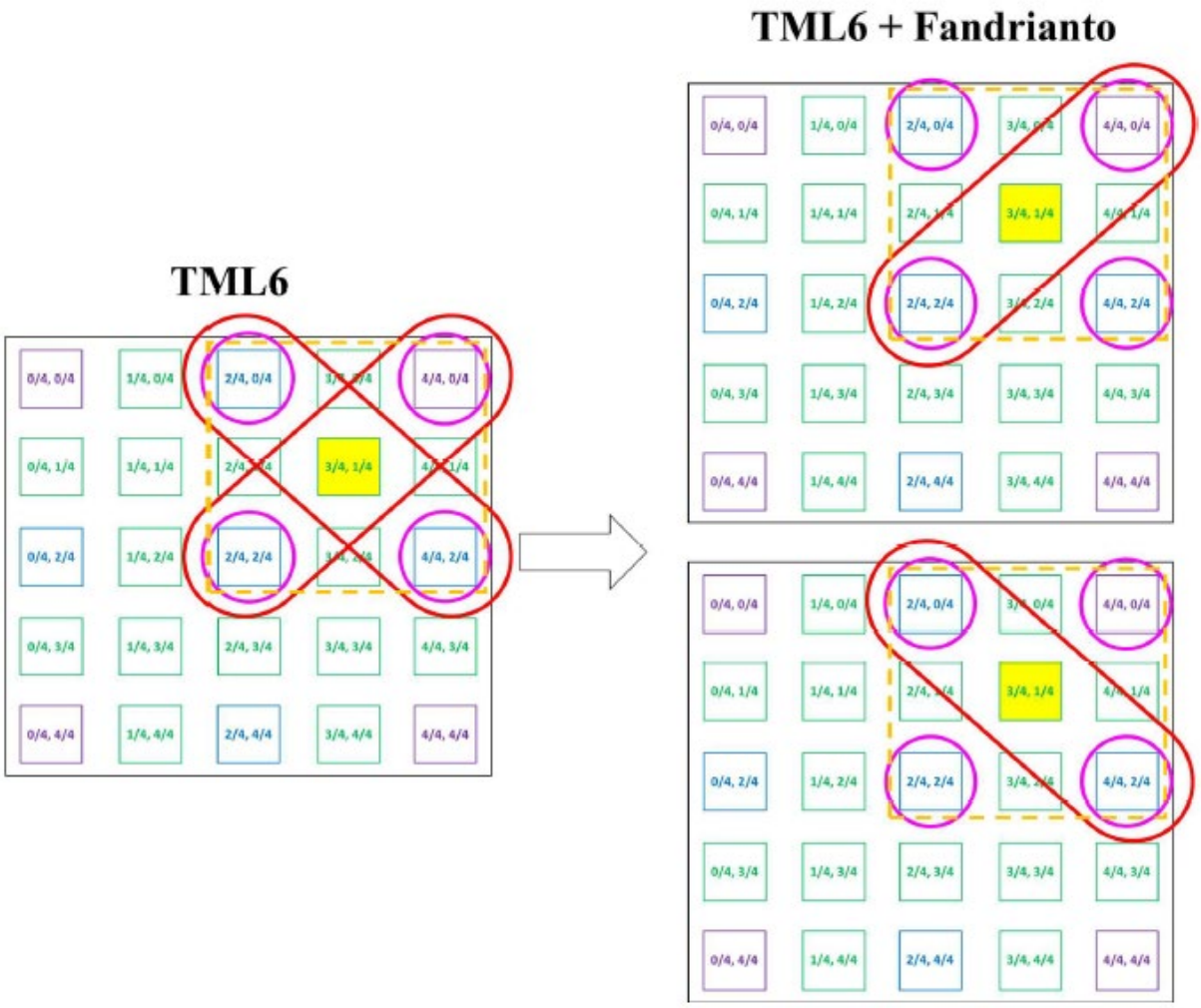
Fandrianto's interpolation method or by using Fandrianto's technique on its hardware. The combined teachings of TML6 and Fandrianto, as illustrated below, thus result in a system or software that interpolates TML6's ¼,¼ sub-pixel (**green**) using (as shown on the bottom right) a weighted average of the value of a nearest-neighbouring 0/4,0/4 pixel (**purple**) and the value of the 2/4,2/4 sub-pixel (**blue**) located at co-ordinates ½,½, ***or*** (as shown on the upper right) a weighted average of the values of a pair of diagonally-opposed sub-pixels (*i.e.,* the 0/4,2/4 and 2/4,0/4 sub-pixels) (**blue**) having co-ordinates with even values of K and L, including zero (*i.e.,* K=0 or 2, L=0 or 2), located within a quadrant (**orange** box) of the rectangular bounded region (**red** box) defined by the sub-pixel having co-ordinates ½,½ and the nearest-neighbouring pixel, as in Fandrianto.

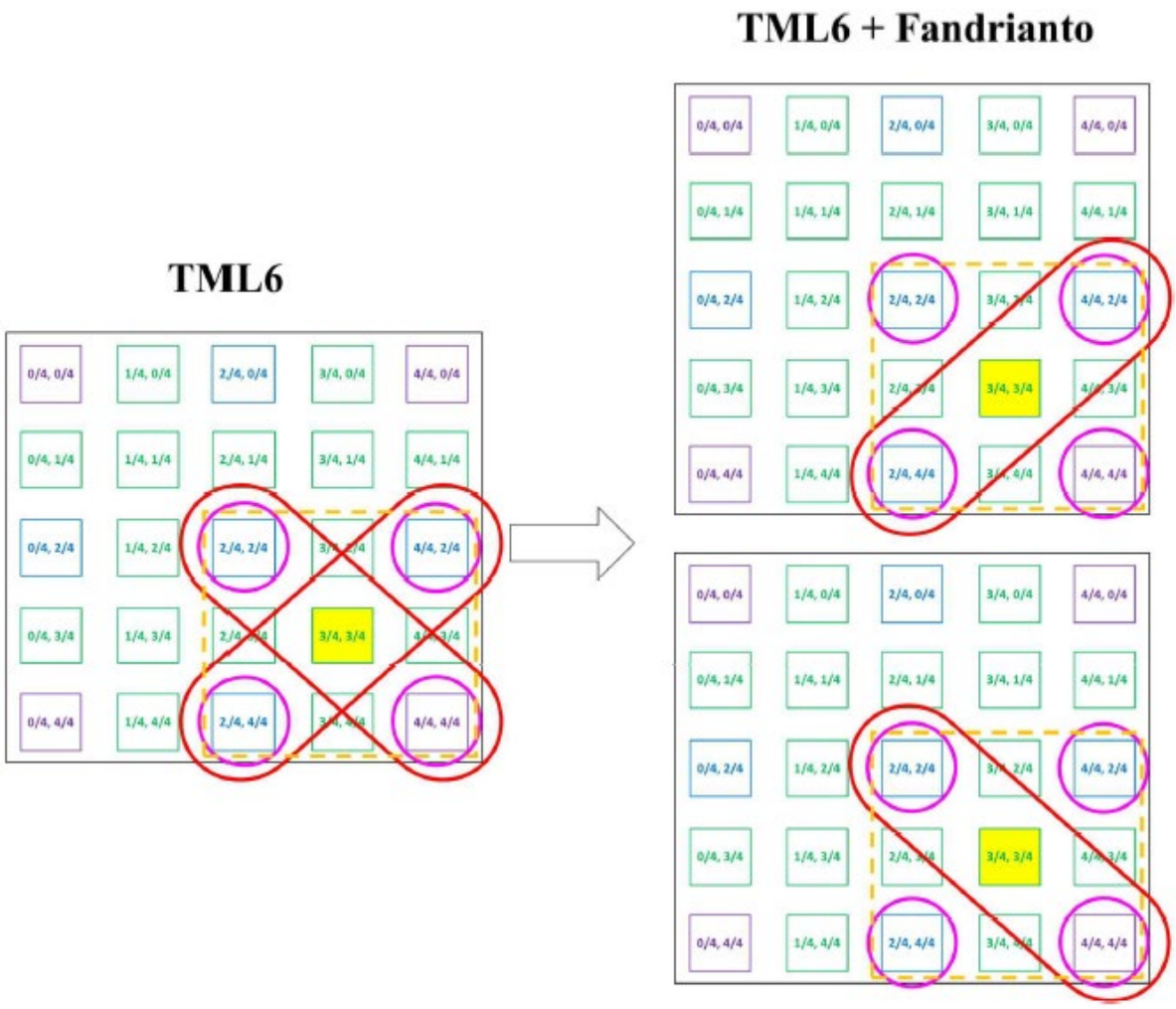


See Section IX.A.1 (motivation to combine TML6 and Fandrianto), *supra*.

Moreover, as illustrated below, the combined teachings of TML6 and Fandrianto teach interpolating TML6's ¼,¾ sub-pixel (**green**), and the ¾,¼ sub-pixel (**green**), and the ¾,¾ sub-pixel (**green**) in the same way.


TML6
TML6 + Fandrianto


TML6
TML6 + Fandrianto



See Section IX.A.1 (motivation to combine TML6 and Fandrianto), *supra*.

163. Furthermore, on the same basis discussed above in Section IX.A.1, *supra*, a POSITA would have found it obvious to interpolate TML6's ¼,¼, ¼,¾, ¾,¼, and/or ¾,¾ sub-pixels using a predetermined choice of the two diagonal interpolation methods depicted above and disclosed in Fandrianto, with each of the two diagonal method used to interpolate at least one of the sub-pixels. Fandrianto is express in its teaching that a finite number of interpolation methods could have

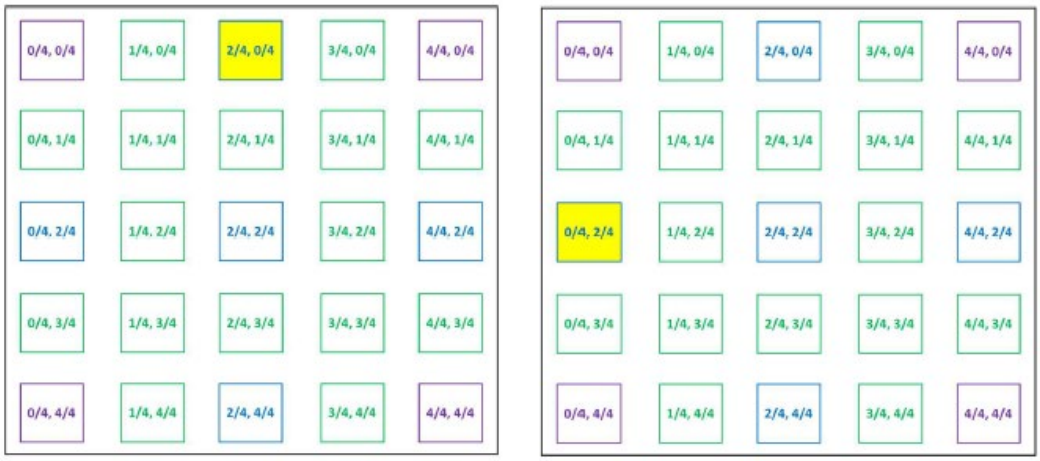
been used to interpolate the ¼,¼, ¼,¾, ¾,¼, and ¾,¾ sub-pixels, including the option of vertical interpolation and the option of two diagonal interpolations. It would, therefore, have been obvious for a POSITA to try these limited options, including making a predetermined choice between these two diagonal-interpolation options, to interpolate any of the ¼,¼, ¼,¾, ¾,¼, and ¾,¾ sub-pixels. Moreover, modifying the TML6 code or Fandrianto's hardware to implement this choice would have been straightforward, and a POSITA would have reasonably expected such a modification to yield the predictable result of interpolating ¼,¼, ¼,¾, ¾,¼, and/or ¾,¾ sub-pixels based on values of nearby pixels and/or sub-pixels.

(iv) Element 1[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

164. TML6 discloses or renders obvious Element 1[b]. As illustrated below, the ***oneforthpix()*** function of TML6 interpolates values for the 2/4,0/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$) with K equal to an even value and L equal zero (K=2, L=0, N=2) and for the 0/4,2/4 sub-pixel having co-ordinates ($K/2^N$,

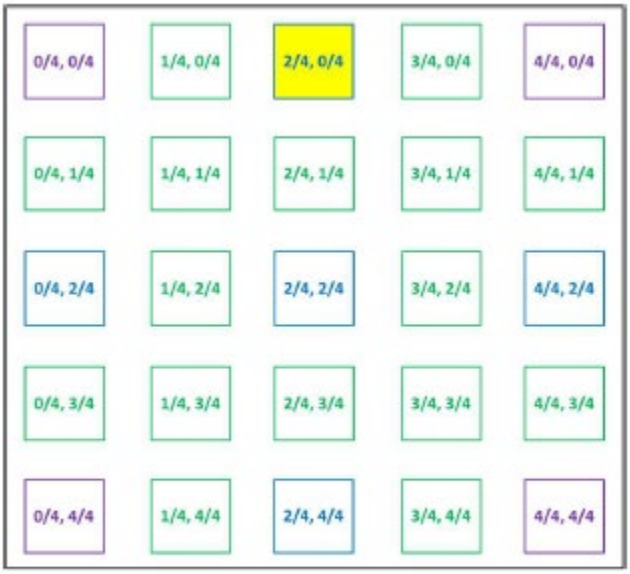
$L/2^N$) with K equal to zero and L equal to an even value (K=0, L=2, N=2).[15] TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 538-592).

[15] In my analysis of Element 1[b], the various illustrations show sub-pixels within one rectangular bounded region of TML6 and Fandrianto. However corresponding sub-pixels in other rectangular bounded regions of pixels and sub-pixels are interpolated in the same manner. TML6, 39 (/tml6pre.zip/lencod/src/lencod.c, ll. 1338-1344), 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632); Fandrianto, Figure 5, 5:26-6:48. Accordingly, TML6 and Fandrianto interpolate multiple sub-pixels with even values of K and L, as claimed. If the claim requires interpolating, in one rectangular bounded region, ***multiple*** sub-pixels having co-ordinates with K equal to an even value and L equal zero and ***multiple*** sub-pixels having co-ordinates with K equal to zero and L equal to an even value, the scope of the claim is unclear, since there is only ***one*** such sub-pixel when N=2. However, TML6 or the combination of TML6 and Fandrianto shows this claimed feature to the same extent that the ’273 patent discloses this claimed feature.



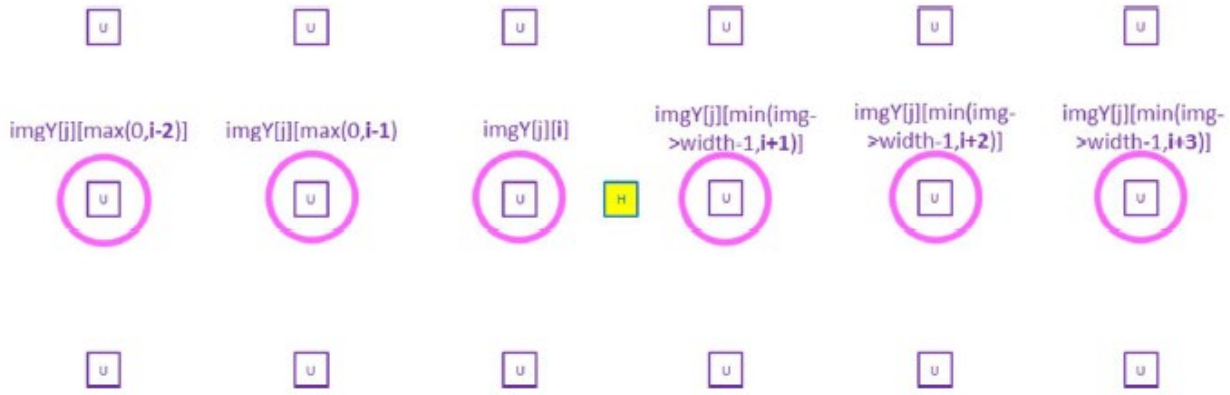
As I explained above for Element 1[a], these 2/4,0/4 and 0/4,2/4 sub-pixels are used to interpolate the values of the sub-pixels having co-ordinates ¼,¼, ¼,¾, ¾,¼ and ¾,¾ with odd values of K and L (K=1 or 3, L=1 or 3). Element 1[a], *supra*. The ***oneforthpix()*** function of TML6 also interpolates values for these 2/4,0/4 and 0/4,2/4 sub-pixels using the values of the unit pixels (**purple**), which are located in rows and columns, respectively. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 538-592).

165. In particular, the ***oneforthpix()*** function interpolates the values for the 2/4,0/4 sub-pixels (**blue**) (highlighted below) with values of the unit pixels (**purple**) that are stored in the data structure ***imgY*** and that are in the same row as the highlighted sub-pixel.



(2/4,0/4)
K=2, L=0

As further illustrated below, these highlighted 2/4,0/4 sub-pixels (**blue**) are interpolated in the first step of the ***oneforthpix()*** function by using a weighted sum of the values of the three adjacent circled unit pixels (**purple**) to the left of the highlighted sub-pixel and the three adjacent circled unit pixels (**purple**) to the right of the highlighted sub-pixel.



```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                    img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                    img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 563-575, 588).

In this weighted sum, the weights assigned to each unit pixel (**purple**) value are stored in a data structure ***ONE_FOURTH_TAP*** (defined in the file ***image.h*** in the ***lencod*** folder).

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                    img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                    img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

```
28   /* TAPs used in the oneforthpix()routine */
29    const int ONE_FOURTH_TAP[3][2] =
30     {
31       {20,20},
32       {-5,-4},
33       { 1, 0},
34     };
35
```

TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 568-570), 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); *id.,* 57 (/tml6pre.zip/lencod/src/image.c, l. 572) (as shown at line 572, the weighted sum is multiplied by 32, *i.e.,* each of the weights from the data structure ***ONE_FOURTH_TAP*** in the weighted sum is multiplied by a scaling factor, 32). A value including this weighted sum is stored in the data structure ***img4Y_tmp*** and then, after being rounded and brought into the dynamic range, is stored in the data structure ***mref***.

```
572                 img4Y_tmp[2*j][i2+1]=is*32;              /* 1/2 pix pos */
```

```
588                 mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                    img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 572, 588).

166. The ’273 patent, in conjunction with Figure 13(b), explains that TML6 interpolates these ½ sub-pixels in this manner.

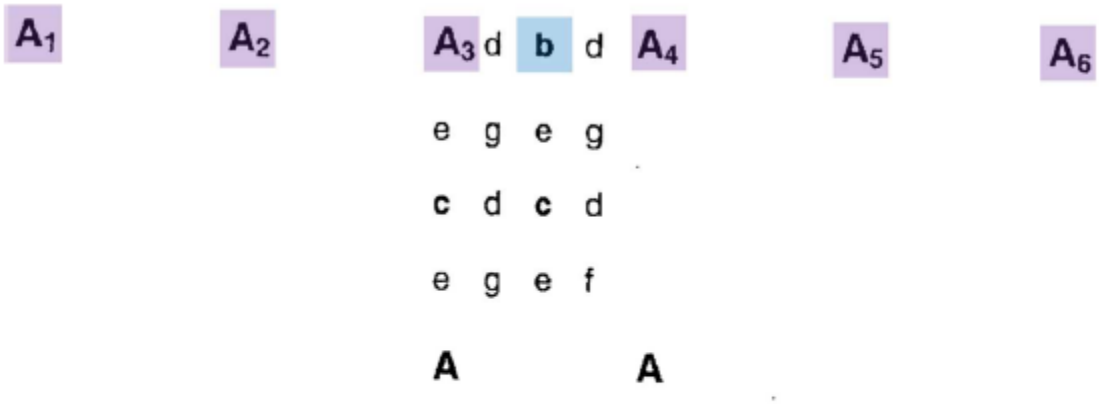
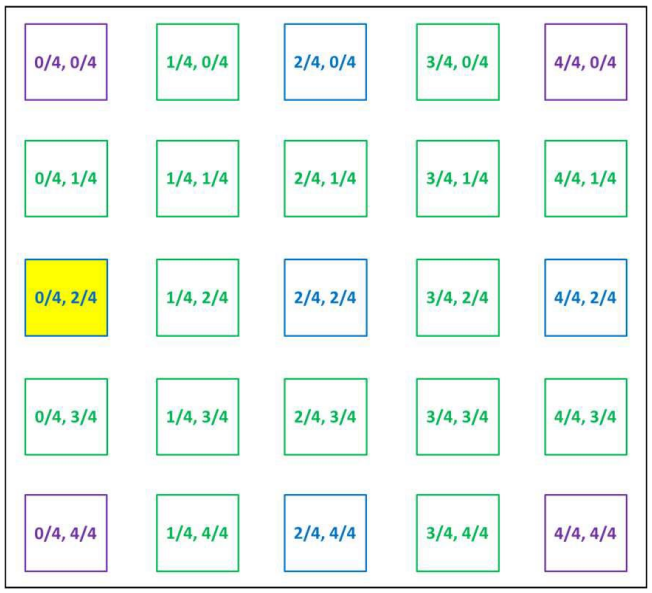


Figure13(b)

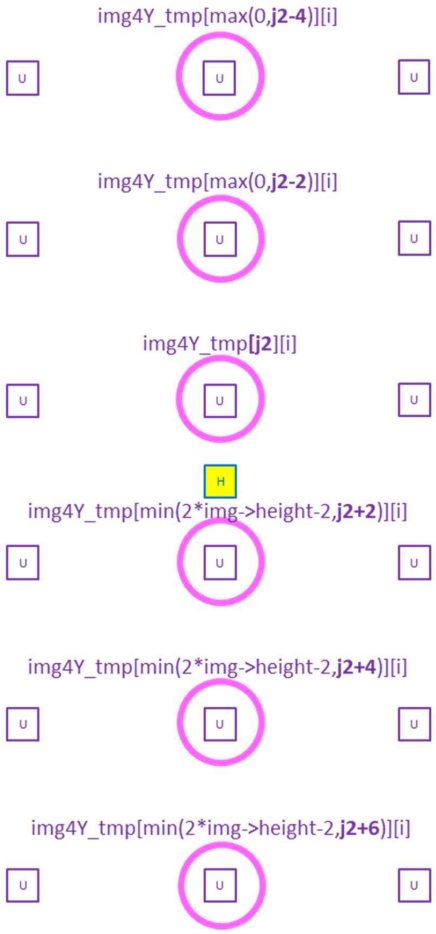
The '273 patent notes that the value for the 2/4,0/4 sub-pixel "b" (**blue**) above is interpolated "based upon the values of the 6 pixels (A_1 to A_6) [**purple**] situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13b, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$." EX-1001, Figures 13(a)-13(b), 11:35-45.

167. The ***oneforthpix()*** function similarly interpolates the values for the 0/4,2/4 sub-pixels (**blue**) (highlighted below) by using unit pixels (**purple**) values that are stored in the data structure ***imgY*** and that are in the same column as the highlighted sub-pixels.



(0/4,2/4)
K=0, L=2

As illustrated below, these highlighted 0/4,2/4 sub-pixels (**blue**) are interpolated in the second step of the ***oneforthpix()*** function by using a weighted sum of the values of the three adjacent circled unit pixels (**purple**) above the highlighted sub-pixel and the three adjacent circled unit pixels (**purple**) below the highlighted sub-pixel.




```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).

168. The weights in this weighted sum that are assigned to these values of the unit pixels (**purple**) are stored in the data structure ***ONE_FOURTH_TAP***, which I discuss above.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

```
28    /* TAPs used in the oneforthpix()routine */
29     const int ONE_FOURTH_TAP[3][2] =
30       {
31         {20,20},
32         {-5,-4},
33         { 1, 0},
34       };
35
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592), 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34). A value including this weighted sum

is stored in the data structure ***img4Y_tmp***. Then, it is rounded and brought into the dynamic range, the value is stored in the data structure ***mref***.

```
587                 img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588                 mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                    img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589                 mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                    512)/1024))); /* 1/2 pix */
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).

169. The ’273 patent, with reference to Figure 13(c), explains that TML6 interpolates these ½ sub-pixels in the same manner.

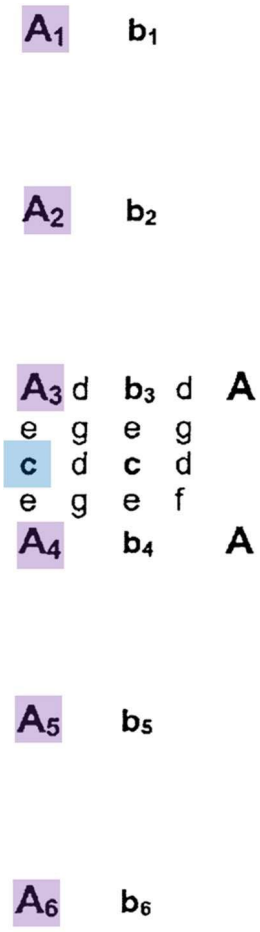


Figure 13(c)

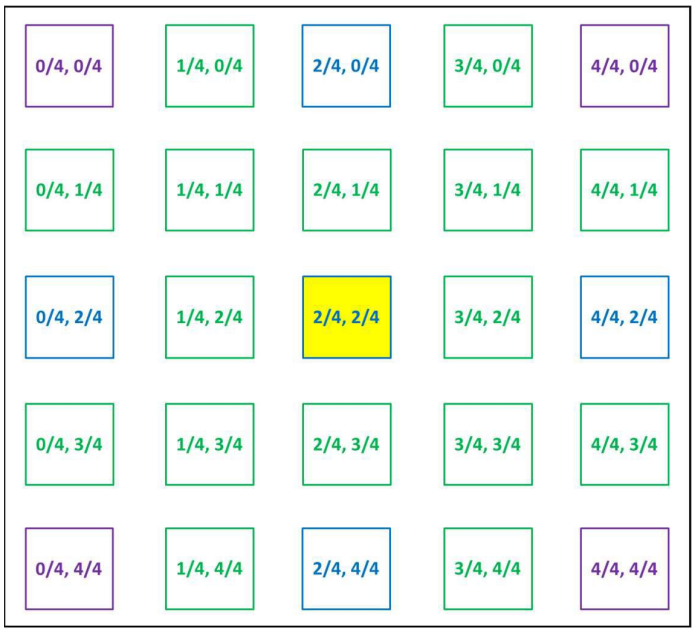
The ’273 patent notes that value for the 0/4,2/4 sub-pixel “c” (**blue**) (above) is interpolated “based upon the values of the 6 pixels (A_1 to A_6) [**purple**] situated in a column at unit horizontal locations and unit vertical locations symmetrically about c [**blue**], according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$.” EX-1001, Figures 13(a)-13(c), 11:46-11:63 (“Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13c, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and

half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to $(c+512)/1024$. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").

(v) Element 1[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions"

170. TML6 combined with Fandrianto renders obvious Element 1[c]. As shown below, TML6's ***oneforthpix()*** function interpolates values for 2/4,2/4 sub-pixels (**blue**) (highlighted below) having co-ordinates ($K/2^N$, $L/2^N$, where K=L=N=2) with even values of K and L.[16]

[16] As I mentioned in my analysis of Element 1[b] above, the combined teachings of TML6 and Fandrianto disclose interpolating these sub-pixels in other rectangular bounded regions. Element 1[b], *supra*. If claim requires interpolating *multiple* such sub-pixels in one rectangular bounded region with even values of K and L (as claimed), the scope of the claim is unclear, since there is only *one* such sub-pixel when N=2. However, TML6 or the combination of TML6 and Fandrianto show



(2/4,2/4)
K=2, L=2

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592). As I discuss with respect to Element 1[a], these 2/4,2/4 sub-pixels are "situated at co-ordinates ½,½" and are used in the interpolation of the values for the sub-pixels having co-ordinates ¼,¼, ¼,¾, ¾,¼, and/or ¾,¾ with odd values of K and L. Element 1[a], *supra*.

171. As illustrated below, the highlighted 2/4,2/4 sub-pixels (**blue**) are interpolated in the second step of the ***oneforthpix()*** function using a weighted sum

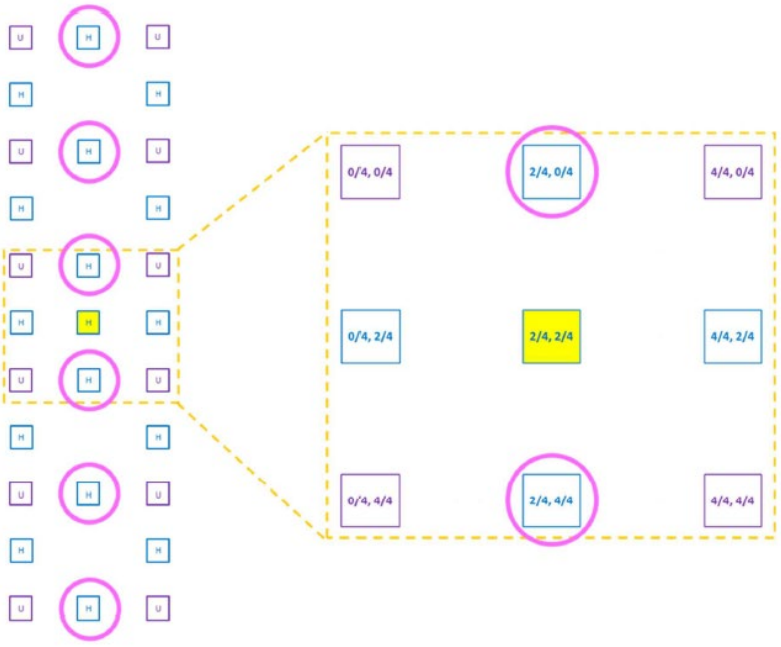
---

this claimed feature to the same extent that the '273 patent discloses this claimed feature.

(hereinafter referred to as the "first weighted sum") of: (1) the values of the circled 2/4,0/4 sub-pixel and 2/4,4/4 sub-pixel (**blue**) (in the **orange** box) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=0 or 4, N=2) with K equal to an even value and L equal to zero,[17] and (2) the values of the circled 2/4,0/4 sub-pixel and 2/4,4/4

[17] Under one claim interpretation, Element 1[c] possibly may require interpolating sub-pixels (with even values of both K and L) using a weighted sum of: (1) *at least two sub-pixels* having co-ordinates with K equal to an even value and *L equal to zero* (both within the same bounded region), and (2) corresponding sub-pixels in adjacent rectangular bounded regions. However, the '273 patent discloses interpolating the 2/4,2/4 sub-pixel c using a weighted sum of: (1) *only one sub-pixel* having co-ordinates with K equal to an even value (*i.e.,* 2) and *L equal to zero* (*i.e.,* the 2/4,0/4 sub-pixel b_3) *and* a second 2/4,4/4 sub-pixel b_4 having co-ordinates with K equal to an even value (*i.e.,* 2) and L equal to *four* (and *not* zero), and (2) corresponding sub-pixels b_1 and b_2 and sub-pixels b_5 and b_6 in adjacent rectangular bounded regions. EX-1001, Figures 14(a)-14(c), 45:2-43 ("$c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6+512)$"). TML6 or the combination of TML6 and Fandrianto discloses interpolating these sub-pixels in the same way as described below and as confirmed by the '273 patent. *Id.*, Figure 13(c), 11:46-63 ("$c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$").

sub-pixel (in the **green** boxes) having corresponding co-ordinates in immediately adjacent rectangular bounded regions.



First Weighted Sum

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592). A value having this first weighted sum is stored in the data structure ***img4Y_tmp***. Then, after it is rounded and brought into the dynamic range, the value is stored in the data structure ***mref***.

```
587                 img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588                 mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                    img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589                 mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                    512)/1024))); /* 1/2 pix */
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).

172. The ’273 patent, with reference to Figure 13(c), notes that TML6 interpolates the value for the 2/4,2/4 sub-pixels c (**blue**) in the same way.

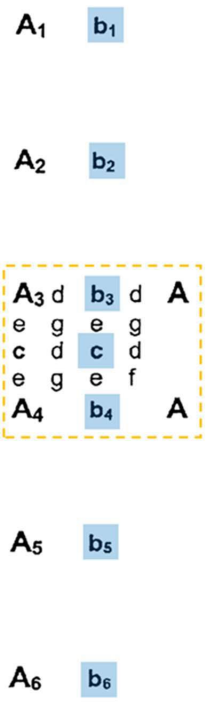


Figure 13(c)

EX-1001, Figure 13(c), 11:46-11:63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13c, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to c=(c+16)/32. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half Vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator/denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.").

173. Fandrianto also explains that the values for the highlighted 2/4,2/4 sub-pixel C (**blue**, below) are interpolated *either* by (1) averaging the circled 2/4,0/4 sub-pixel A above and the 2/4,4/4 sub-pixel A below the highlighted sub-pixel ("vertical interpolation") *or* (2) averaging the circled 0/4,2/4 sub-pixel B to the left and the 4/4,2/4 sub-pixel B to the right of the highlighted 2/4,2/4 sub-pixel C ("horizontal interpolation").

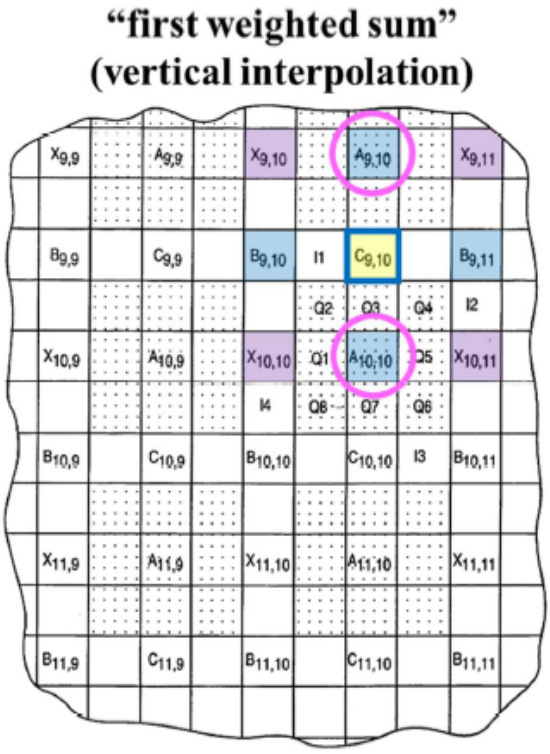


FIG. 5

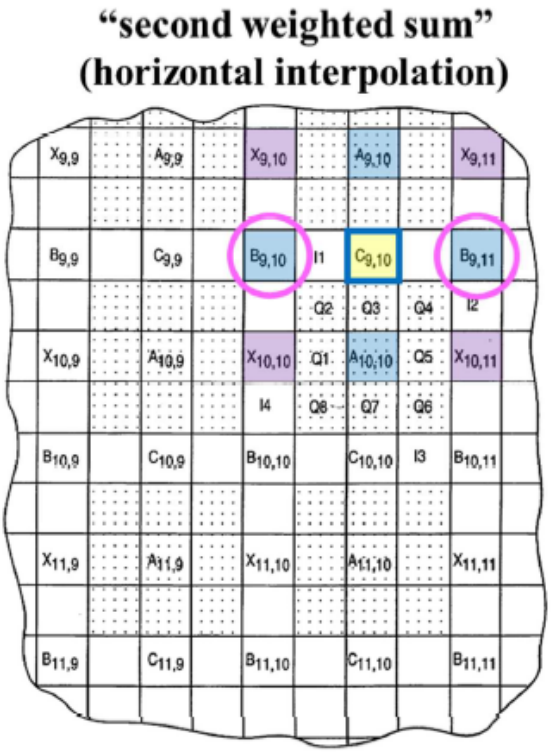


FIG. 5

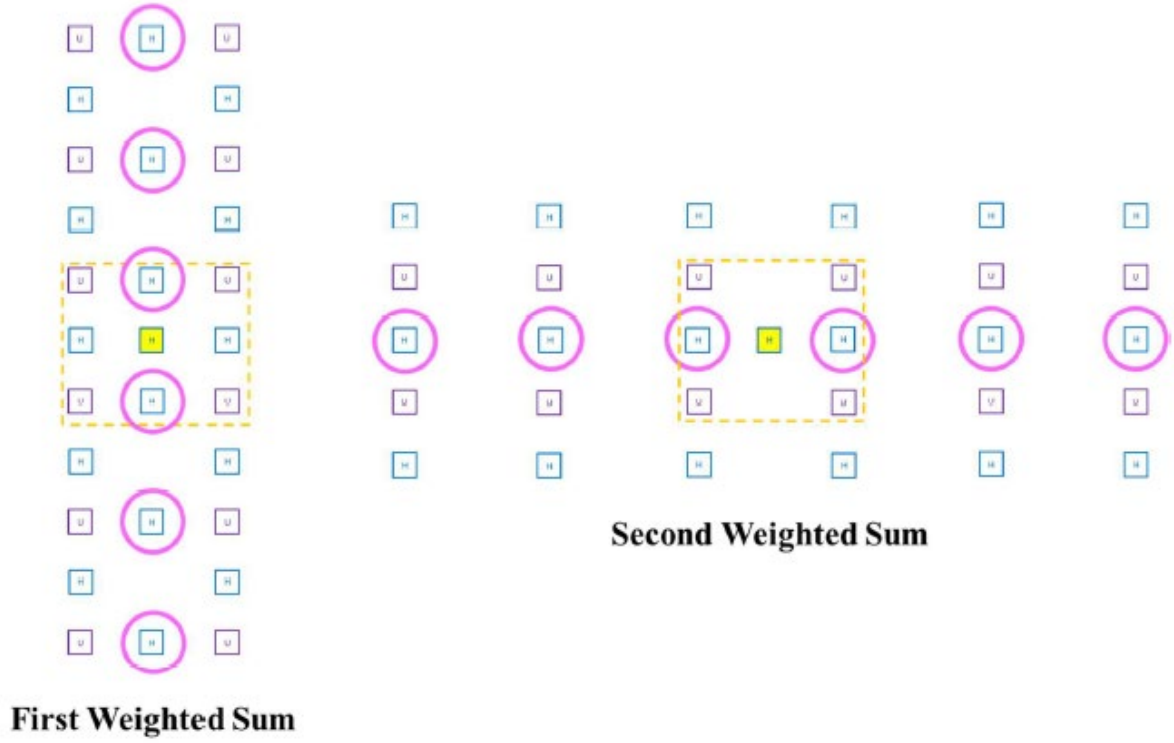
Fandrianto, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 6:4-12 ("Pixels $A_{r,c}$ (r=0, . . . ,17; c=0, . . . , 16) of the A block 72 are horizontally interpolated at half-pixel locations, from the X block 70. Pixels $B_{r,c}$ (r=0, . . . , 16; c=0, . . . , 17) of the B block 74 are vertically interpolated at half-pixel locations, from the X block 70. Pixels $C_{r,c}$ (r=0, . . . , 16; c=0, . . . , 16) are pixels interpolated at half-pixel locations, preferably vertically from the A matrix

72 or horizontally from the B matrix 74, but may also be interpolated diagonally from the X matrix 70."), 6:49-6:54 ("The C block 76 is formed by progressive interpolation of preferably ***either*** the A block 72 vertically, or the B block 74 horizontally."), 6:18-48, 8:36-10:50. Accordingly, as set forth in Fandrianto, the highlighted 2/4,2/4 sub-pixels are interpolated by using either the "first weighted sum" of the values of sub-pixels having co-ordinates with K equal to an even value 2 and L equal to zero ("A block") *or* another weighted sum ("second weighted sum") of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value ("B block"),[18] with respective weights in each weighted sum equal to "1." *Id.*

174. It would have been obvious for a POSITA to modify the teachings of TML6 based on the teachings of Fandrianto, for analogous reasons as discussed above in Section IX.A.1, *supra*, such that the ***oneforthpix()*** function interpolates the 2/4,2/4 sub-pixels using a predetermined choice of either the first weighted sum of TML6 (summing six values of sub-pixels having co-ordinates with K equal to

[18] As I have previously explained above in footnotes 16-17, the combined teachings of TML6 and Fandrianto disclose the claimed feature in the same manner that the embodiments of '273 patent discloses this feature. EX-1001, Figures 14(a)-14(c), 45:25-43 ("$c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6+512)/1024$").

an even value (two) and L equal to zero) or the second weighted sum of TML6 as modified by Fandrianto (also summing six values horizontally like Fandrianto of sub-pixels having co-ordinates with K equal to zero and L equal to an even value (two)).



Put another way, Fandrianto teaches a vertical or a horizontal weighted sum of two values, and TML6 teaches a vertical sum of six values, so the combination teaches a vertical or horizontal weighted sum, each of six values. Fandrianto expressly teaches that a finite number of interpolation techniques could have been used to interpolate the 2/4,2/4 sub-pixels, including vertical interpolation using a first weighted sum (as also used in TML6), and horizontal interpolation using a second

weighted sum. Fandrianto, 6:4-12 ("Pixels $A_{r,c}$ (r=0, . . . ,17; c=0, . . . ,16) of the A block 72 are horizontally interpolated at half-pixel locations, from the X block 70. Pixels $B_{r,c}$ (r=0, . . . , 16; c=0, . . . , 17) of the B block 74 are vertically interpolated at half-pixel locations, from the X block 70. Pixels $C_{r,c}$ (r=0, . . . , 16; c=0, . . . , 16) are pixels interpolated at half-pixel locations, preferably vertically from the A matrix 72 or horizontally from the B matrix 74, but may also be interpolated diagonally from the X matrix 70."), 6:49-7:2 ("The C block 76 is formed by progressive interpolation of preferably either the A block 72 vertically, or the B block 74 horizontally."). It would, therefore, would be obvious for a POSITA to try such finite options, including using a predetermined choice between the first and second weighted sums, to interpolate the 2/4,2/4 sub-pixels in TML6, and both using six sub-pixels as is consistent with TML6. Revising the TML6 code to implement this modification would also be straightforward, and a POSITA would reasonably expect this modification to yield the predictable result of interpolating the 2/4,2/4 sub-pixels vertically or horizontally. The '273 patent recognizes that interpolating the 2/4,2/4 sub-pixels "c," as illustrated in Figure 14(a) below, using either a vertical weighted sum of sub-pixels "b" or a horizontal weighted sum of sub-pixels "b" results in the same outcome.

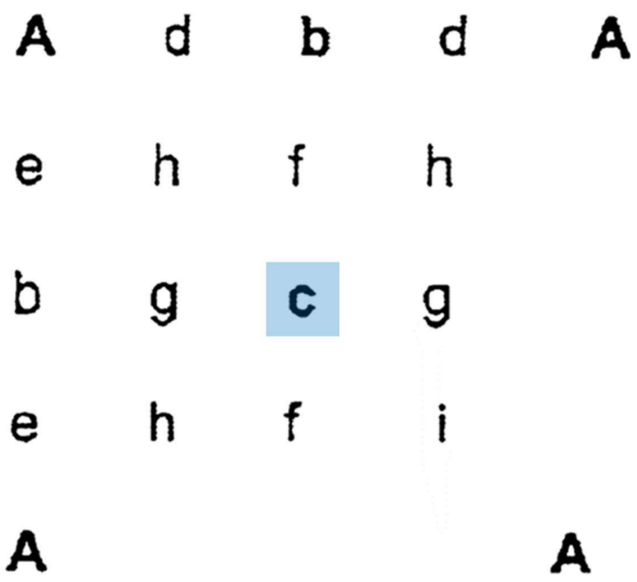


Fig. 14(a)

EX-1001, Figure 14(a), 42:18-52 ("It should be noted that the use of intermediate values b in the horizontal direction leads to the same result as using intermediate values b in the vertical direction."), 45:25-43 (same).

175. Accordingly, the combined teachings of TML6 and Fandrianto disclose or render obvious a predetermined choice between the weighted sums of values, as claimed.

b) Dependent Claim 4: "A method according to claim 1, comprising causing the apparatus to use a first and a second weight in the weighted average chosen to interpolate the sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, wherein when the chosen weighted average is that involving the values of a pair of diagonally-opposed sub-pixels

having co-ordinates with even values of K and L, causing the apparatus to select the relative magnitudes of the first and second weights to be inversely proportional to the respective straight-line diagonal distances of the sub-pixels having co-ordinates with even values of K and L to the sub-pixel having co-ordinates with odd values of K and L whose value is being interpolated"

176. Claim 4 would have been obvious in view of TML6 in combination with Fandrianto. As I discussed earlier with respect to Element 1[a], and as illustrated below, Fandrianto discloses interpolating the value for the highlighted ¼,¾ sub-pixel Q2 (**green**) having co-ordinates with odd values of K ($K/2^N$, K=1, N=2) and L ($L/2^N$, L=3, N=2) according to a predetermined choice of two weighted averages.

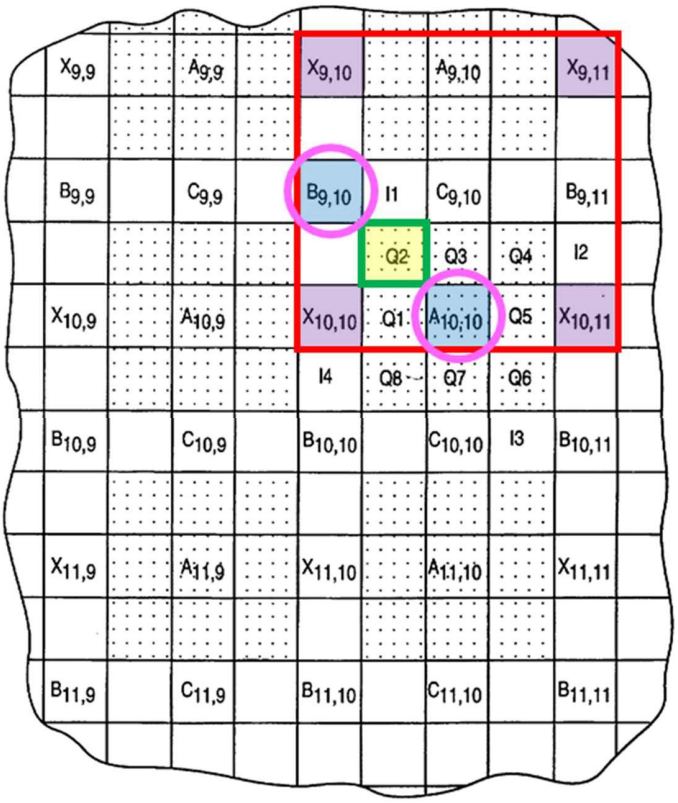


FIG. 5

Fandrianto, Figures 2, 5, 6:18-7:2, 8:36-10:58; Element 1[a] *supra*.

177. When the chosen weighted average involves the values of the pair of diagonally-opposed 0/4,2/4 sub-pixel B and 2/4,4/4 sub-pixel A (**blue** and circled above), which have co-ordinates with even values of K ($K/2^N$, K=0 or 2, N=2) and L ($L/2^N$, L=2 or 4, N=2), the magnitudes of the first and second weights (both equal to "1") in the weighted average of the sub-pixels A and B (**blue**) are inversely proportional to the respective straight-line diagonal distances (shown by the **orange** arrows below) of the circled sub-pixels A and B (**blue**) (which have co-

ordinates with even values of both K and L) to the interpolated highlighted sub-pixel Q2 (**green**) (which have co-ordinates with odd values of both K and L).

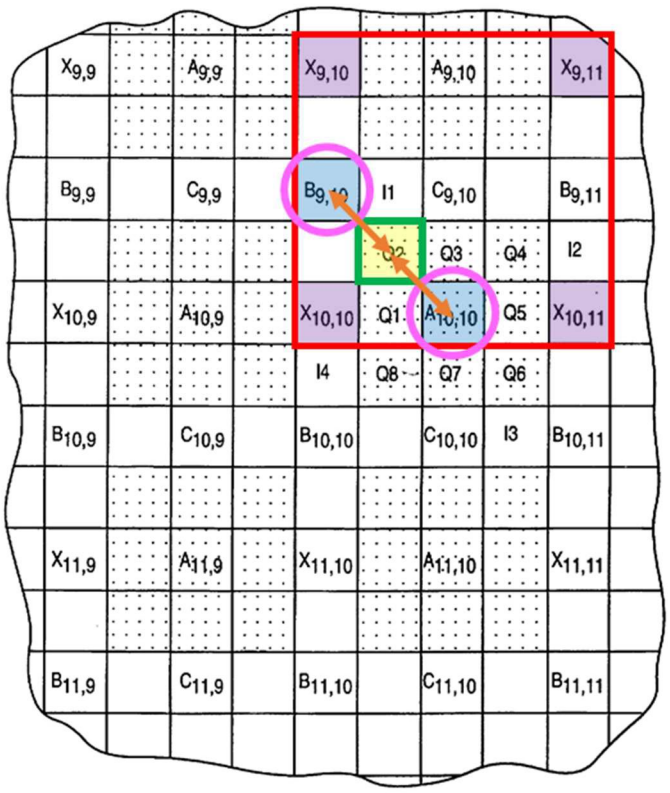


FIG. 5

Fandrianto, Figure 5; Element 1[a], *supra*. Specifically, as shown in the figure above, these sub-pixels A and B (**blue**) are the same distance from the sub-pixel Q2 (**green**). As a result, the magnitudes or values of their weights ("1") are inversely proportional to the distances (depicted by **orange** arrows above) of the sub-pixels A and B from the sub-pixel Q2 (**green**).

178. Insofar as the claim is interpreted to require affirmatively "causing the apparatus to select" the relative magnitudes of the first and second weights of the

weighted average to be inversely proportional to the respective distances of the sub-pixels A and B (**blue**) to the sub-pixel Q2 (**green**), these features would have been obvious. As a general principle, if the sub-pixels A and B are ***closer*** to the sub-pixel Q2, the ***more*** likely their values are similar to the value of the sub-pixel Q2. Therefore, the values of the "***closer***" sub-pixels A and B (with ***smaller*** distances to the sub-pixel Q2) should have ***larger*** weights. On the other hand, if the sub-pixels A and B are ***farther*** from the sub-pixel Q2, the ***less*** likely their values are similar to the value of the sub-pixel Q2. Accordingly, the values of the "***farther***" sub-pixels A and B (with ***larger*** distances to the subpixel Q2) should have ***smaller*** weights. A POSITA would therefore have readily known to select the relative magnitudes of the weights of the sub-pixels A and B in the weighted average to be inversely proportional to the respective distances of the sub-pixels A and B from the sub-pixel Q2 being interpolated.

179. Further, a POSITA would have known to use linear interpolation when interpolating between two points, which is acknowledged in the '273 patent. EX-1001, 10:45-62, Table 1, Table 2; *see also*, VCEG-M28 (describing 2-tap filters); Britannica, 356 (interpolation, generally). Inherent to linear interpolation is the property that the ratio of the weights are inversely proportional to the respective distances of known points to the interpolated point. Britannica, 356 (linear interpolation of a value "y" corresponding to location "x" using "two points

(x_0,y_0) and (x_1,y_1),” with x_1>x> x_0, is based on “the equation y=…y_0+b(x-x_0),” and rearranging this equation manifests that the weights multiplying y_0 and y_1 are respectively (x_1-x)/(x_1-x_0) and (x-x_0)/(x_1-x_0), such that the ratio between weights for y_0 and y_1 is (x_1-x)/(x-x_0), which is the inverse between their respective distances).

180. Thus, TML6 and Fandrianto in combination disclose interpolating values of TML6’s ¼,¼; ¼,¾; ¾,¼; and ¾ ¾ sub-pixels having co-ordinates with odd values of both K and L ($K/2^N$, K=1 or 3, N=2) and L ($L/2^N$, L=1 or 3, N=2), as set forth in claim 4. Element 1[a], *supra*.

c) Dependent Claim 5: “A method according to claim 4, comprising causing the apparatus to use first and second weights with equal values when the sub-pixels having co-ordinates with even values of K and L are equidistant from the sub-pixel having co-ordinates with odd values of K and L whose value is being interpolated”

181. Claim 5 would have been obvious over TML6 in combination with Fandrianto. As I outlined above for claim 4 and demonstrate below, the first and second weights equal “1” when the sub-pixels A and B (**blue**) (which have co-ordinates with even values of both K and L) are the same distance from the sub-pixel Q2 (**green**) (which has co-ordinates with odd values of both K and L) that is being interpolated.

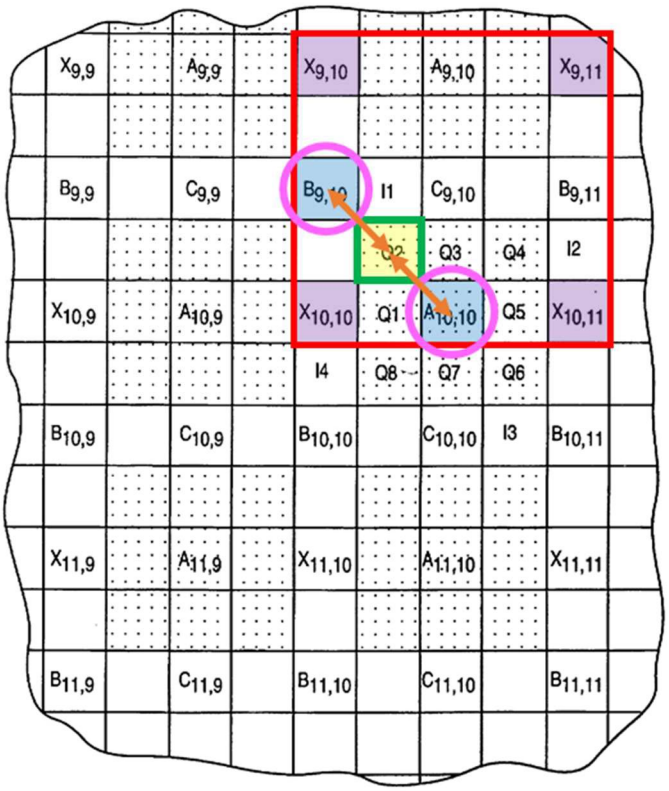


FIG. 5

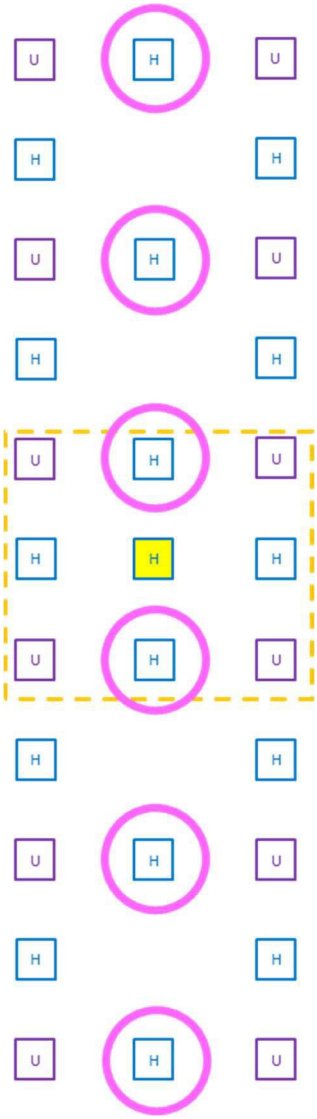
Fandrianto, Figure 5; Claim 4 *supra*.

d) Dependent Claim 6: "A method according to claim 1, comprising causing the apparatus to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions when the value of a sub-pixel having

co-ordinates with K equal to an even value and L equal to an odd value is also required"

182. As I discuss above, the combination of TML6 and Fandrianto discloses or renders obvious claim 1. This combination also discloses or renders obvious the additional features of claim 6.

183. As noted above, Element 1[c] notes that values for the sub-pixels (*e.g.*, 2/4,2/4 sub-pixels), which have co-ordinates with even values of K and L, are interpolated using a ***predetermined choice of*** a first weighted sum of certain sub-pixel values ***and*** a second weighted sum of other sub-pixel values. Element 1[c], *supra*. Claim 6 repeats a lot of Element 1[c] word for word and further recites that the sub-pixel values (*e.g.*, for the 2/4,2/4 sub-pixels) are interpolated using the first weighted sum "when the value of a sub-pixel having co-ordinates with K equal to an even value and L equal to an odd value is also required." As I explain above for Element 1[c], TML6 explains that these 2/4,2/4 sub-pixels (highlighted below) are interpolated using the first weighted sum of the sub-pixel values (circled below).



Element 1[c], *supra*.

184. Fandrianto further describes interpolating the value of the highlighted 2/4,1/4 sub-pixel "Q7" (**green**), which has co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=1, N=2) with K equal to an even value and L equal to an odd value, and corresponding Q7 sub-pixels in other rectangular bounded regions, when these pixels are required for motion estimation.

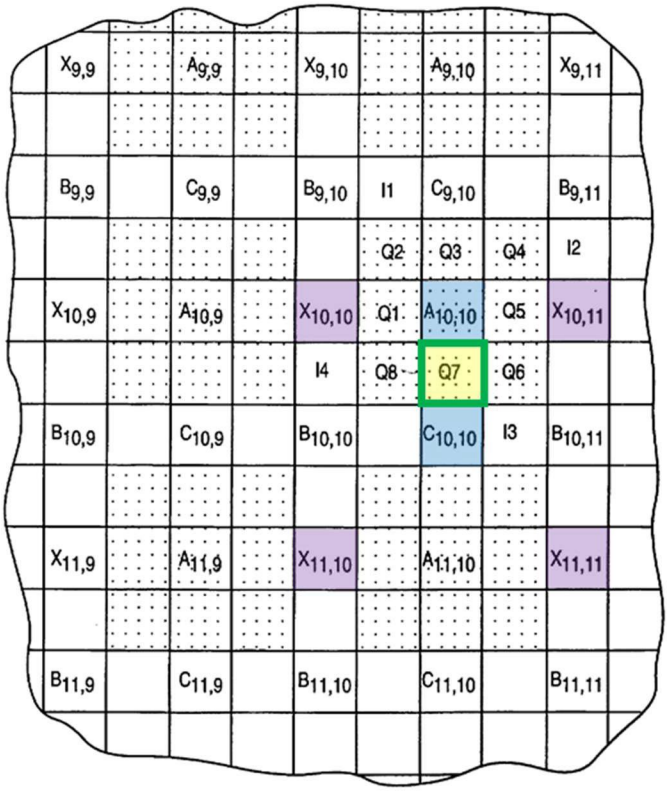
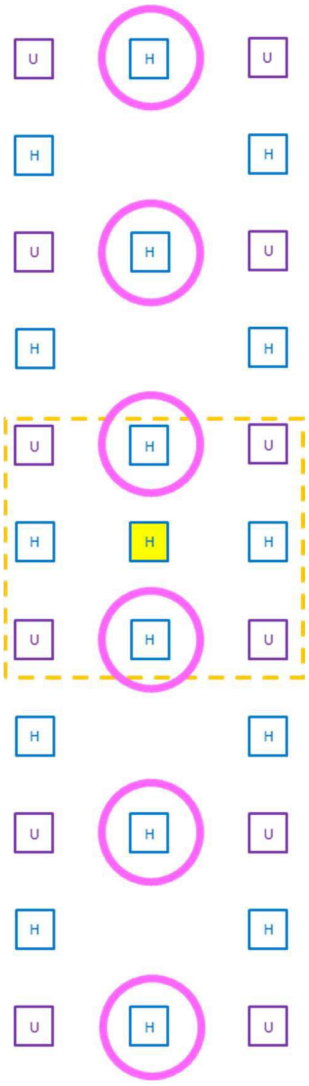


FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed…."), 8:29-35 ("First, the best match block is moved from A block 72 of the DPCM memory 130 into the D block 82 of the DP memory 124. In conformance with FIG. 5, however, the pixels of the best match block now stored in the D block 82 are referred to as $A_{r,c}$. This frees up the A block 72 to hold the current quarter pixel interpolated block, the pixels of which for convenience are referred to as $Q_{r,c}$."), 10:23-33 ("$Q7_{r,c}=(C_{r,c}+A_{r,c})/2$"); *see also id*., 8:36-57 ("Q1"), 9:33-43 ("Q3"). As illustrated in the figure above, the highlighted Q7 sub-pixel (**green**) is interpolated by using the value of the 2/4,2/4 sub-pixel $C_{10,10}$ (**blue**), and thus, this sub-pixel $C_{10,10}$ would

be interpolated when interpolating the sub-pixel Q7 is required. *Id*., Figure 5, 5:26-7:19, 8:29-35, 10:23-33 ("$Q7_{r,c}=(C_{r,c}+A_{r,c})/2$"); *see also id.*, 8:36-57, 9:33-43. Therefore, the combination of TML6 and Fandrianto's interpolation method teach interpolating 2/4,2/4 sub-pixels using the first weighted sum (shown below) when the 2/4,1/4 sub-pixel is also required for motion estimation.



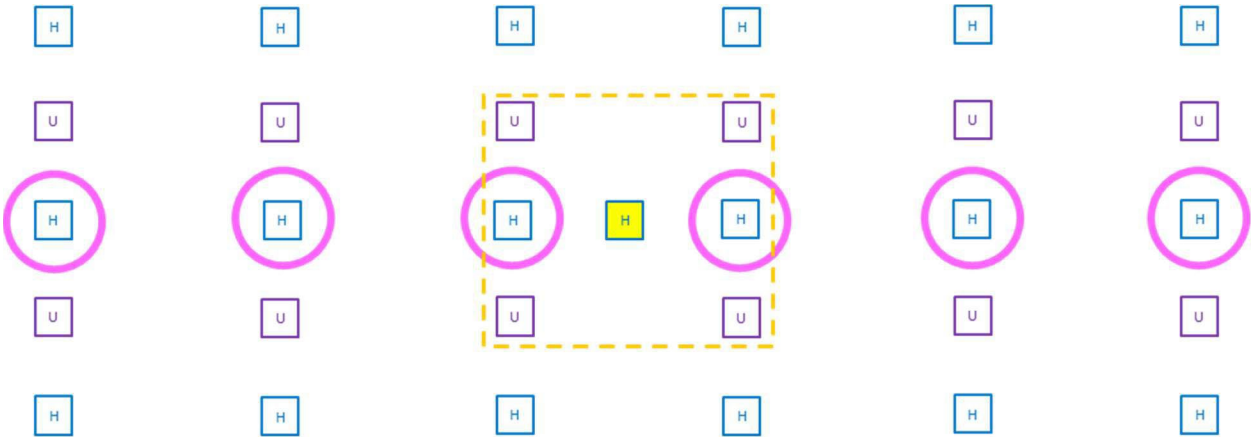
Id.; Section IX.A.1, Element 1[c], *supra*.

e) Dependent Claim 7: "A method according to claim 1, comprising causing the apparatus to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions when the value of a sub-pixel having co-ordinates with K equal to an odd value and L equal to an even value is also required"

185. As I discuss above, the combination of TML6 and Fandrianto discloses or renders obvious claim 1. This combination also discloses or renders obvious the additional features of claim 7.

186. As noted above, Element 1[c] recites that values for sub-pixels (*e.g.*, 2/4,2/4 sub-pixels) with co-ordinates having even values of K and L are interpolated using a ***predetermined choice of*** a first weighted sum of certain sub-pixel values ***and*** a second weighted sum of other sub-pixel values. Element 1[c], *supra*. Like claim 6, claim 7 repeats much of Element 1[c] word for word and further requires interpolating the sub-pixel values (*e.g.*, for the 2/4,2/4 sub-pixels) by using the second weighted sum "when the value of a sub-pixel having co-ordinates with K equal to an odd value and L equal to an even value is also required."

187. As I explain above for Element 1[c] and as shown below, the combination of TML6 and Fandrianto further show interpolating these highlighted 2/4,2/4 sub-pixels using the second weighted sum of the circled sub-pixels.



Element 1[c], *supra*.

188. Fandrianto also interpolates the value of the highlighted 1/4,2/4 sub-pixel "I1" (**green**) having co-ordinates ($K/2^N$, $L/2^N$, where K=1, L=2, N=2) with K equal to an odd value and L equal to an even value, and corresponding sub-pixels in other rectangular bounded regions, when required for motion estimation. Also, I note that the sub-pixels I1, as well as the other sub-pixels I2 to I4 are, in fact, interpolated sub-pixels since they are located at fractional pixel locations. These sub-pixels I1 to I4, like the sub-pixels Q1 to Q8, are mathematically interpolated based on values of other pixels and/or other sub-pixels. *See*, *e.g.*, Fandrianto, 8:15-10:58.

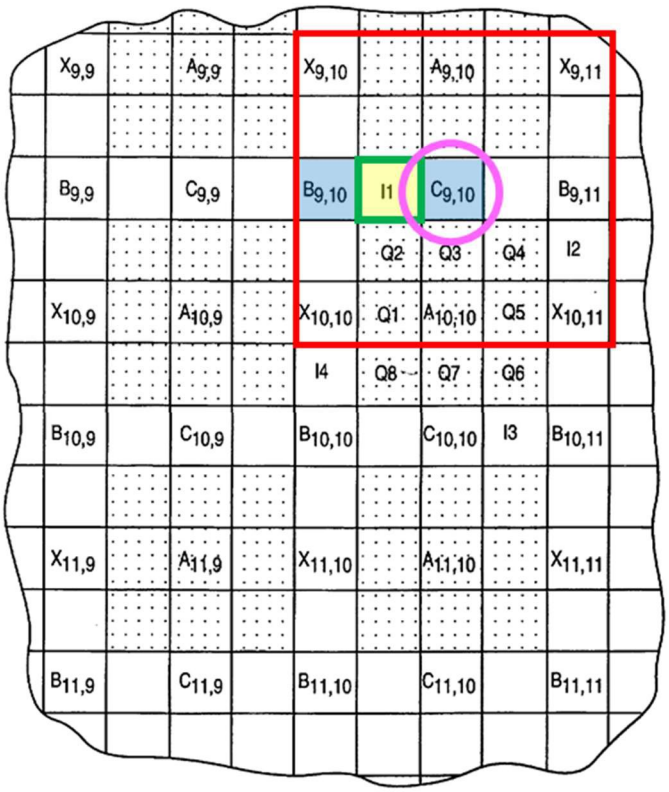
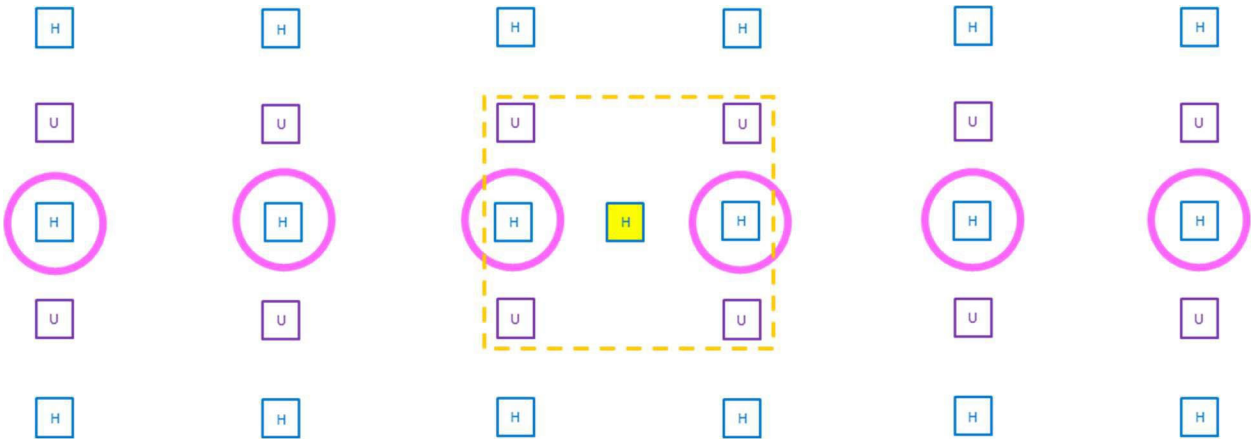


FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed…."), 8:29-35 ("First, the best match block is moved from A block 72 of the DPCM memory 130 into the D block 82 of the DP memory 124. In conformance with FIG. 5, however, the pixels of the best match block now stored in the D block 82 are referred to as $A_{r,c}$. This frees up the A block 72 to hold the current quarter pixel interpolated block, the pixels of which for convenience are referred to as $Q_{r,c}$."), 8:58-9:32 ("$I1_{r,c}=(B_{r-1,c}+C_{r-1,c})/2$") 10:6-22 ("I3"). As illustrated in the figure above, interpolating the highlighted I1 sub-pixel (**green**) uses the value of the 2/4,2/4 sub-pixel $C_{9,10}$ (**blue**).

Accordingly, this sub-pixel $C_{9,10}$ would be interpolated when interpolating the sub-pixel I1 is required. *Id*., Figure 5, 5:26-7:19, 8:29-35, 8:58-9:32 ("$I1_{r,c}=(B_{r-1,c}+C_{r-1,c})/2$"); *see also id*., 10:6-22. Therefore, the combination of TML6 and Fandrianto's sub-pixel interpolation techniques teach interpolating 2/4,2/4 sub-pixels using the second weighted sum (shown below) when the 1/4,2/4 sub-pixel is also required.

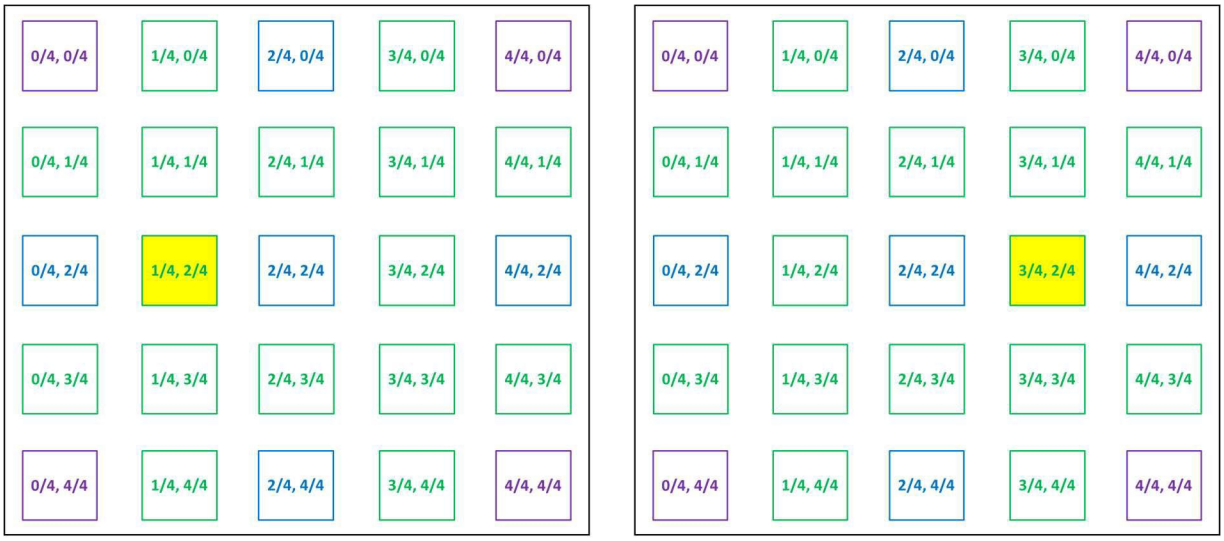


Id.; Section IX.A.1, Element 1[c], *supra*.

f) Dependent Claim 8: "A method according to claim 1, comprising causing the apparatus to interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an odd value O and L equal to an even value E, not including zero or 2^N, by taking an average of the value of a first sub-pixel having co-ordinates with K equal to the even value O-1 and L equal to E and a second sub-pixel having co-ordinates with K equal to the even value O+1 and L equal to E"

189. As I discuss above, the combination of TML6 and Fandrianto discloses or renders obvious claim 1. This combination also discloses or renders

obvious the additional features of claim 8. For example, as illustrated below, the combination of TML6 and Fandrianto discloses interpolating sub-pixels (*e.g.*, the 1/4,2/4 and 3/4,2/4 sub-pixels) having co-ordinates ($K/2^N$, $L/2^N$, where K=1 or 3, L=2, N=2) with K equal to an odd value O (*e.g.*, 1 or 3), and L equal to an even value E (*e.g.*, 2), which is not zero or 2^N (*i.e.*, 4).



(1/4,2/4)
K=O=1, L=E=2

(3/4,2/4)
K=O=3, L=E=2

190. As further illustrated in the left portion of the illustration below, in Fandrianto, the highlighted 1/4,2/4 sub-pixel "I1" (**green**) is interpolated by averaging the values of a first 0/4,2/4 sub-pixel $B_{9,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=2, N=2) with K equal to the even value O-1 (*e.g.*, 0 (1-1)) and L equal to E (*e.g.*, 2) and a second 2/4,2/4 sub-pixel $C_{9,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=2, N=2) with K equal to the even value O+1 (*e.g.*, 2 (1+1)) and L equal to E (*e.g.*, 2). In a similar

fashion, as also illustrated below (right), in Fandrianto, the highlighted 3/4,2/4 sub-pixel "I3" (**green**) is interpolated by averaging the values of a first 2/4,2/4 sub-pixel $C_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=2, N=2) with K equal to the even value O-1 (*e.g.*, 2 (3-1) and L equal to E (*e.g.*, 2) and a second 4/4,2/4 sub-pixel $B_{10,11}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=2, N=2) with K equal to the even value O+1 (*e.g.*, 4 (3+1)) and L equal to E (*e.g.*, 2).

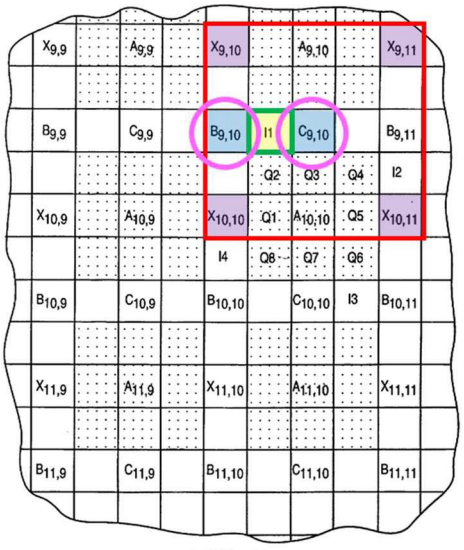


FIG. 5

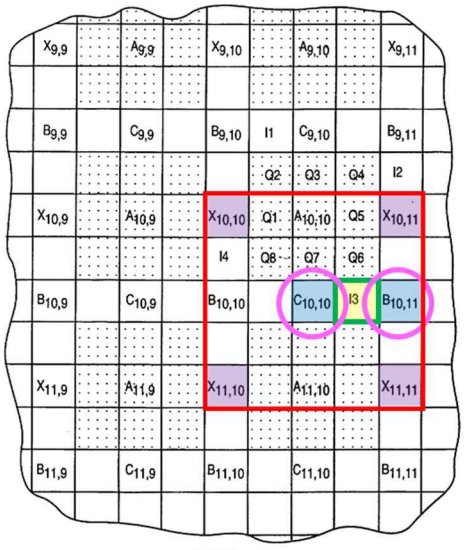


FIG. 5

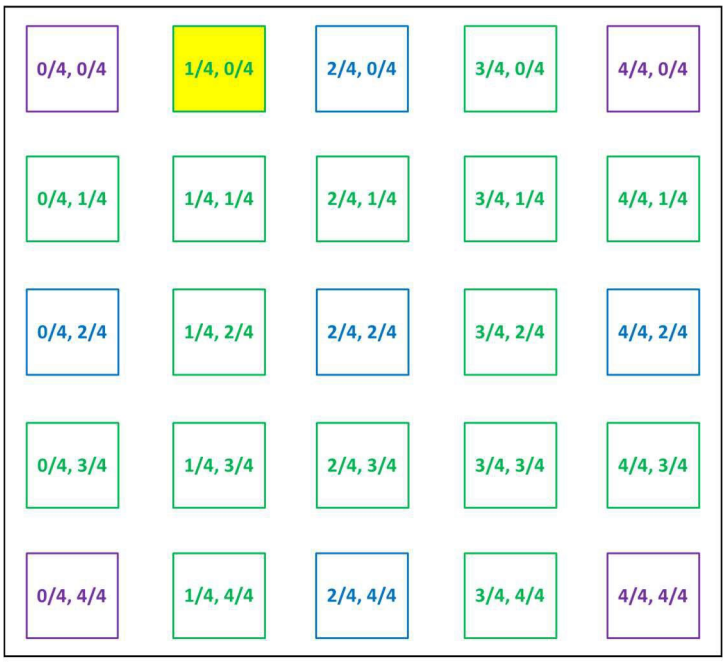
Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed…."), 8:29-57, 8:58-9:32 ("$I1_{r,c}=(B_{r-1,c}+C_{r-1,c})/2$"), 10:6-22 ($I3_{r,c}=(B_{r,c+1}+C_{r,c})/2$").

191. As I discuss above in Section IX.A.1 (motivation to combine), and for Element 1[a] and 1[c], it would have been obvious to substitute Fandrianto's interpolation technique for that of TML6. This combination interpolates the sub-

pixels required by claim 8 using values for the recited first and second sub-pixels from the ***mref*** data structure of TML6. Section IX.A.1, Elements 1[a], and 1[c], *supra*.

g) Dependent Claim 9: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 1 and L equal to zero, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to zero, and a sub-pixel having co-ordinates with K equal to 2 and L equal to zero"

192. As I discuss above, the combination of TML6 and Fandrianto discloses or renders obvious claim 1. This combination also discloses or renders obvious the additional features of claim 9. For example, as illustrated below, the combination of TML6 and Fandrianto discloses interpolating the highlighted 1/4,0/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$, where K=1, L=0, N=2) with K equal to 1 and L equal to zero.



(1/4,0/4)
K=1, L=0

193. In Fandrianto, as illustrated below, the highlighted 1/4,0/4 sub-pixel "Q1" (**green**) is interpolated by averaging the values of a 0/4,0/4 pixel $X_{10,10}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=0, N=2) with K equal to zero and L equal to zero and a 2/4,0/4 sub-pixel $A_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=0, N=2) with K equal to 2 and L equal to zero.

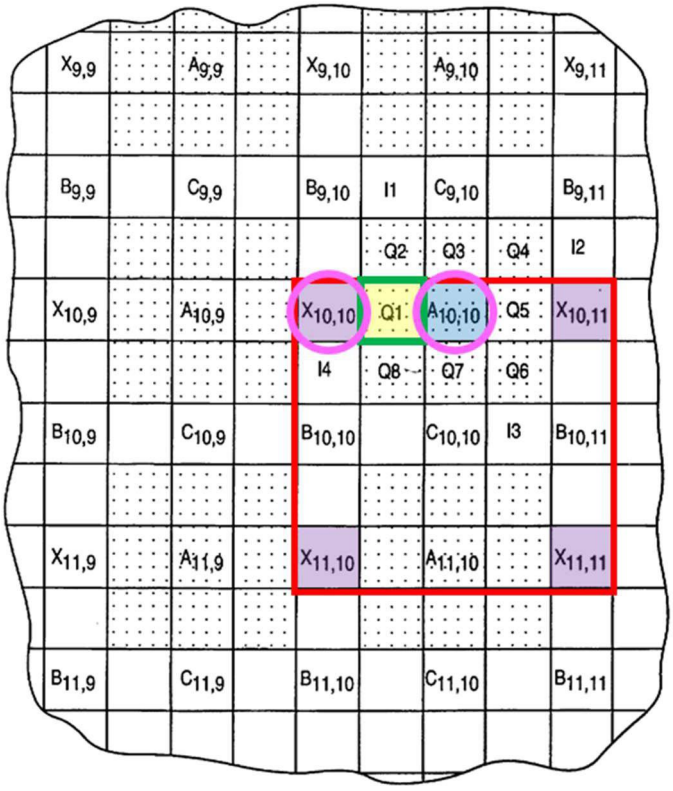


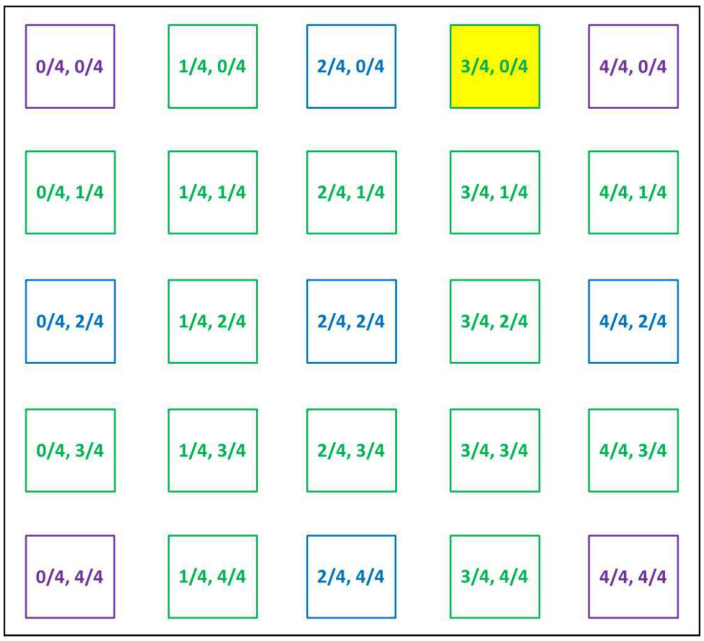
FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed…."), 8:29-57 ("$Q1_{r,c}=(X_{r,c}+A_{r,c})/2$").

194. As I discuss above in Section IX.A.1 (motivation to combine) and for Element 1[a], it would have been obvious to substitute Fandrianto's ¼ sub-pixel interpolation scheme for that of TML6. This combination interpolates the sub-pixels required by claim 9 using values for the recited first and second pixels and sub-pixels from the ***mref*** data structure of TML6. Section IX.A.1, Elements 1[a], *supra*.

h) Dependent Claim 10: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N-1 and L equal to zero, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to zero, and a sub-pixel having co-ordinates with K equal to 2^N-2 and L equal to zero"

195. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 1. TML6 and Fandrianto combined also discloses or renders obvious the additional features of claim 10. For example, as illustrated below, the combination discloses interpolating the highlighted 3/4,0/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$, where K=3, L=0, N=2) with K equal to 2^N-1 (*e.g.*, 3) and L equal to zero.



(3/4,0/4)
K=3, L=0

196. As further illustrated below, Fandrianto interpolates the highlighted 3/4,0/4 sub-pixel "Q5" (**green**) by averaging the values of a 4/4,0/4 pixel $X_{10,11}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=0, N=2) with K equal to 2^N (*e.g.*, 4) and L equal to zero and a 2/4,0/4 sub-pixel $A_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=0, N=2) with K equal to 2^N-2 (*e.g.*, 2) and L equal to zero.

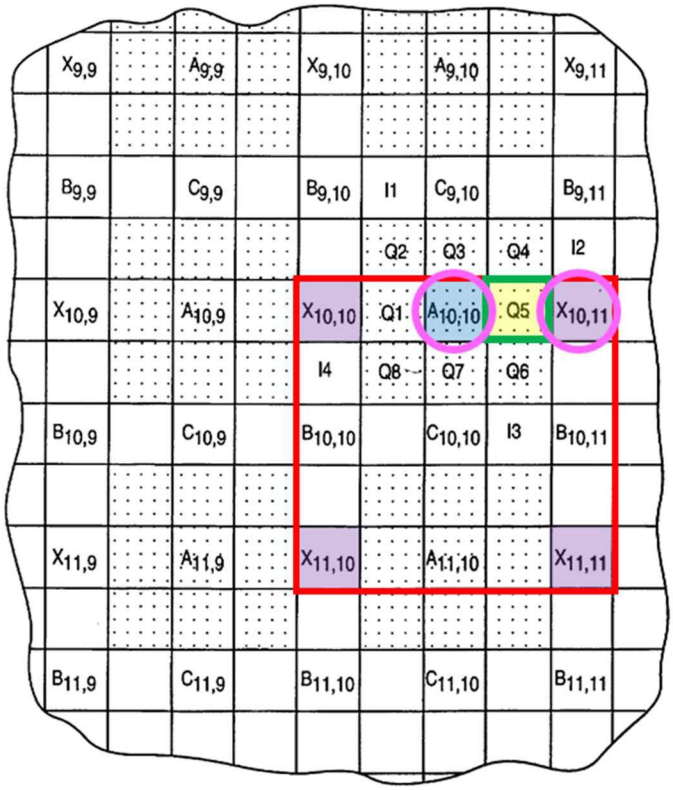


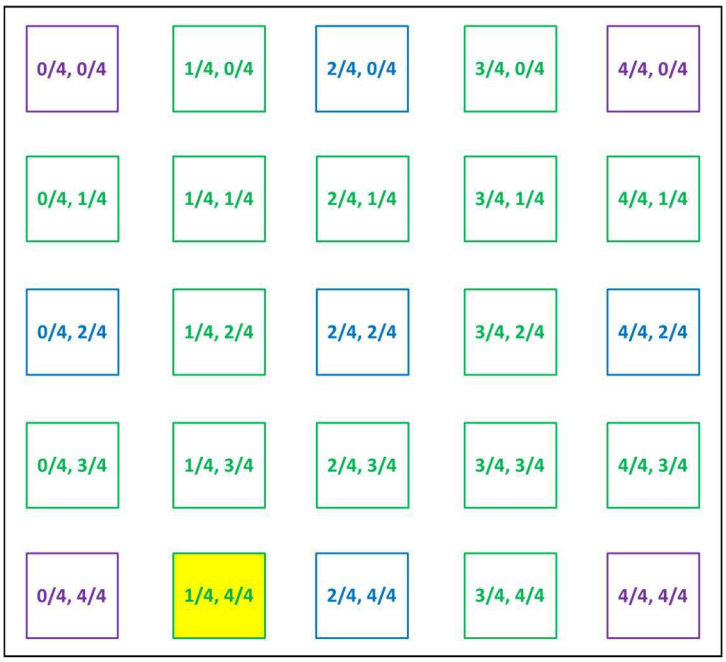
FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed…."), 8:29-57, 9:63-10:5 ("$Q5_{r,c}=(X_{r,c+1}+A_{r,c})/2$").

197. As I discuss above in Sections IX.A.1 (motivation to combine) and for Element 1[a], it would have been obvious to substitute Fandrianto's ¼ sub-pixel interpolation scheme for that of TML6. This combination interpolates the sub-pixel required by claim 10 using values for the required pixel and sub-pixel from the ***mref*** data structure of TML6. Section IX.A.1, Elements 1[a], *supra*.

i) Dependent Claim 11: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 1 and L equal to 2^N, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to 2^N, and a sub-pixel having co-ordinates with K equal to 2 and L equal to 2^N"

198. As I discuss above, the combination of TML6 and Fandrianto discloses or renders obvious claim 1. This combination also discloses or renders obvious the additional features of claim 11. For example, as illustrated below, the combination of TML6 with Fandrianto discloses interpolating the highlighted 1/4,4/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$, where K=1, L=4, N=2) with K equal to 1 and L equal to 2^N.



(1/4,4/4)
K=1, L=4

199. As further illustrated below, in Fandrianto, the highlighted 1/4,4/4 sub-pixel "Q1" (**green**) is interpolated by averaging the values of a 0/4,4/4 pixel $X_{10,10}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=4, N=2) with K equal to zero and L equal to 2^N and a 2/4,4/4 sub-pixel $A_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=4, N=2) with K equal to 2 and L equal to 2^N.

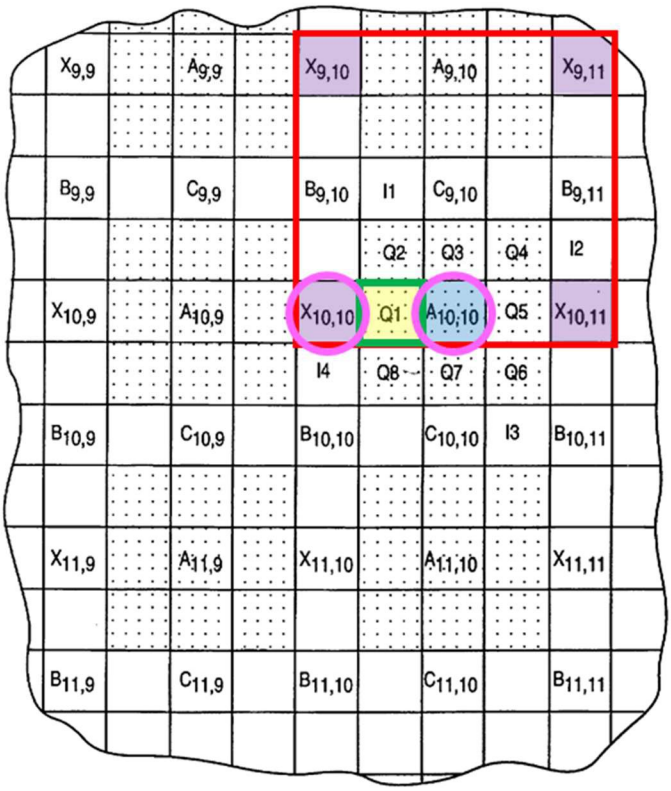


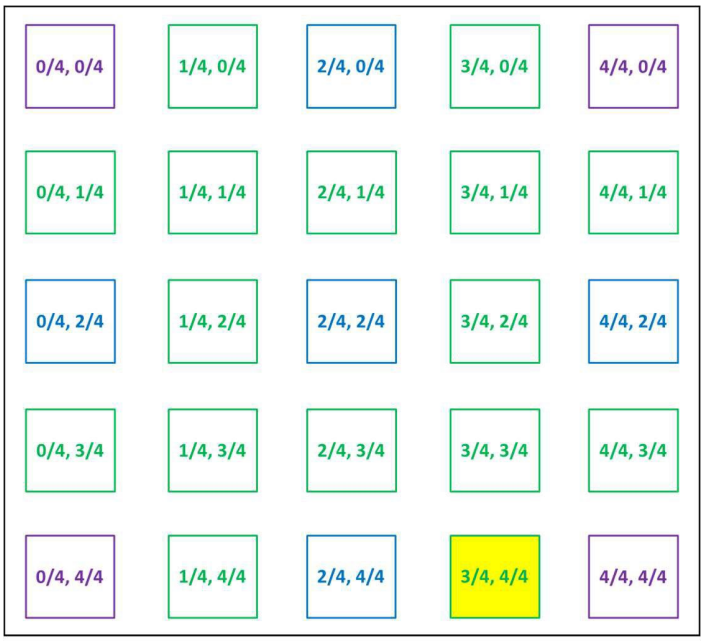
FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed…."), 8:29-57 ("$Q1_{r,c}=(X_{r,c}+A_{r,c})/2$").

200. As I discuss above in Section IX.A.1 (motivation to combine) and for Element 1[a], it would have been obvious for a POSITA to substitute Fandrianto's interpolation scheme for that of TML6. In this combination, TML6 would interpolate the sub-pixel required by claim 11 using values for the recited pixel and sub-pixel from the ***mref*** data structure of TML6. Section IX.A.1, Elements 1[a], *supra*.

j) Dependent Claim 12: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N-1 and L equal to 2^N, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to 2^N, and a sub-pixel having co-ordinates with K equal to 2^N-2 and L equal to 2^N"

201. As I discuss above, the combination of TML6 and Fandrianto discloses or renders obvious claim 1. This combination also discloses or renders obvious the additional features of claim 12. For example, as illustrated below, the combination discloses interpolating the highlighted 3/4,4/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$, where K=3, L=4, N=2) with K equal to 2^N-1 (*e.g.*, 3) and L equal to 2^N (*e.g.*, 4).



¾,4/4)

K=1, L=4

202. As further illustrated below, in Fandrianto, the highlighted 3/4,4/4 sub-pixel "Q5" (**green**) is interpolated by averaging the values of a 4/4,4/4 pixel $X_{10,11}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=4, N=2) with K equal to 2^N (*e.g.*, 4) and L equal to 2^N and a 2/4,4/4 sub-pixel $A_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=4, N=2) with K equal to 2^N-2 (*e.g.*, 2) and L equal to 2^N (*e.g.*, 4) ($K/2^N$, $L/2^N$; K=2, L=4, N=2).

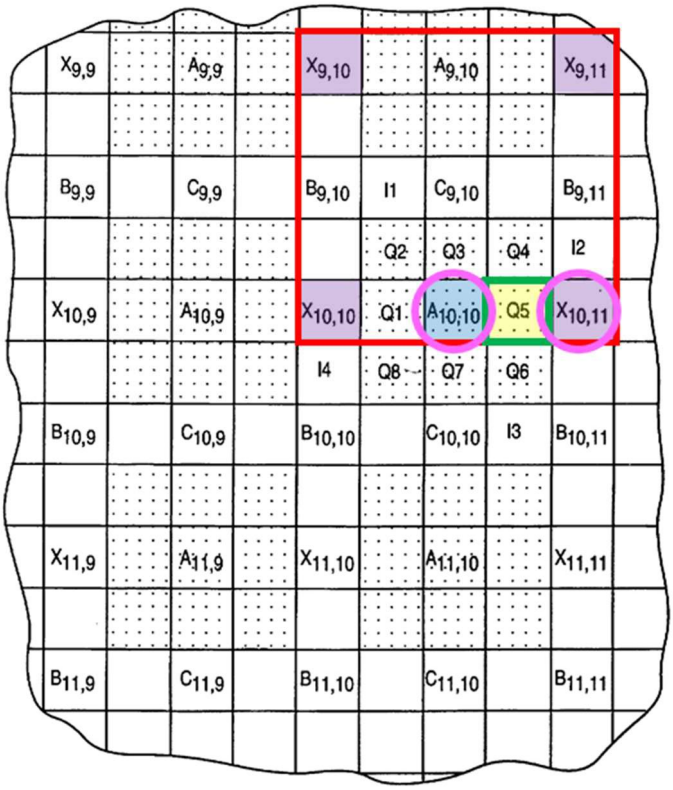


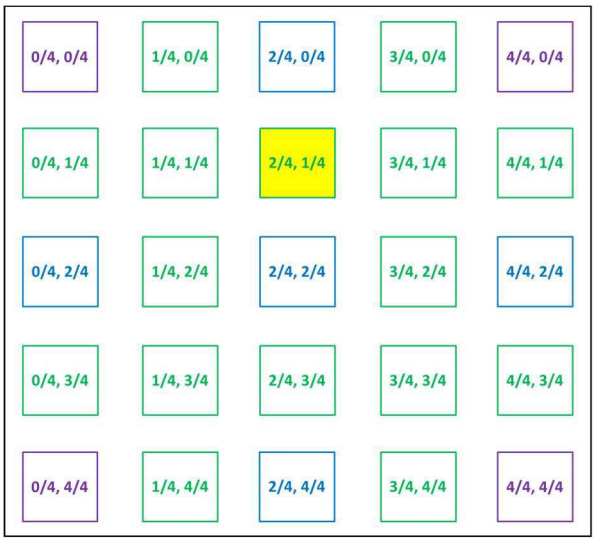
FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed…."), 8:29-57, 9:63-10:5 ("$Q5_{r,c}=(X_{r,c+1}+A_{r,c})/2$").

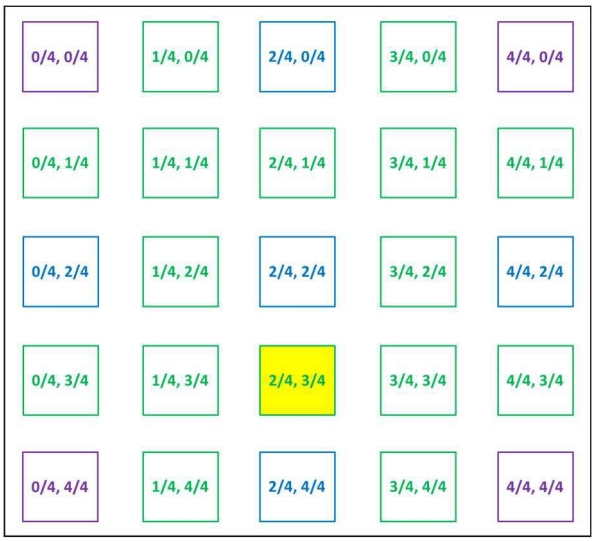
203. As I discuss above in Section IX.A.1 (motivation to combine) and for Element 1[a], it would have been obvious for a POSITA to substitute Fandrianto's interpolation scheme into TML6. This combination of TML6 and Fandrianto would interpolate the sub-pixels required by claim 12 using values for the recited pixel and sub-pixel from the ***mref*** data structure of TML6. Section IX.A.1, Elements 1[pre][a], *supra*.

k) Dependent Claim 13: "A method according to claim 1, comprising causing the apparatus to interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value E, not including zero or 2^N, and L equal to an odd value O, by taking an average of the value of a first sub-pixel having co-ordinates with K equal to E and L equal to the even value O−1 and a second sub-pixel having co-ordinates with K equal to E and L equal to the even value O+1"

204. Claim 13 would have been obvious in view of TML6 in combination with Fandrianto. As explained and illustrated below, the combination of TML6 and Fandrianto discloses interpolating values for the highlighted 2/4,1/4 and 2/4,3/4 sub-pixels, which have co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=1 or 3, N=2) with K equal to an even value E (*e.g.*, 2), not including zero or 2^N (*i.e.*, 4) and L equal to an odd value O (*e.g.*, 1 or 3).



(2/4,1/4)
K=2, L=1



(2/4,3/4)
K=2, L=3

Element 1[pre][a], *supra*.

205. In addition, as illustrated below on the left, Fandrianto interpolates the highlighted 2/4,1/4 sub-pixel "Q7" (**green**) by averaging the values of a first 2/4,0/4 sub-pixel $A_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=0, N=2) with K equal to E (*e.g.*, 2) and L equal to the even value O-1 (*e.g.*, 0 (1-1)) and a second 2/4,2/4 sub-pixel $C_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=2, N=2) with K equal to E (*e.g.*, 2) and L equal to the even value O+1 (*e.g.*, 2 (1+1)). As demonstrated below on the right, Fandrianto interpolates the highlighted 2/4,3/4 sub-pixel "Q3" (**green**) by averaging the values of a first 2/4,2/4 sub-pixel $C_{9,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=2, N=2) with K equal to E (*e.g.*, 2) and L equal to the even value O-1 (*e.g.*, 2 (3-1)) and a second 2/4,4/4 sub-pixel $A_{10,10}$

(**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=4, N=2) with K equal to E (*e.g.*, 2) and L equal to the even value O+1 (*e.g.*, 4 (3+1)).

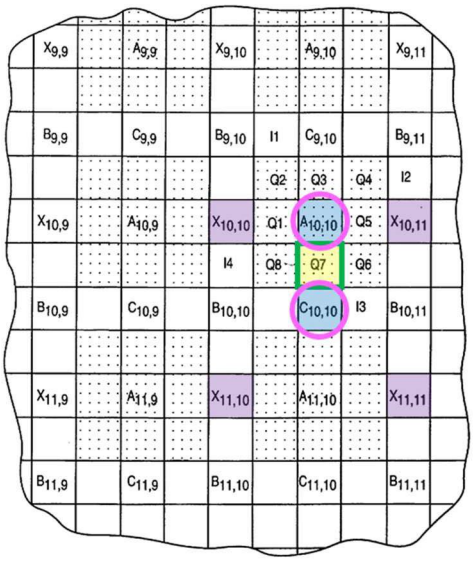


FIG. 5

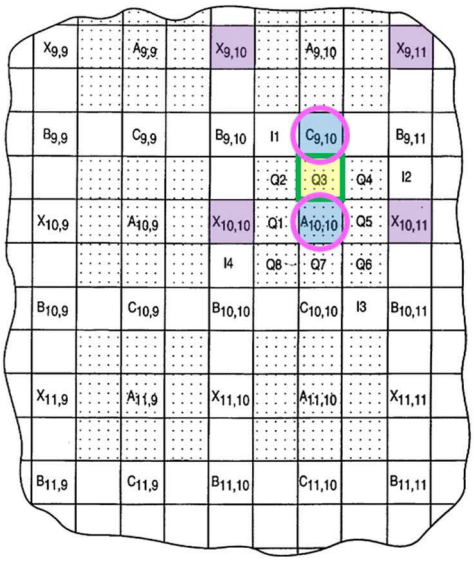


FIG. 5

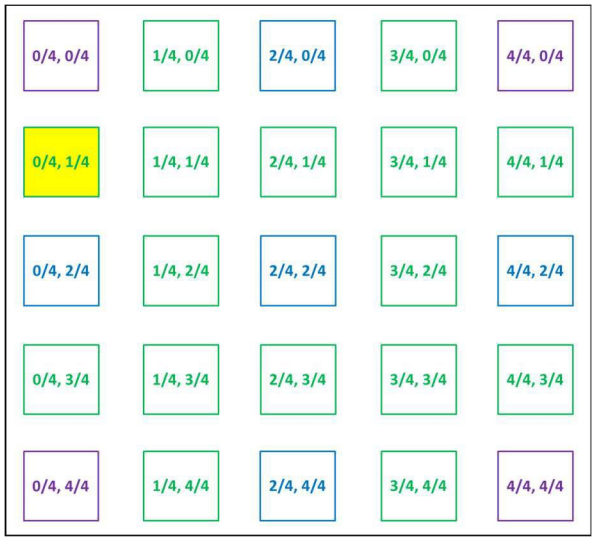
Fandrianto, Figure 5, 5:26-32, 8:29-57, 10:23-33 ("$Q7_{r,c}=(C_{r,c}+A_{r,c})/2$"), 9:32-44 ($Q3_{r,c}=(C_{r-1,c+1}+A_{r,c+1})/2$").

206. It therefore would have been obvious to a POSITA to substitute Fandrianto's interpolation method for TML6's interpolation method. Section IX.A.1; Elements 1[pre][a], 1[a] and 1[c], *supra*. In such a combination, TML6 would interpolate the sub-pixels (as set forth in claim 13) by averaging the values of the appropriate sub-pixels from the ***mref*** data structure of TML6. *Id*.

l) Dependent Claim 14: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to zero and L equal to 1, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to zero, and a sub-

pixel having co-ordinates with K equal to zero and L equal to 2"

207. Claim 14 would have been obvious over TML6 and Fandrianto. As illustrated below, the combined teachings of these references teach interpolating values for the highlighted 0/4,1/4 sub-pixels, which have co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=1, N=2) where K is equal to zero and L is equal to 1.



(0/4,1/4)
K=0, L=1

Element 1[pre][a], *supra*.

208. In addition, as illustrated below, Fandrianto discloses that the highlighted 0/4,1/4 sub-pixel "I4" (**green**) is interpolated by averaging the values of a 0/4,0/4 pixel $X_{10,10}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=0, N=2) where K is equal to zero and L is equal to zero and a 0/4,2/4 sub-pixel $B_{10,10}$ (**blue** and circled), which has co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=2, N=2) where K is equal to zero and L is equal to 2.

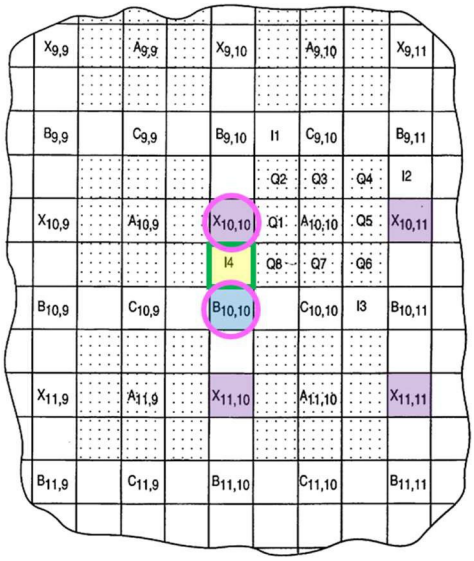


FIG. 5

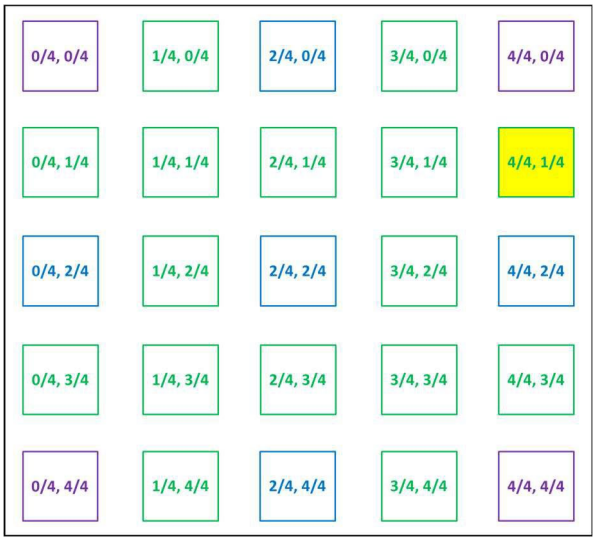
Fandrianto, Figure 5, 5:26-32, 8:29-57, 10:34-10:50 ("$I4_{r,c}=(B_{r,c}+X_{r,c})/2$").

209. It would have been obvious to a POSITA to substitute Fandrianto's interpolation method for the method disclosed in TML6. Section IX.A.1, Elements 1[pre][a], 1[a], 1[c], *supra*. With this combination of TML6 and Fandrianto, TML6 would interpolate the sub-pixels as stated in claim14 by averaging values of the appropriate sub-pixels (as claimed) from the ***mref*** data structure of TML6. *Id*.

m) Dependent Claim 15: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N and L equal to 1, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to zero, and a sub-pixel having co-ordinates with K equal to 2^N and L equal to 2"

210. Claim 15 would have been obvious over TML6 and Fandrianto. As demonstrated below, the combination of prior art discloses interpolating the values

for the highlighted 4/4,1/4 sub-pixels having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=1, N=2) where K is equal to 2^N (*i.e.*, 4) and L is equal to 1.



(4/4,1/4)
K=4, L=1

Element 1[pre][a], *supra*.

211. As shown below, Fandrianto interpolates the highlighted 4/4,1/4 sub-pixel "I4" (**green**) by averaging the values of a 4/4,0/4 pixel $X_{10,10}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=0, N=2) with K equal to 2^N (*e.g.*, 4) and L equal to zero and a 4/4,2/4 sub-pixel $B_{10,10}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=2, N=2) with K equal to 2^N and L equal to 2.

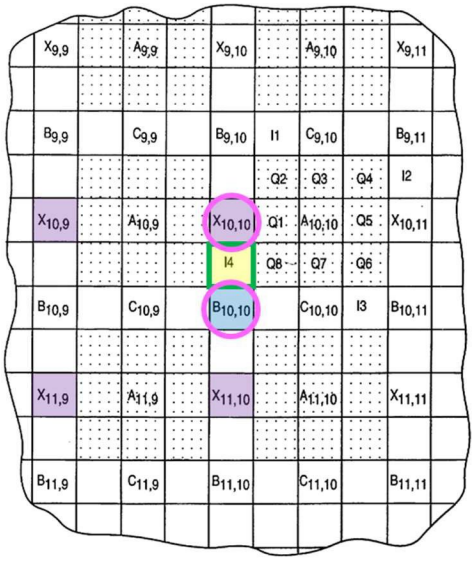


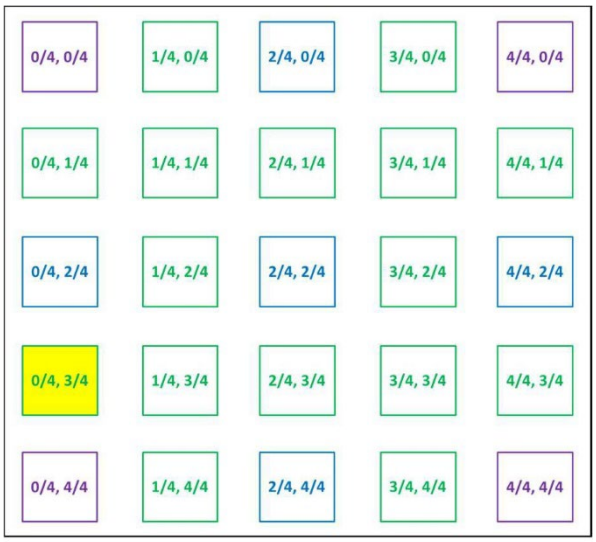
FIG. 5

Fandrianto, Figure 5, 5:26-32, 8:29-57, 10:34-10:50 ("$I4_{r,c}=(B_{r,c}+X_{r,c})/2$").

212. It would have been obvious to a POSITA to swap Fandrianto's interpolation technique for TML6's interpolation technique. Section IX.A.1, Elements 1[pre][a], 1[a], 1[c], Claim 14, *supra*.

n) Dependent Claim 16: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to zero and L equal to 2^N-1, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to 2^N, and a sub-pixel having co-ordinates with K equal to zero and L equal to 2^N-2"

213. Claim 16 would have been obvious over TML6 and Fandrianto. As illustrated below, this prior art, in combination, discloses interpolating values for the highlighted 0/4,3/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=3, N=2) where K is equal to 0 and L is equal to 2^N-1 (*e.g.*, $3=2^2-1$).



(0/4,3/4)
K=0, L=3

Element 1[pre][a], *supra*.

214. As demonstrated in the figure below of Fandrianto (which has been annotated to illustrate unit pixels $X_{9,12}$ and $X_{10,12}$), the highlighted 0/4,3/4 sub-pixel "I2" (**green**) is interpolated by averaging the values of a 0/4,4/4 pixel $X_{10,11}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=4, N=2) where K is equal to 0 and L is equal to 2^N (*e.g.*, 4) and a 0/4,2/4 sub-pixel $B_{9,11}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=0, L=2, N=2) where K is equal to zero and L is equal to 2^N-2 ($2=2^2$-2).

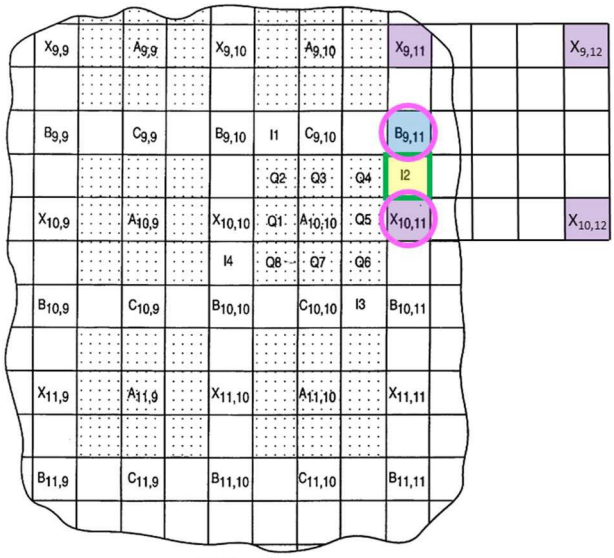


FIG. 5

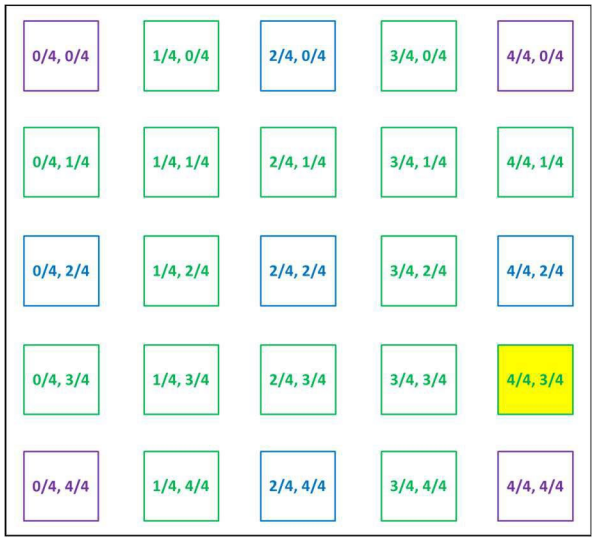
Fandrianto, Figure 5, 5:26-32, 8:29-57, 9:45-62 ($I2_{r,c}=(B_{r-1,c+1}+X_{r,c+1})/2$").

215. It would have been obvious to a POSITA to incorporate Fandrianto's interpolation technique into TML6. Section IX.A.1, Elements 1[pre][a], 1[a], 1[c], *supra*. Therefore, according to the combination of prior art, TML6 would interpolate the sub-pixels as set forth in claim 16 by averaging values of the sub-pixels (as recited in the claim) from the ***mref*** data structure of TML6. *Id*.

o) Dependent Claim 17: "A method according to claim 1, comprising causing the apparatus to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N and L equal to 2^N-1, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to 2^N, and a sub-pixel having co-ordinates with K equal to 2^N and L equal to 2^N-2"

216. Claim 17 would have been obvious over TML6 and Fandrianto. As illustrated below, the combined teachings of TML6 and Fandrianto disclose

interpolating values for the highlighted 4/4,3/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=3, N=2) where K is equal to 2^N (*i.e.*, 4) and L is equal to 2^N-1 (*e.g.*, $3=2^2-1$).



(4/4,3/4)
K=4, L=3

Element 1[pre][a], *supra*.

217. As illustrated below in Figure 5 of Fandrianto, the highlighted 4/4,3/4 sub-pixel "I2" (**green**) is interpolated by averaging the values of a 4/4,4/4 pixel $X_{10,11}$ (**purple** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=4, N=2) where K is equal to 2^N (*i.e.*, 4) and L is equal to 2^N and a 4/4,2/4 sub-pixel $B_{9,11}$ (**blue** and circled) having co-ordinates ($K/2^N$, $L/2^N$, where K=4, L=2, N=2) where K is equal to 2^N (*i.e.*, 4) and L is equal to 2^N-2 ($2=2^2-2$).

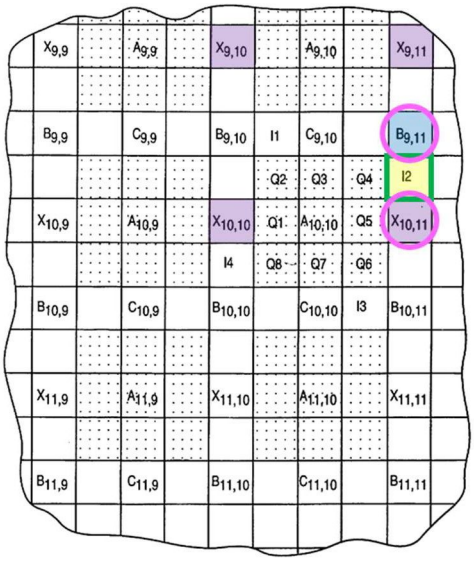


FIG. 5

Fandrianto, Figure 5, 5:26-32, 8:29-57, 9:45-62 ($I2_{r,c}=(B_{r-1,c+1}+X_{r,c+1})/2$").

218. It would have been obvious to a POSITA to substitute Fandrianto's interpolation scheme for TML6's scheme. Section IX.A.1, Elements 1[pre][a], 1[a], 1[c], Claim 16, *supra*.

p) Dependent Claim 19: "A method according to claim 1, in which N is one of the values 2, 3, and 4."

219. As I have described above in my assessment of Element 1[pre][b], the combined teachings of TML6 and Fandrianto disclose the specific case where N equals 2 (*i.e.*, "one of the values 2, 3, and 4"). Accordingly, TML6 and Fandrianto, in combination, disclose or render obvious claim 19 for the reasons that I set forth for Elements 1[pre][a]-1[c]. Elements 1[pre][a]-1[c], *supra*.

q) Dependent Claim 20: "A method according to claim 1, wherein the pixels have a predetermined dynamic range, said predetermined dynamic range

corresponding to the range of values said pixels can take"

220. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 1. TML6 and Fandrianto combined also discloses or renders obvious "the pixels have a predetermined dynamic range, said predetermined dynamic range corresponding to the range of values said pixels can take."

221. TML6 discloses that the pixel values stored in the data structures ***imgY*** and ***mref*** are 8-bit values representing integers in a dynamic range from 0-255 that represent the values that the pixels can take.[19] TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 563-625), 131 (/tml6pre.zip/ldecod/inc/global.h, ll. 131-132), 72-73 (/tml6pre.zip/lencod/src/lencod.c, ll. 215-221 ("currently only 8 bit/pel is supported")). For example, values representing unit pixels in the data structure ***imgY*** that are within the dynamic range are multiplied by 1024 when written to the data structure ***img4Y_tmp***.

[19] The '273 patent confirms my assessment that TML6 discloses a dynamic range of "a minimum value of 0 and a maximum value of 2^n-1, where n is the number of bits reserved for a pixel value … typically 8." EX-1001, 9:54-64. The range 0 to 2^8-1 represents a dynamic range between 0 and 255.

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

Id., 57 (/tml6pre.zip/lencod/src/image.c, l. 571). These values from the data structure ***img4Y_tmp*** are normalized using division by 1024 ("/1024") to bring them back to the dynamic range of 0-255, and stored in the data structure ***mref*** as integer values within the range of 0-255 ("max(0,min(255,(int)").

```
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
```

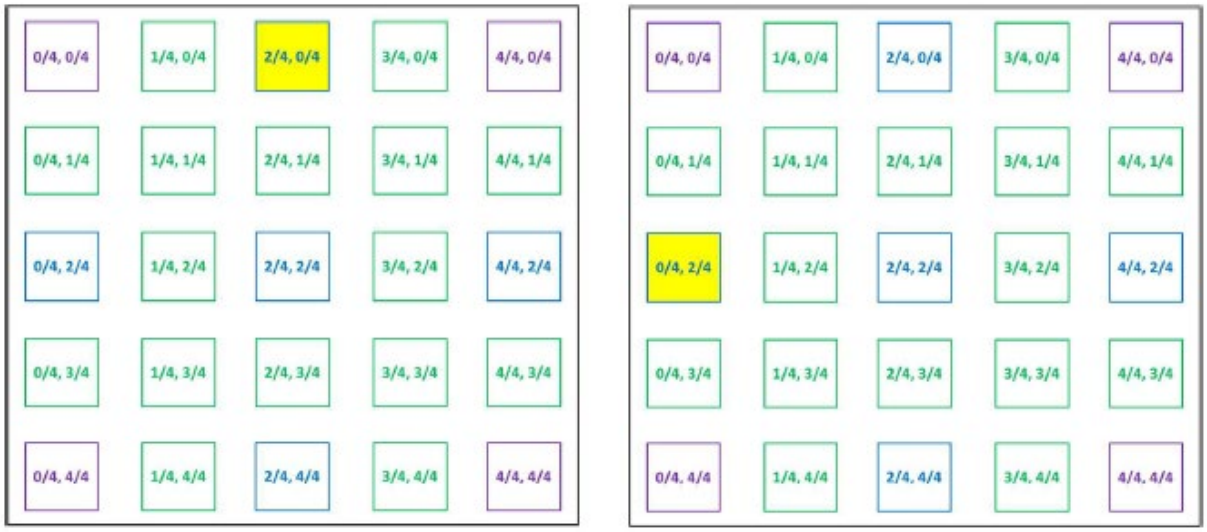
Id., 58 (/tml6pre.zip/lencod/src/image.c, l. 588).

r) Dependent Claim 21: "A method according to claim 20, comprising interpolating sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and for sub-pixels having co-ordinates with K equal to zero and L equal to an even value, using weighted sums of the values of pixels located in rows and columns respectively, to produce a first intermediate value, said first intermediate value having a dynamic range equal to the predetermined dynamic range of the pixels used in said weighted sum multiplied by a value equal to the

sum of the respective weights used in said weighted sum"

222. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 20. TML6 and Fandrianto combined also discloses or renders obvious the additional limitations of claim 21.

223. As I discuss above in my assessment of Element 1[b], TML6 discloses interpolating "values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and for sub-pixels having co-ordinates with K equal to zero and L equal to an even value" using weighted sums of the values of pixels (**purple**) located in rows and columns, respectively. Element 1[b].



224. Claim 21 also requires "interpolating [these] sub-pixel values…to produce a first intermediate value." TML6 discloses two "alternative" first intermediate values for the 2/4,0/4 sub-pixel: (1) ***is*** and (2) ***is**32**.

225. Regarding the first alternative, the highlighted 2/4,0/4 sub-pixels (**blue**) are interpolated using the values of the unit pixels (**purple**) from the ***imgY*** data structure to create a first intermediate value ***is***.

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 563-575, particularly l. 568). This first intermediate value ***is*** has a dynamic range (0-8160) equal to the dynamic range (0-255) of the values of the pixels (**purple**) from the data structure ***imgY*** used in the weighted sum multiplied by a value (**32**) equal to the sum of the respective weights used in the weighted sum from the ***ONE_FOURTH_TAP*** data structure (20+20-5-5+1+1=**32**).

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

```
28    /* TAPs used in the oneforthpix()routine */
29     const int ONE_FOURTH_TAP[3][2] =
30       {
31         {20,20},
32         {-5,-4},
33         { 1, 0},
34       };
35
```

Id.; *id.,* 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); Elements 1[b]-1[c], *supra*.

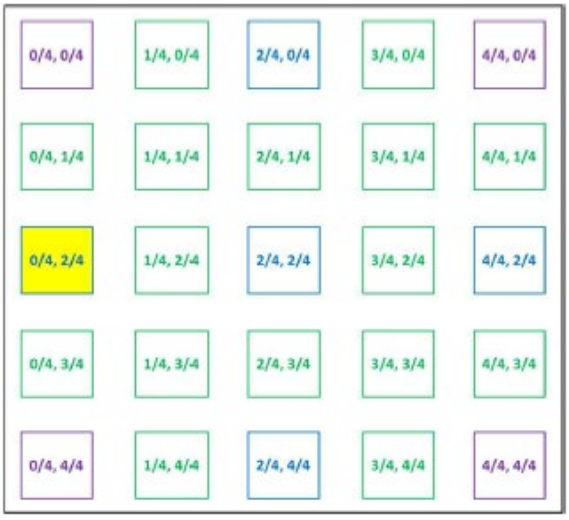
226. For the second alternative, the first intermediate value ***is*32**—representing 2/4,0/4 sub-pixels—is written to the data structure ***img4Y_tmp***. TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 563-575, particularly l. 573). This value ***is*32** has a dynamic range (0-261120) equal to the dynamic range (0-255) of the values of the pixels (**purple**) from the data structure ***imgY*** used in the weighted sum multiplied by a value (**1024**) equal to the sum of the respective weights used in the weighted sum, which are from the ***ONE_FOURTH_TAP*** data structure and multiplied by 32 (32*20+32*20-32*5-32*5+32*1+32*1=**1024**).

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

```
28    /* TAPs used in the oneforthpix()routine */
29     const int ONE_FOURTH_TAP[3][2] =
30       {
31         {20,20},
32         {-5,-4},
33         { 1, 0},
34       };
35
```

Id.; *id.,* 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); Elements 1[b]-1[c], *supra*.

227. TML6 similarly discloses first intermediate values ***is*** for the 0/4,2/4 sub-pixel, which are calculated using the values representing the unit pixels (**purple**) in the data structure ***img4Y_tmp***.



(0/4,2/4)
K=0, L=2

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592, particularly ll. 583-585). This first intermediate value ***is*** has a dynamic range (0-261120 (=0-255*1024)) equal to the dynamic range (0-255) of the values of the

unit pixels (**purple**) used in the weighted sum multiplied by a value (**1024**) equal to the sum of the respective weights used in the weighted sum (1024*20/32+1024*20/32-1024*5/32-1024*5/32+1024*1/32+1024*1/32=**1024**). *Id.*; *see* Elements 1[b]-1[c], *supra*. These respective weights used in the weighted sum are calculated using values from ***ONE_FOURTH_TAP***,

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

multiplied by 1024 (from writing unit pixels (**purple**) to the data structure ***img4Y_tmp***),

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

and divided by 32 (from the calculation of ***is***).

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

Id.; TML6, 57 (/tml6pre.zip/lencod/inc/image.h, ll.563-575), 3

(/tml6pre.zip/lencod/inc/image.h, ll. 28-34).

228. The ’273 patent explains that TML6 interpolates the 2/4,0/4 and 0/4,2/4 sub-pixels (**blue**) to produce a first intermediate value having a dynamic range equal to the predetermined dynamic range of the pixels (**purple**) used in the weighted sum multiplied by a value equal to the sum of the respective weights used in the weighted sum.

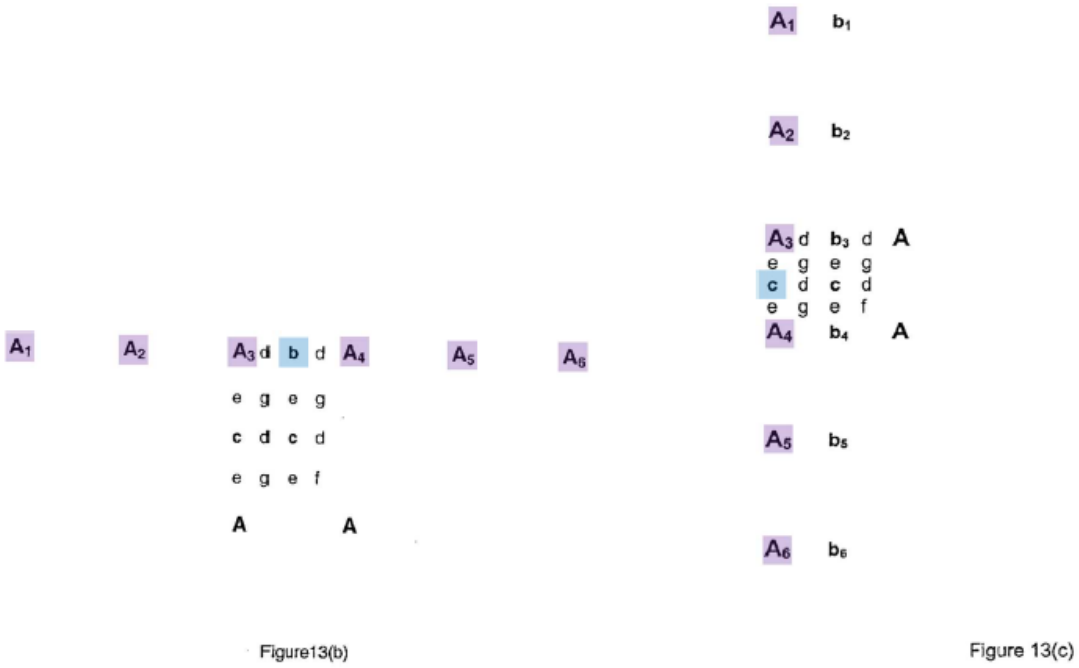


Figure13(b) Figure 13(c)

EX-1001, Figures 13(a)-13(b), 11:35-63 (“[A]n intermediate value b … [is calculated] based upon the values of the 6 pixels (A_1 to A_6) …, as shown in FIG. 13*b*, … $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$…. Referring to FIG. 13*c*, an intermediate value c … is calculated based upon the values of the 6 pixels (A_1 to A_6) … $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$.”).

s) Dependent Claim 22: "A method according to claim 21, wherein the dynamic range of said first intermediate value is defined by the number of bits used to represent the respective pixels used in said weighted sum plus the number of bits required to represent the sum of the respective weights used in said weighted sum"

229. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 21. TML6 and Fandrianto combined also discloses or renders obvious "the dynamic range of said first intermediate value is defined by the number of bits used to represent the respective pixels used in said weighted sum plus the number of bits required to represent the sum of the respective weights used in said weighted sum." For example, as I discuss above, the dynamic range of the first intermediate value ***is*** for the 2/4,0/4 sub-pixel is 0-8160. Claim 21, Element 1[b], *supra*. This range is representable by 13 bits (*i.e.,* 2^{13}-1=8191 (required to represent 8160)). The dynamic range of the pixel values from the data structure ***imgY*** used to calculate the first intermediate value for the 2/4,0/4 sub-pixels is 0-255, which is defined by 8 bits (2^8-1=255). *Id.* The sum of the weights used in the weighted sum to calculate the first intermediate value for the 2/4,0/4 sub-pixels is 32, defined by 5 bits (2^5=32). *Id.* Thus, the dynamic range of the first intermediate value ***is*** for the 2/4,0/4 sub-pixel is defined by 13 bits (8+5 bits). *Id.*

230. As I discuss above in my assessments of claim 21 and Element 1[b], the dynamic range of the alternative first intermediate value ***is*32** for the 2/4,0/4

sub-pixel is (0-261120). *See* Element 1[b], Claim 21), *supra*. This range is representable by 18 bits (*i.e.,* 2^{18}-1=262,143 (required to represent 261,120)). The dynamic range of the pixel values from the data structure ***imgY*** used to calculate the first intermediate value for the 2/4,0/4 sub-pixels is 0-255, which is defined by 8 bits (2^8-1=255). *Id.* The sum of the weights used in the weighted sum to calculate the first intermediate value for the 2/4,0/4 sub-pixels is 1024, defined by 10 bits (2^{10}=1024). *Id.* Thus, the dynamic range of the first intermediate value ***is*** for the 2/4,0/4 sub-pixel is defined by 18 bits (8+10 bits).

231. Finally, as I also discuss above in my assessments of claim 21 and Element 1[b], the dynamic range of the first intermediate value ***is*** for the 0/4,2/4 sub-pixel is (0-261120). *See* Element 1[b], Claim 21, *supra*. This range is representable by 18 bits (*i.e.,* 2^{18}-1=262,143 (required to represent 261,120)). The dynamic range of the pixel values from the data structure ***imgY*** used to calculate the first intermediate value for the 0/4,2/4 sub-pixels is 0-255, which is defined by 8 bits (2^8-1=255). *Id.* The sum of the weights used in the weighted sum to calculate the first intermediate value for the 0/4,2/4 sub-pixels is 1024, defined by 10 bits (2^{10}=1024). *Id.* Thus, the dynamic range of the first intermediate value ***is*** for the 0/4,2/4 sub-pixel is defined by 18 bits (8+10 bits).

t) Dependent Claim 23: "A method according to claim 21, comprising truncating the first intermediate value by mathematically dividing the first intermediate

value by a first scale factor, said first scale factor having a value equal to the sum of the respective weights used in said weighted sum, thereby forming a sub-pixel value with a dynamic range equal to the predetermined dynamic range of said pixels"

232. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 21. TML6 and Fandrianto combined also discloses or renders obvious the additional limitations of claim 23.

233. As I discuss above in my assessment of claim 21, TML6 discloses two "alternative" first intermediate values for the 2/4,0/4 sub-pixels: (1) ***is*** and (2) ***is*32**. *See* Claim 21, *supra*. The first intermediate value ***is*** for the 2/4,0/4 sub-pixel is truncated by dividing the value ***is*** by a first scale factor **32**. Namely, as I highlight below, the value ***is*** is initially multiplied by 32 (which is different than the scale factor **32**) and written to the data structure ***img4Y_tmp***.

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 552-592, particularly l. 572); Claim 21, *supra*. This value (***is***×32) in the data structure ***img4Y_tmp*** is divided by 1024 (=**32**×32) (highlighted below) to create a truncated first intermediate value that is written to the data structure ***mref***.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 563-592, particularly l. 588). Accordingly, the first intermediate value ***is*** is truncated by dividing the value ***is*** by a first scale factor **32**: (***is***×32)/(**32**×32). Moreover, the scaling factor 32 equals the sum of the weights used in the weighted sum from ***ONE_FOURTH_TAP*** (20+20-5-5+1+1=**32**).

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

```
28    /* TAPs used in the oneforthpix()routine */
29     const int ONE_FOURTH_TAP[3][2] =
30      {
31        {20,20},
32        {-5,-4},
33        { 1, 0},
34      };
35
```

Id.; *id*, 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); Claim 21, Elements 1[b]-1[c], *supra*. Since ***is*** has a dynamic range of 0-8160, dividing it by the scaling factor **32** results in a dynamic range of 0-255 equal to the dynamic range of the unit pixels (8160/**32**=255). *Id.*

234. For the second alternative, the first intermediate value ***is*32** for the 2/4,0/4 sub-pixel is truncated by dividing the value ***is*32** by a first scale factor **1024**. The value ***is*32** (highlighted below) is written to the data structure ***img4Y_tmp***.

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3))));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 552-575, particularly l. 572); Claim 21, *supra*. The value ***is*32*** is divided by the first scale factor **1024** (highlighted below) when it is written from the data structure ***img4Y_tmp*** to the data structure ***mref***.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

Id.; TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 552-592, particularly l. 588). Accordingly, the first intermediate value ***is*32** is truncated by dividing the value ***is*32** by a first scale factor **1024**: (***is*32**)/**1024**. This scaling factor **1024** equals the sum of the weights (*i.e.,* 32*20+32*20-32*5-32*5+32*1+32*1=**1024**) used in the weighted sum. Section IX.A.4 (claim 21). Since ***is*32** has a dynamic range of (0-261120), dividing it by the scaling factor **1024** produces a dynamic range of 0-255 equal to the dynamic range of the unit pixels (261,120/**1024**=255). *Id.*

235. Similarly, the first intermediate value ***is*** for the 0/4,2/4 sub-pixel is truncated by dividing the value ***is*** by a first scale factor **1024**. The value ***is*** (highlighted below) is divided by a first scale factor **1024** when written to the data structure ***mref***.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

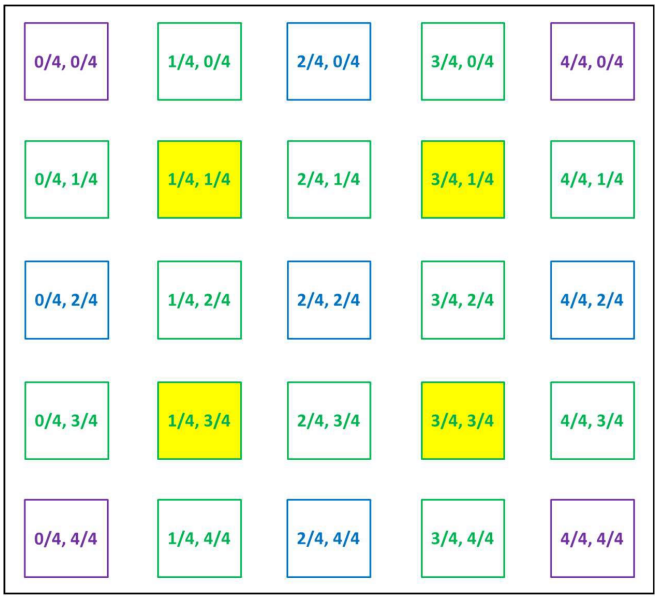
TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592, particularly l. 589); Claim 21, *supra*. This scaling factor **1024** equals the sum of the weights (*i.e.,* 32*20+32*20-32*5-32*5+32*1+32*1=**1024**) used in the weighted sum. Claim 21, *supra*. Since ***is*** has a dynamic range of (0-261120), dividing it by the scaling factor **1024** produces a dynamic range of 0-255 equal to the dynamic range of the unit pixels (261,120/**1024**=255). *Id.*

236. The '273 patent confirms my assessment that TML6 discloses truncation, as claimed. EX-1001, 11:35-45 ("The final value of b is then calculated as b=(b+16)/32 and is clipped to lie in the range [0, 2^n-1]."), 11:46-56 ("The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to c=(c+16)/32."). The preferred embodiment of the '273 patent also discloses operation in this manner. *Id.,* 45:3-43.

u) Dependent Claim 24: "A method according to claim 23, comprising using a truncated first intermediate sub-pixel value when interpolating a value for a sub-pixel having co-ordinates with odd values of K and L"

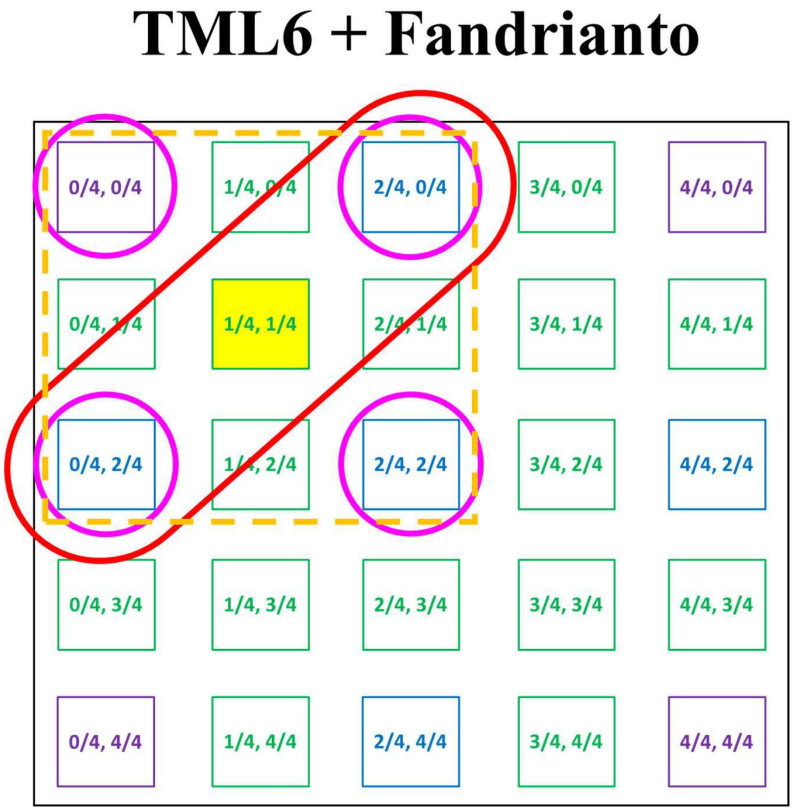
237. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 24. TML6 and Fandrianto combined also discloses or renders obvious "comprising using a truncated first intermediate sub-pixel value when interpolating a value for a sub-pixel having co-ordinates with odd values of K and L."

238. As I discuss above in my assessment of Element 1[a], sub-pixels having co-ordinates with odd values of K and L includes the ¼,¼; ¼,¾; ¾,¼; or ¾,¾ sub-pixel (**green**), highlighted below. *See* Element 1[a], *supra*.



K=Odd, L=Odd

For example, as shown below, the combination of TML6 and Fandrianto discloses interpolating values for the highlighted ¼,¼ sub-pixel (**green**) based on a predetermined choice of two weighted averages, one of which (below) includes values for circled (**red**) sub-pixels at the 2/4,0/4 and at 0/4,2/4 co-ordinates (**blue**), stored in the data structure ***mref***.



Id. As I discussed above in my assessment of claim 23, these values stored in the data structure ***mref*** are the truncated first intermediate values for the 2/4,0/4 sub-pixels and the 0/4,2/4 sub-pixels.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE FOURTH TAP[0][0]*(img4Y tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                    img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

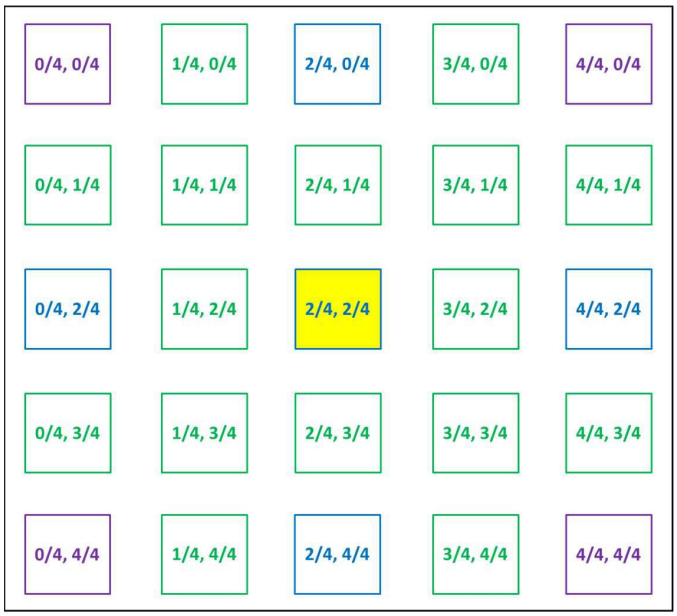
Claim 23, *supra*. Therefore, the combination of TML6 and Fandrianto disclose using a truncated first intermediate sub-pixel value when interpolating a value for a sub-pixel having co-ordinates with odd values of K and L. Element 1[a], Claim 23, *supra*.

v) **Dependent Claim 25: "A method according to claim 21, comprising interpolating sub-pixel values for sub-pixels having co-ordinates with even values of K and L, using a weighted sum comprising first intermediate values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and corresponding first intermediate values for sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, to produce a second intermediate value, said second intermediate value having a dynamic range equal to the dynamic range of the first intermediate values used in said**

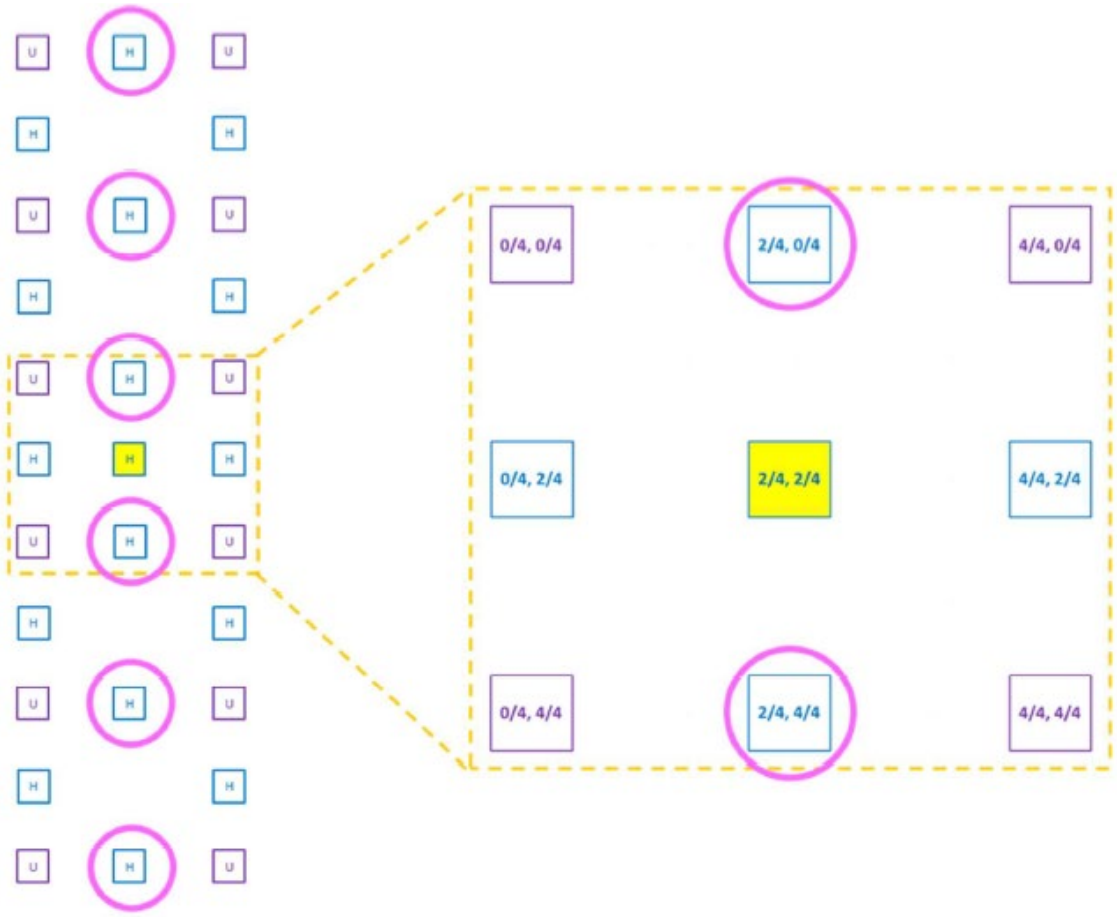
weighted sum, multiplied by a value equal to the sum of the respective weights used in said weighted sum"

239. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 21. TML6 and Fandrianto combined also discloses or renders obvious the limitations of claim 25.

240. As I discuss above in my assessment of Element 1[c], TML6 discloses that the highlighted 2/4,2/4 sub-pixels (**blue**) having co-ordinates with even values of K and L ($K/2^N$, $L/2^N$, where K=L=N=2) are interpolated using values for the 2/4,0/4 sub-pixels (**blue**) (and corresponding values for sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions) from the data structure ***img4Y_tmp***, to produce a second intermediate value ***is***.



(2/4,2/4)
K=2, L=2



First Weighted Sum

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 563-592, particularly l. 583); Element 1[c], *supra*. This second intermediate value ***is*** has a dynamic range (0-261120) of values equal to the dynamic range (0-261120) of the first intermediate values (corresponding to the 2/4,0/4 sub-pixels (**blue**) in the data structure ***img4Y_tmp*** from the data structure ***imgY***) (Claim 21, *supra*) used in the weighted sum, multiplied by a value (1) equal to the sum of the respective weights used in the weighted sum. As highlighted below, this weighted sum (for the value ***is***) has weights equal to the values from the ***ONE_FOURTH_TAP*** data structure divided by 32 (*i.e.,* 20/32+20/32-5/32-5/32+1/32+1/32=1).

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

```
28    /* TAPs used in the oneforthpix()routine */
29     const int ONE_FOURTH_TAP[3][2] =
30      {
31        {20,20},
32        {-5,-4},
33        { 1, 0},
34      };
35
```

Id.; TML6, 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); Claim 21, *supra*.

241. As an alternative to setting the second intermediate value to the ***entire*** value of ***is***, the second intermediate value may be the portion of the value ***is*** (within **brown** boundary below) before it is divided by 32. In this case, the second intermediate value has a dynamic range of ***is*** times 32 (*i.e.,* 0-8,355,840) of values equal to the dynamic range (0-261120) of the first intermediate values (corresponding to the 2/4,0/4 sub-pixels (**blue**) in the data structure ***img4Y_tmp***)

(*see* Claim 23, *supra*) used in the weighted sum multiplied by a value (32) equal to the sum of the respective weights used in the weighted sum.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 3 (/tml6pre.zip/lencod/inc/image.c, ll. 563-592); Claims 21, Claim 23, *supra*. As highlighted below, the weights for the second intermediate value (in **brown**) are values from the ***ONE_FOURTH_TAP*** data structure (20+20-5-5+1+1=32).

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

```
28    /* TAPs used in the oneforthpix()routine */
29     const int ONE_FOURTH_TAP[3][2] =
30       {
31         {20,20},
32         {-5,-4},
33         { 1, 0},
34       };
35
```

Id.; TML6, 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34).

242. As shown below, the '273 patent confirms that TML6 interpolates the 2/4,2/4 sub-pixels (**blue**) to produce a second intermediate value having a dynamic range equal to the predetermined dynamic range of the 2/4,0/4 sub-pixels (**blue**) used in the weighted sum multiplied by a value equal to the sum of the respective weights used in the weighted sum.

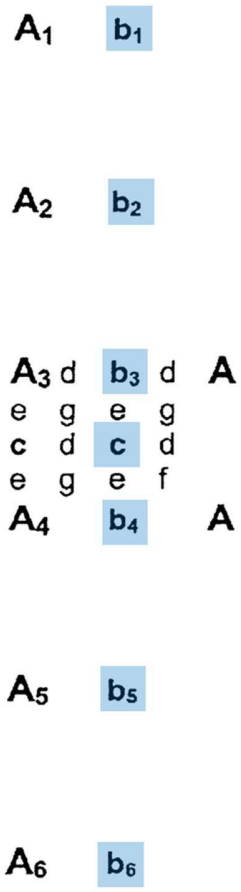


Figure 13(c)

EX-1001, Figures 13(a)-13(c), 11:46-63 ("[A]n intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$").

w) **Dependent Claim 26: "A method according to claim 25, comprising truncating the second intermediate value by mathematically dividing the second intermediate value by a second scale factor, said second scale factor having a value equal to the sum of the respective weights used in the weighted sum of claim 21 multiplied by the sum of the respective weights used in the weighted sum of claim 25, thereby**

forming a sub-pixel value with a dynamic range equal to the predetermined dynamic range of said pixels"

243. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 25. TML6 and Fandrianto combined also discloses or renders obvious the limitations of claim 26.

244. As I discussed in my assessment of claim 25, TML6 discloses the second intermediate value ***is*** for the 2/4,2/4 sub-pixels.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                  /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

Claim 25, *supra*. As highlighted below, this value ***is*** is truncated by dividing it by a second scale factor **1024** and then written to the data structure ***mref***.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

Id.; TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 552-592, particularly l. 589). The second scale factor **1024** is equal to the sum of the respective weights used in the weighted sum of claim 21 (*i.e.,* **1024** when the value ***is**32** constitutes the first intermediate values for the 2/4,0/2 sub-pixels) multiplied by the sum of the weights used in the weighted sum of claim 25 (*i.e.,* 1 when the value is constitutes the second intermediate values for the 2/4,2/4 sub-pixels). *Id.*; Elements 1[b]-[c], Claim 21, Claim 25, *supra*. Since the second intermediate value ***is*** has a dynamic range of (0-261120), dividing by the scaling factor **1024** gives a dynamic range of 0-255 equal to the dynamic range of the unit pixels (261,120/**1024**=255). *Id.*

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

Id; Claim 20, *supra*.

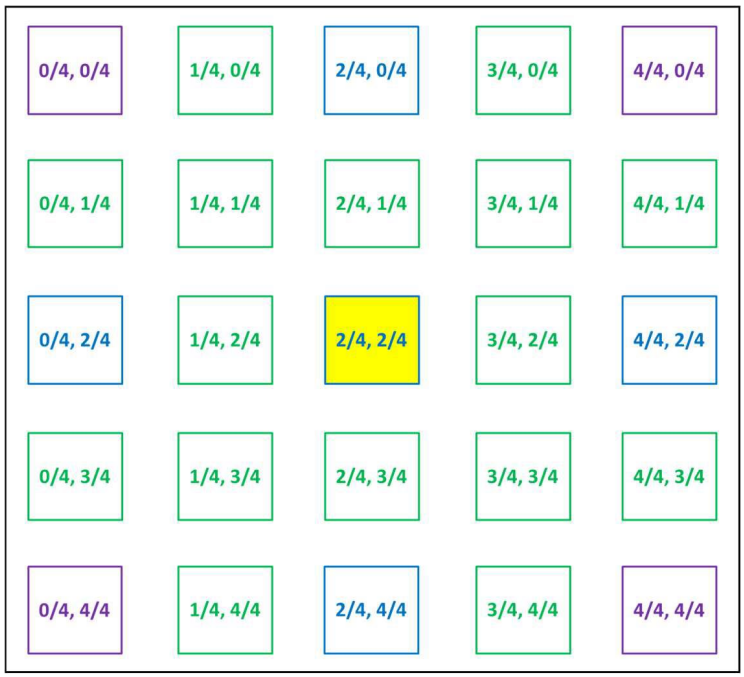
245. The ’273 patent confirms that TML6 discloses the truncation, as claimed. EX-1001, 11:46-63 (“A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024.”). The preferred embodiment of the ’273 patent also discloses operation in this manner. *Id.,* 45:3-43.

x) Dependent Claim 28: “A method according to claim 21, comprising interpolating sub-pixel values for sub-pixels having co-ordinates with even values of K and L, using a weighted sum comprising first intermediate values for sub-pixels having co-ordinates with K equal to zero and L equal to an even value and corresponding first intermediate values for sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions to produce a second intermediate value, said second intermediate value having a dynamic range equal to the dynamic

range of the first intermediate values used in said weighted sum, multiplied by a value equal to the sum of the respective weights used in said weighted sum"

246. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 21. TML6 and Fandrianto combined also discloses or renders obvious the limitations of claim 28.

247. As I discuss above in my assessment of Element 1[c] and claim 25, TML6 and Fandrianto combined discloses or renders obvious that the highlighted 2/4,2/4 sub-pixels (**blue**) (below) have co-ordinates with even values of K and L ($K/2^N$, $L/2^N$, where K=L=N=2). These highlighted sub-pixels are also interpolated as I discuss above in my assessment of claim 25, except that the values for the 0/4,2/4 sub-pixels (**blue**) and corresponding values for sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions are used instead of values for the 2/4,0/4 sub-pixels (**blue**) and corresponding values for sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions.



(2/4,2/4)
K=2, L=2

Element 1[c], Claim 25, *supra*. Therefore, consistent with my assessment above regarding values for 2/4,0/4 sub-pixels in the context of claim 25, the combination of TML6 and Fandrianto discloses or renders obvious interpolating the 2/4,2/4 sub-pixels as claimed: using a weighted sum comprising first intermediate values for sub-pixels having co-ordinates with K equal to zero and L equal to an even value (*i.e.,* $K/2^N$,$L/2^N$, where K=0,L=2, N=4) and corresponding first intermediate values for sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions to produce a second intermediate value having a dynamic range equal to the dynamic range of the first intermediate values used in

said weighted sum, multiplied by a value equal to the sum of the respective weights used in said weighted sum.

y) Dependent Claim 29: "A method according to claim 28, comprising truncating the second intermediate value by mathematically dividing the second intermediate value by a second scale factor, said second scale factor having a value equal to the sum of the respective weights used in the weighted sum of claim 21 multiplied by the sum of the respective weights used in the weighted sum of claim 28, thereby forming a sub-pixel value with a dynamic range equal to the dynamic range of said pixels"

248. As I discuss above, TML6 and Fandrianto combined discloses or renders obvious claim 28. TML6 and Fandrianto combined also discloses or renders obvious the limitations of claim 29.

249. As I discuss above in my assessment of claim 26, TML6 discloses "truncating the second intermediate value by mathematically dividing the second intermediate value by a second scale factor, said second scale factor having a value equal to the sum of the respective weights used in the weighted sum of claim 21 multiplied by the sum of the respective weights used in the weighted sum [to interpolate the 2/4,2/4 sub-pixels], thereby forming a sub-pixel value with a dynamic range equal to the dynamic range of said pixels." Applying reasoning similar to my analysis of claims 26 and 28 and Element 1[c], the combination of TML6 and Fandrianto discloses the same use of 0/4,2/4 sub-pixels to interpolate

the 2/4,2/4 sub-pixel as recited in claim 28. *See* Element 1[c], Claim 26, Claim 28, *supra*. Thus, TML6 and Fandrianto combined discloses or renders obvious claim 29. The preferred embodiment of the ’273 patent also discloses operation in this manner. *Id.,* 45:3-43.

3. Independent Claim 31

Element 31[pre][a]: “A method for quarter resolution sub-pixel value interpolation to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,”

Element 31[pre][b]: “pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation K/4, L/4, K and L being positive integers having respective values between zero and 4, the method comprising causing an apparatus to:”

Element 31[a]: “interpolate a sub-pixel value for a sub-pixel having co-ordinates with K and L equal to 1 or 3, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates 2/4, 2/4, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with K and L equal to zero, 2 or 4, situated within a quadrant of the rectangular bounded region, the quadrant being defined by

the sub-pixel having co-ordinates 2/4, 2/4 and the nearest neighbouring pixel;"

Element 31[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to 2 and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to 2, used in the interpolation of the sub-pixels having co-ordinates with K and L equal to 1 or 3, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 31[c]: "interpolate a sub-pixel value for the sub-pixel having co-ordinates with K and L equal to 2, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with K and L equal to 1 or 3, using a predetermined choice of either a weighted sum of the values of the sub-pixel having co-ordinates with K equal to 2 and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the value of the sub-pixel having co-ordinates with K equal to zero and L equal to 2 and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

250. Upon comparing claim 1 and claim 31, claim 31 is substantially the same as claim 1, but is directed to the specific case in claim 1 where N=2. As I explain above in my assessment of Elements 1[pre][a]-1[c] and claim 19, the combined teachings of TML6 and Fandrianto disclose the specific case where N=2. Accordingly, TML6 and Fandrianto, in combination, disclose or render

obvious Elements 31[pre][a]-31[c] for similar reasons to those I have explained above for Elements 1[pre][a]-1[c], Claim 19.

4. Claims 33, 36-49, 51, 53-59, and 61-62

a) Independent Claim 33

Element 33[pre][a]: "An interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,

Element 33[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 33[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel

having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 33[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 33[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

251. Elements 33[pre][a]-33[c] are directed to an "interpolator" instead of a method of interpolation but have very similar scope to the method set forth in Elements 1[pre][a]-1[c]. Thus, the combination of TML6 and Fandrianto discloses or renders obvious these elements for the same reasons that I have stated above for Element 1[pre][a]-1[c].

252. I understand that the term "interpolator" is part of the preamble and therefore may not be limiting. However, if the Board determines that "interpolator" is limiting in some way, as I explain above for Elements 1[pre][a]-1[c], both TML6 and Fandrianto disclose interpolating sub-pixels from other pixels and/or previously interpolated sub-pixels and are "interpolators." For example, a POSITA would have understood that TML6 discloses reference code that is created to execute on a processor that would execute the code to act as an interpolator and interpolate sub-pixels. See Fandrianto, 1:61-67 ("effective video coding requires the intimate integration of …. integrated circuit technology, and digital storage media…."). And the '273 patent explains that TML6 has such circuitry. EX-1001, 12:65-13:18 ("TML6… requires more silicon area in ASICs and requires more computations in some CPUs"). TML6 explicitly or implicitly, thus, discloses an interpolator to perform the interpolations in the ***oneforthpix()*** function, as set forth in Elements 33[pre][a]-33[c].

253. Fandrianto also sets forth an apparatus having circuitry operable to interpolate sub-pixels, as shown below, and is thus, an interpolator. Fandrianto describes arithmetic logic unit (ALU) 154 within "illustrative programmable architecture 100" that receives pixel data and performs pixel averaging arithmetic to generate intermediate pixel values during half-pixel and quarter-pixel interpolation processes. These interpolation operations involve averaging pixel

groups retrieved from memory and storing the resulting interpolated values, demonstrating the functionality of an interpolator.

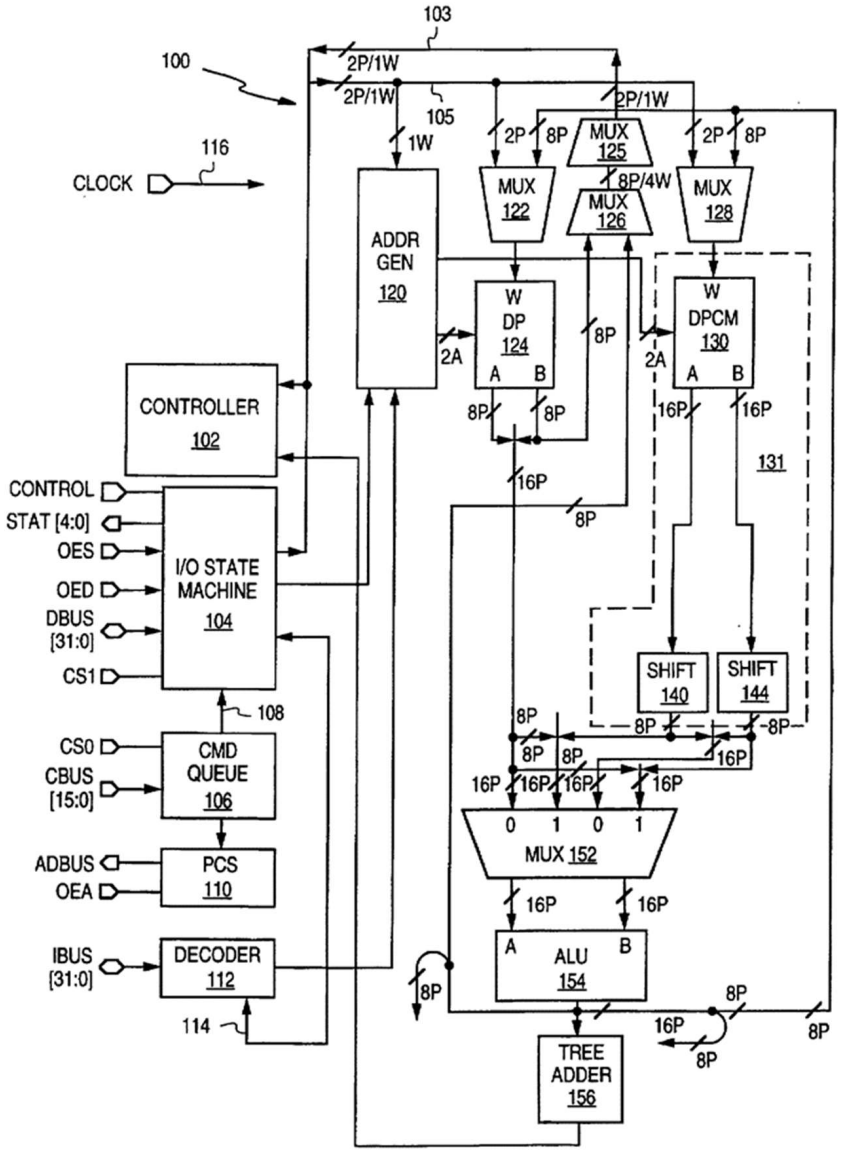


FIG. 2

Fandrianto, Figures 2-5, 1:61-67, 2:48-58, 3:55-4:53, 4:56-64, 5:26-7:2, 7:27-10:58 ("[v]arious interpolation techniques may be used…[t]he following [interpolation] technique is suitable for the architecture of FIG. 2").

254. The combined teachings of TML6 and Fandrianto, therefore, disclose an apparatus that includes an interpolator to interpolate sub-pixels as claimed.

b) Dependent Claim 36: "An interpolator according to claim 33, wherein the interpolator is configured to use a first and a second weight in the weighted average chosen to interpolate the sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, wherein when the chosen weighted average is that involving the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, the interpolator is configured to select the relative magnitudes of the first and second weights to be inversely proportional to the respective straight-line diagonal distances of the sub-pixels having co-ordinates with even values of K and L to the sub-pixel having co-ordinates with odd values of K and L whose value is being interpolated"

c) Dependent Claim 37: "An interpolator according to claim 36, wherein the interpolator is configured to use first and second weights with equal values when the sub-pixels having co-ordinates with even values of K and L are equidistant from the sub-pixel having co-ordinates with odd values of K and L whose value is being interpolated"

255. As I explained above, claim 33 is directed to an "interpolator" instead of a method of interpolation as recited in claim 1. Claims 36 and 37 directly or indirectly depend on claim 33, are likewise directed to an "interpolator." These claims have very similar scope to method claims 4, 5, and 13-17, respectively, which directly or indirectly depend on claim 1 and which I analyzed above. Accordingly, the combined teachings of TML6 and Fandrianto disclose or render

obvious claims 36, 37, and 45-49 for the reasons that I have stated above for claims 4, 5, and 13-17. Elements 1[pre][a]-1[c], 33[pre][a]-33[c]; Claims 4-5, 13-17, *supra*.

d) Dependent Claim 38: An interpolator according to claim 33, wherein interpolator is configured to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions when the value of a sub-pixel having co-ordinates with K equal to an even value and L equal to an odd value is also required"

e) Dependent Claim 39: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions when the value of a sub-pixel having co-ordinates with K equal to an odd value and L equal to an even value is also required"

f) Dependent Claim 40: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an odd value O and L equal to an even value E, not including zero or 2^N, by taking

an average of the value of a first sub-pixel having co-ordinates with K equal to the even value O-1 and L equal to E and a second sub-pixel having co-ordinates with K equal to the even value O + 1 and L equal to E"

g) Dependent Claim 41: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 1 and L equal to zero, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to zero, and a sub-pixel having co-ordinates with K equal to 2 and L equal to zero"

h) Dependent Claim 42: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N-1 and L equal to zero, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to zero, and a sub-pixel having co-ordinates with K equal to 2^N-2 and L equal to zero"

i) Dependent Claim 43: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 1 and L equal to 2^N, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to 2^N, and a sub-pixel having co-ordinates with K equal to 2 and L equal to 2^N"

j) Dependent Claim 44: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N -1 and L equal to 2^N, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to 2, and a

sub-pixel having co-ordinates with K equal to 2^N-2 and L equal to 2^N"

256. Claims 38-44 are directed to an "interpolator" instead of a method of interpolation but have very similar scope to the methods set forth in claims 6-12, respectively. Thus, the combination of TML6 and Fandrianto discloses or renders obvious claims 38-44 for the reasons discussed above for claims 6-12, respectively. Claims 6-12, *supra*.

k) Dependent Claim 45: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value E, not including zero or 2^N, and L equal to an odd value O, by taking an average of the value of a first sub-pixel having co-ordinates with K equal to E and L equal to the even value O–1 and a second sub-pixel having co-ordinates with K equal to E and L equal to the even value O+1"

l) Dependent Claim 46: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to zero and L equal to 1, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to zero, and a sub-pixel having co-ordinates with K equal to zero and L equal to 2"

m) Dependent Claim 47: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N and L equal to 1, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to zero, and

a sub-pixel having co-ordinates with K equal to 2^N and L equal to 2"

n) **Dependent Claim 48: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to zero and L equal to 2^N-1, by taking an average of the value of a pixel having co-ordinates with K equal to zero and L equal to 2^N, and a sub-pixel having co-ordinates with K equal to zero and L equal to 2^N-2"**

o) **Dependent Claim 49: "An interpolator according to claim 33, wherein the interpolator is configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with K equal to 2^N and L equal to 2^N-1, by taking an average of the value of a pixel having co-ordinates with K equal to 2^N and L equal to 2^N, and a sub-pixel having co-ordinates with K equal to 2^N and L equal to 2^N-2"**

257. As I explained above, claim 33 is directed to an "interpolator" instead of a method of interpolation as recited in claim 1. Claims 45-49 directly or indirectly depend on claim 33, are likewise directed to an "interpolator." These claims have very similar scope to method claims 4, 5, and 13-17, respectively, which directly or indirectly depend on claim 1 and which I analyzed above. Accordingly, the combined teachings of TML6 and Fandrianto disclose or render obvious claims 36, 37, and 45-49 for the reasons that I have stated above for claims 4, 5, and 13-17. Elements 1[pre][a]-1[c], 33[pre][a]-33[c]; Claims 4-5, 13-17, *supra*.

p) **Dependent Claim 53: "An interpolator according to claim 33, wherein the pixels have a predetermined dynamic range, said predetermined dynamic range corresponding to the range of values said pixels can take"**

q) **Dependent Claim 54: "An interpolator according to claim 53, wherein the interpolator is configured to interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and for sub-pixels having co-ordinates with K equal to zero and L equal to an even value, using weighted sums of the values of pixels located in rows and columns respectively, to produce a first intermediate value, said first intermediate value having a dynamic range equal to the predetermined dynamic range of the pixels used in said weighted sum multiplied by a value equal to the sum of the respective weights used in said weighted sum"**

r) **Dependent Claim 55: "An interpolator according to claim 54, wherein the dynamic range of said first intermediate value is defined by the number of bits used to represent the respective pixels used in said weighted sum plus the number of bits required to represent the sum of the respective weights used in said weighted sum"**

s) **Dependent Claim 56: "An interpolator according to claim 54, wherein the interpolator is configured to truncate the first intermediate value by mathematically dividing the first intermediate value by a first scale factor, said first scale factor having a value equal to the sum of the respective weights used in said weighted sum, thereby forming a sub-pixel value with a dynamic range equal to the predetermined dynamic range of said pixels"**

t) **Dependent Claim 57: "An interpolator according to claim 56, wherein the interpolator is configured to use**

a truncated first intermediate sub-pixel value when interpolating a value for a sub-pixel having co-ordinates with odd values of K and L"

u) **Dependent Claim 58: "An interpolator according to claim 54, wherein the interpolator is configured to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, using a weighted sum comprising first intermediate values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and corresponding first intermediate values for sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, to produce a second intermediate value, said second intermediate value having a dynamic range equal to the dynamic range of the first intermediate values used in said weighted sum, multiplied by a value equal to the sum of the respective weights used in said weighted sum"**

v) **Dependent Claim 59: "An interpolator according to claim 58, wherein the interpolator is configured to truncate the second intermediate value by mathematically dividing the second intermediate value by a second scale factor, said second scale factor having a value equal to the sum of the respective weights used in the weighted sum of claim 54 multiplied by the sum of the respective weights used in the weighted sum of claim 58, thereby forming a sub-pixel value with a dynamic range equal to the predetermined dynamic range of said pixels"**

w) **Dependent Claim 61: "An interpolator according to claim 54, wherein the interpolator is configured to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, using a weighted sum comprising first intermediate values for sub-pixels having co-ordinates with K equal to zero and L equal to an even value and corresponding first intermediate values for sub-pixels having**

corresponding co-ordinates in immediately adjacent rectangular bounded regions to produce a second intermediate value, said second intermediate value having a dynamic range equal to the dynamic range of the first intermediate values used in said weighted sum, multiplied by a value equal to the sum of the respective weights used in said weighted sum"

x) Dependent Claim 62: "An interpolator according to claim 61, wherein the interpolator is configured to truncate the second intermediate value by mathematically dividing the second intermediate value by a second scale factor, said second scale factor having a value equal to the sum of the respective weights used in the weighted sum of claim 54 multiplied by the sum of the respective weights used in the weighted sum of claim 61, thereby forming a sub-pixel value with a dynamic range equal to the predetermined dynamic range of said pixels"

258. Claims 53-59 and 61-62 are directed to an "interpolator" instead of a method of interpolation but have very similar scope to the methods set forth in claims 20-26 and 28-29, respectively. Thus, TML6 and Fandrianto combined discloses or renders obvious claims 53-59 and 61-62 for the same reasons I discuss above for claims 20-26 and 28-29, respectively. See Claims 20-26, 28-29, *supra*.

y) Dependent Claim 51: "An interpolator according to claim 33, wherein N is set equal to 2"

259. As I have previously explained, TML6 and Fandrianto disclose the specific case where N=2. Claims 1, 31-33, *supra*.

5. Claims 64-67

a) Independent Claims 64

Element 64[pre][a]: "A video encoder comprising an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 64[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 64[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 64[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to

> **an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"**
>
> **Element 64[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions"**

260. Claim 64 is substantively the same as Claim 1, except it is written in terms of "[a] video encoder comprising an interpolator…configured to" perform the method steps of Claim 1. Thus, the combination of TML6 and Fandrianto discloses Elements 64[pre][a]-64[c] for the reasons discussed for Claim 1. Claim 1, *supra*.

261. I understand that the term "video encoder" is part of the preamble and therefore may not be limiting. However, if the Board determines that this term is

limiting in some way, a POSITA would have understood that the source code disclosed in TML6 is designed to run on a processor or be implemented in hardware as a video encoder and decoder for interpolating sub-pixels. For example, in the "***lencod***" folder within TML6, the file "***lencod.c***" in the "***src***" sub-folder discloses that TML6 is used for encoding video data. TML6, 69-70 (/tml6pre.zip/lencod/src/lencod.c, ll. 3 ("Lencod.c H.26L encoder project main"), 86 ("encode one I- or P-frame")), 1 (readme.txt, l. 3) ("Encoder:"); Element 75[a] *supra*. In addition, in the "***ldecod***" folder of TML6, the file "***ldecod.c***" in the "***src***" sub-folder discloses using TML6 for decoding video data. TML6, 155 (/tml6pre.zip/ldecod/src/ldecod.c, l. 5 ("TML decoder project main ()"), 1 (readme.txt, l. 18 ("Decoder:")). *See also* ¶¶ 106-108, *supra*. Furthermore, set forth in VCEG-L45, TML6 was prepared by the VCEG of Study Group 16 of ITU Telecommunications Standardization Sector. VCEG-L45, 1. As is also outlined by Dr. Stephan Wenger, VCEG considered the ITU's Question 6 ("Q.6") and Question 15 ("Q.15"), and both questions were well known to a POSITA to address standardization of video coding to be implemented as hardware or software coders in services such as videoconferencing and video broadcast devices. *Id*. Wenger, ¶¶11-14.

262. In addition, it would have been obvious to a POSITA to employ the same interpolation method in both an encoder and a decoder for interpolating

sub-pixels. Specifically, a POSITA would have understood that using the same encoding and decoding method for a video signal would have been the easiest and most efficient way to design an encoder and a decoder. For instance, designing an encoder and decoder to interpolate sub-pixels in the same way would reduce the amount of time and resources necessary to (1) design and program the technique, (2) debug and error-correct the technique, and (3) implement it in an encoder and decoder. Also, once the technique was designed and incorporated into an encoder, a POSITA would have been able to implement the same technique in a decoder in a predictable way with a reasonable expectation of success.

263. Lastly, a POSITA would have recognized that the videoconferencing or video telephony applications that use video coding and decoding would receive packetized data over IP networks, and would require communications terminals. VCEG-L45, §8. Fandrianto's interpolation technique in combination with TML6 is designed for digital television broadcast and video telephony systems, and such systems also require communications terminals. Fandrianto, 1:12-41. A POSITA would have further understood that TML6 and Fandrianto both disclose "a video coder...comprising an interpolator" for the reasons discussed for Claim 33. Thus, a POSITA would have understood that TML6 and Fandrianto disclose or render obvious a "video encoder."

b) Dependent Claim 65

Element 65[pre][a]: "A codec comprising a video encoder according to claim 64 and a video decoder comprising an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,

Element 65[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 65[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 65[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 65[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions"

264. Claim 65 is directed to "[a] codec comprising a video encoder according to Claim 64 and a video decoder comprising an interpolator…configured to" perform the method steps of Claim 1. Thus, TML6 and Fandrianto disclose or render obvious a "codec" comprising a video encoder and decoder. Claim 1, *supra*.

265. I understand that the terms "codec," "video encoder," and "video decoder" are part of the preamble and therefore may not be limiting. However, if the Board determines that either of these terms is limiting in some way, a POSITA would have understood that the source code disclosed in TML6 is designed to run on a processor or be implemented in hardware as a video encoder and decoder for interpolating sub-pixels. For example, in the "***lencod***" folder within TML6, the file "***lencod.c***" in the "***src***" sub-folder discloses that TML6 is used for encoding video data. TML6, 69-70 (/tml6pre.zip/lencod/src/lencod.c, ll. 3 ("Lencod.c H.26L encoder project main"), 86 ("encode one I- or P-frame")), 1 (readme.txt, l. 3) ("Encoder:"); Element 75[a] *supra*. In addition, in the "***ldecod***" folder of TML6, the file "***ldecod.c***" in the "***src***" sub-folder discloses using TML6 for decoding video data. TML6, 155 (/tml6pre.zip/ldecod/src/ldecod.c, l. 5 ("TML decoder project main ()"), 1 (readme.txt, l. 18 ("Decoder:")). *See also* ¶¶ 106-108, *supra*. Furthermore, set forth in VCEG-L45, TML6 was prepared by the VCEG of Study Group 16 of ITU Telecommunications Standardization Sector. VCEG-L45, 1. As is also outlined by Dr. Stephan Wenger, VCEG considered the ITU's Question 6 ("Q.6") and Question 15 ("Q.15"), and both questions were well known to a POSITA to address standardization of video coding to be implemented as hardware or software coders in services such as videoconferencing and video broadcast devices. *Id*. Wenger, ¶¶11-14.

266. Thus, a POSITA would have understood that TML6 either discloses or that it would have been obvious to use TML6 in a video coder that includes both a video encoder and decoder. In addition, the term "codec" refers to a device or software "that performs encoding and decoding of communications protocols." CRC-Dictionary, 117; *see also* Richardson, 28 ("[a] device or program that compresses a signal is an *encoder* and a device or program that decompresses a signal is a *decoder*. An enCOder/DECoder pair is a *CODEC*"). It naturally follows to a POSITA that, since TML6 teaches a video encoder and a video encoder, it also discloses or renders obvious a codec. Claims 23-25, *supra*; *see also* VCEG-L45, §8; Fandrianto, 1:12-41.

267. In addition, it would have been obvious to a POSITA to employ the same interpolation method in both an encoder and a decoder for interpolating sub-pixels. Specifically, a POSITA would have understood that using the same encoding and decoding method for a video signal would have been the easiest and most efficient way to design an encoder and a decoder. For instance, designing an encoder and decoder to interpolate sub-pixels in the same way would reduce the amount of time and resources necessary to (1) design and program the technique, (2) debug and error-correct the technique, and (3) implement it in an encoder and decoder. Also, once the technique was designed and incorporated into an encoder,

a POSITA would have been able to implement the same technique in a decoder in a predictable way with a reasonable expectation of success.

268. Lastly, a POSITA have would recognize that the videoconferencing or video telephony applications that use video coding and decoding would receive packetized data over IP networks, and would require communications terminals. VCEG-L45, §8. Fandrianto's interpolation technique in combination with TML6 is designed for digital television broadcast and video telephony systems, and such systems also require communications terminals. Fandrianto, 1:12-41.

c) Dependent Claim 66: "A communications terminal comprising a video encoder according to claim 64"

269. TML6 in combination with Fandrianto disclose or render obvious the additional limitations of claim 66. To the extent "[a] communications terminal comprising a video encoder" is limiting, TML6 and Fandrianto disclose it. As discussed for claims 64-65, a POSITA would have understood that TML6 was developed by VCEG for videoconferencing or video telephony applications, which receive packetized data over IP networks, and would require communications terminals. VCEG-L45, §8; Claims 64-65, *supra*. In addition, Fandrianto discloses that its interpolation technique, as combined with TML6, is designed for digital television broadcast (*i.e.*, over ***wireless*** radio waves to be received by wireless terminals such as television antennae) and video telephony systems, which

likewise require communications terminals. Fandrianto, 1:12-41. A POSITA would have thus implemented the video coder/encoder/decoder of the TML6-Fandrianto combination in a "communications terminal."

d) Dependent Claim 67: "A communications terminal according to claim 66, wherein the communications terminal is a wireless terminal"

270. TML6 in combination with Fandrianto disclose or render obvious the additional limitations of claim 67. To the extent "[a] communications terminal comprising a video encoder" is limiting, TML6 and Fandrianto disclose it. As discussed for claims 64 and 65, a POSITA would have understood that TML6 was developed by VCEG for videoconferencing or video telephony applications, which receive packetized data over IP networks, and would require communications terminals. VCEG-L45, §8; Claims 64-65, *supra*. In addition, Fandrianto discloses that its interpolation technique, as combined with TML6, is designed for digital television broadcast (*i.e.*, over ***wireless*** radio waves to be received by wireless terminals such as television antennae) and video telephony systems, which likewise require communications terminals. Fandrianto, 1:12-41. A POSITA would have thus implemented the video coder/encoder/decoder of the TML6-Fandrianto combination in a "communications terminal."

6. Claims 68-70

a) Independent Claim 68

Element 68[pre][a]: "A still image encoder comprising an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 68[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 68[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 68[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to

an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 68[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

271. Claim 68 is substantively the same as Claim 64 except it is directed to a "still image encoder" rather than a "video encoder." Claim 64, *supra*.

272. Video comprises a series of still images played in rapid succession to create the illusion of motion. Video encoders compress/encode this series of still images for efficient transmission and storage. A POSITA would have thus understood that a video encoder/decoder is also a still image encoder/decoder

because it must be capable of processing each frame of a video individually as a still image.

b) Dependent Claim 69

Element 69[pre][a]: "A codec comprising a still image encoder according to claim 68 and a still image decoder comprising an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 69[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 69[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel

having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 69[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 69[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

273. Claim 69 is substantively the same as Claim 65, except it is directed to a "**still image** encoder" rather than a "**video** encoder." Thus, the TML6-Fandrianto combination renders Claim 69 obvious for the same reasons as Claim 68. Claim 68, *supra*.

c) Dependent Claim 70: "A communications terminal comprising a still image encoder according to claim 68"

274. Claim 70 is substantively the same as Claim 66, except it is directed to a "**still image** encoder" rather than a "**video** encoder." The TML6-Fandrianto combination renders Claim 70 obvious for the same reasons as Claim 69. Claim 69, *supra*.

7. Claims 71-72

a) Independent Claim 71

Element 71[pre][a]: "A video decoder comprising an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 71[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 71[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a

predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates 1/2, 1/2, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates 1/2, 1/2 and the nearest neighbouring pixel;"

Element 71[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 71[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded region."

275. Elements 71[pre][a]-71[c] are directed to a "video decoder" having an "interpolator" instead of a method of interpolation but have very similar scope to the method set forth in Elements 1[pre][a]-1[c]. Thus, the combination of TML6 and Fandrianto discloses or renders obvious these elements for the same reasons that I have stated above for Element 1[pre][a]-1[c]. *See* Elements 1[pre][a]-1[c], *supra*.

276. I understand that the terms "video decoder" and "interpolator" are part of the preamble and therefore may not be limiting. However, if the Board determines that either of these terms is limiting in some way, a POSITA would have understood that the TML6 source code is designed to run on a processor or be implemented in hardware as a video encoder and decoder that interpolate (and are thus an interpolator). *See also* Claim 64. As described by VCEG-L45, the source code in TML6 was prepared by the VCEG of Study Group 16 of ITU Telecommunications Standardization Sector. VCEG-L45, 1. As Dr. Stephan Wenger and I have explained, VCEG addresses Question 6 ("Q.6") and Question 15 ("Q.15"), which was well known to a POSITA to address standardization of video coding standards to be implemented as hardware or software coders in services such as videoconferencing and video broadcast devices. *Id*.; Wenger, ¶¶11-14. Therefore, in view of VCEG-L45, a POSITA would have understood that TML6 discloses or that it would have been obvious to use TML6 source code in a

video decoder and interpolator. Thus TML6, Fandrianto, and VCEG-L45 combined discloses or renders obvious claim 71.

b) Dependent Claim 72: "A communications terminal comprising a video decoder [sic: encoder][20] according to claim 71."

277. As I explained above, the combination of TML6, Fandrianto, and VCEG-L45 discloses or renders obvious claims 71 and 66. Claim 72 is directed to a "communications terminal" with the video encoder recited in claim 71 and has very similar scope. Thus, the combination of TML6, Fandrianto, and VCEG-L45 discloses or renders obvious these elements for the same reasons discussed above for claims 71 and 66.

278. In addition, to the extent that "a communications terminal" is limiting, a POSITA would have understood that the code in TML6 is designed for a video coder in a communications terminal. As I discuss above in my analysis of claims 33 and 71, a POSITA would have understood that TML6 discloses or renders obvious using the interpolation method in a communications terminal having a

[20] Claim 72 contains an error because it recites "video ***decoder*** according to claim 71." However, since Claim 71 does not recite a "video ***decoder***," and instead recites a "video ***encoder***," Claim 72 should recite a "video encoder according to claim 71."

video encoder (as well as a video decoder) because TML6 source was developed by VCEG for videoconferencing or video telephony applications, which receive packetized data over IP networks, and thus would require communications terminals. VCEG-L45, §8. In addition, Fandrianto's interpolation technique is used in digital television broadcast and video telephony systems, which likewise include communications terminals. Fandrianto, 1:12-41.

8. Claims 73-74

a) Independent Claim 73:

Element 73[pre][a]: "A still image decoder comprising an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 73[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 73[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a

predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 73[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 73[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

279. Claim 73 is substantively the same as Claim 68, except it is written in terms of a "still image decoder" rather than a "still image encoder." Thus, the TML6-Fandrianto combination renders Claim 73 obvious for the same reasons as Claims 68-70. Claims 68-70, *supra*.

b) Dependent Claim 74: "A communications terminal comprising a still image decoder according to claim 73."

280. The TML6-Fandrianto combination renders Claim 74 obvious for the same reasons as Claims 68-72. Claims 68-72, *supra*.

9. Independent Claim 75:

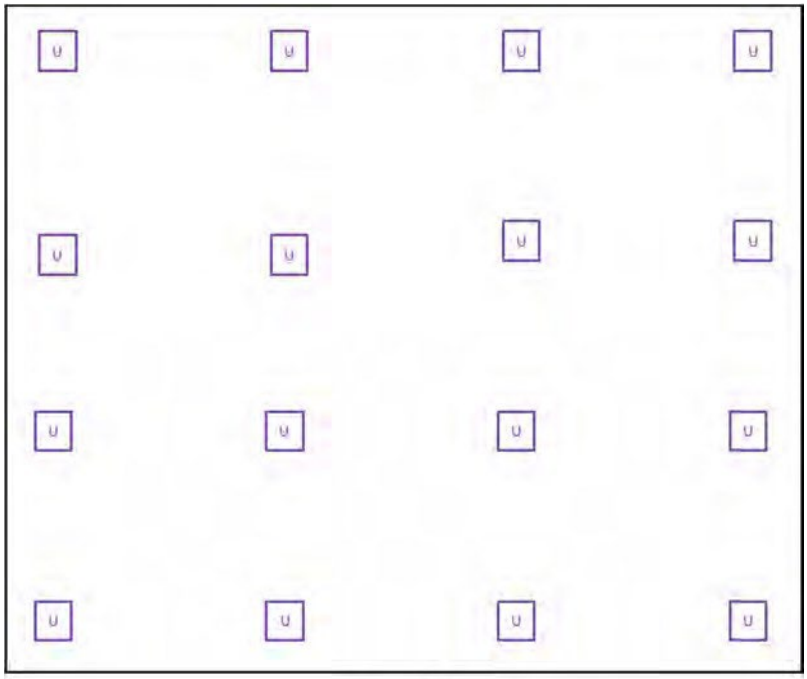
a) Element 75[pre][a]: "An interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

281. Claim 75 is substantively the same as Claim 1, except it is written in terms of "[a]n interpolator…comprising[] circuitry configured to" perform the steps of Claim 1. I understand that this limitation forms part of the preamble of this claim and therefore may not be limiting. To the extent that the Board treats Element 75[pre][a] as limiting, the prior art teaches it.

282. For example, TML6 describes an interpolator for sub-pixel value interpolation. TML6, 48 (/tml6pre.zip/lencod/src/image.c, l. 3) ("Code one

image/slice"). TML6 includes a file ***image.c*** (in the ***src*** folder of the ***encoder*** folder) containing a function ***encode_one_frame()*** that "[e]ncodes one frame" of video data. *Id*., 48-50 (/tml6pre.zip/lencod/src/image.c, ll. 33-162). This function ***encode_one_frame()*** calls the function ***oneforthpix()*** to interpolate sub-pixels for the frame of video data. *Id*., 49 (/tml6pre.zip/lencod/src/image.c, ll. 102-109), 57-60 (/tml6pre.zip/lencod/src/image.c, ll. 538-719). Furthermore, explanation of how the ***oneforthpix()*** function works applies equally to the ***oneforthpix_2()*** function, because these functions interpolate sub-pixels in the same way.

283. The ***imgY*** data structure stores data values representing the brightness—or "luminance"—of unit pixels "U" (**purple**) for a frame of video data as illustrated below.

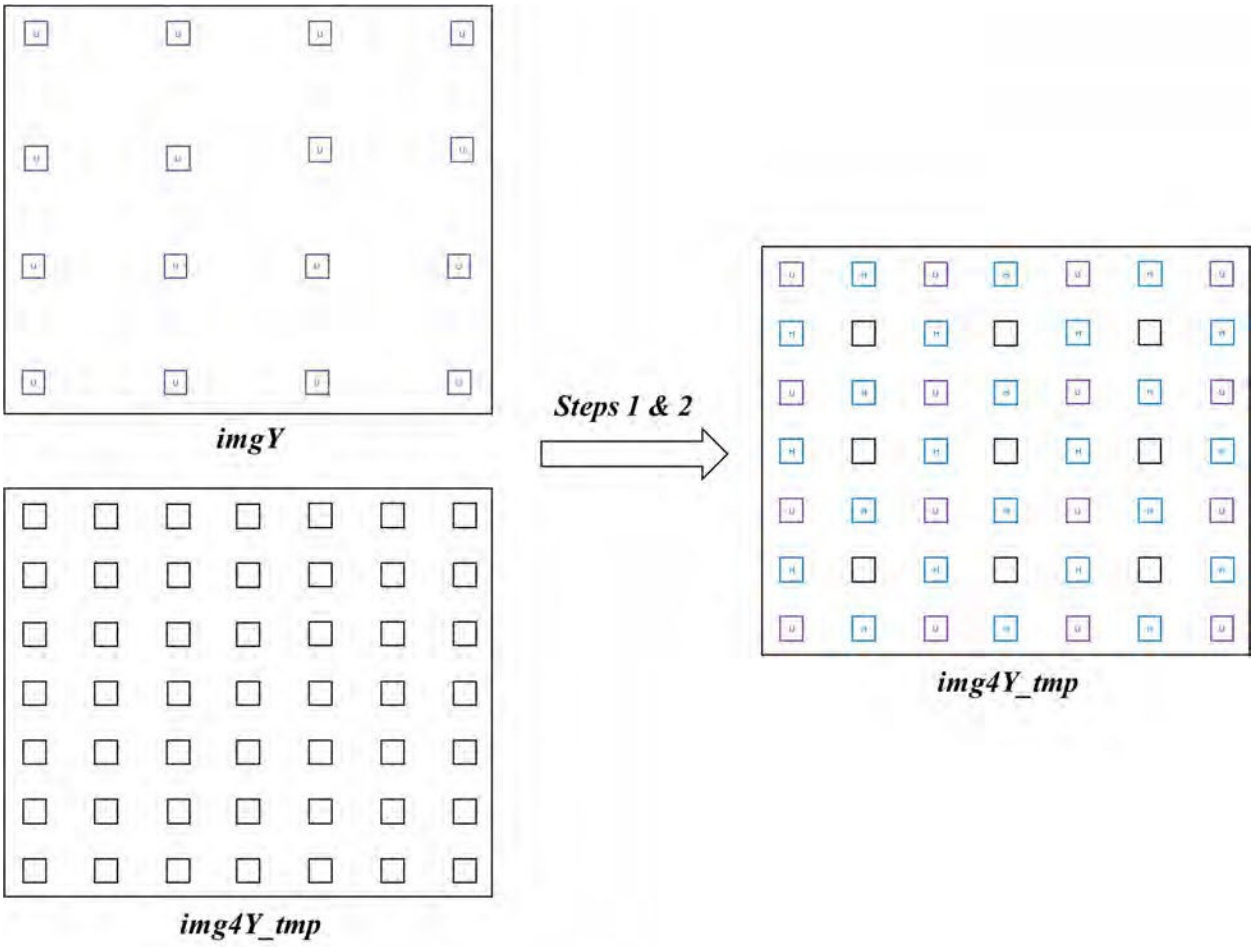


imgY

TML6, 93 (/tml6pre.zip/lencod/src/lencod.c, ll. 1338-1344), 49-50 (/tml6pre.zip/lencod/src/image.c, ll. 97-109), 110-112 (/tml6pre.zip/lencod/src/macroblock.c, ll. 480-600), 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 563-632), 131 (/tml6pre.zip/lencod/src/global.h, l. 149). Then, as I explain in further detail below for Elements 75[a]-75[c], the ***oneforthpix()*** function performs three primary steps to interpolate sub-pixel values based on the stored unit pixel values in the ***imgY*** data structure. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 553-625).

284. As I note below, in the initial two steps, the ***oneforthpix()*** function generates a temporary data structure—***img4Y_tmp***—interpolates values for some

of the ½ sub-pixels (**blue**) based on the values of the unit pixels (**purple**), and stores these values in the ***img4Y_tmp*** data structure.

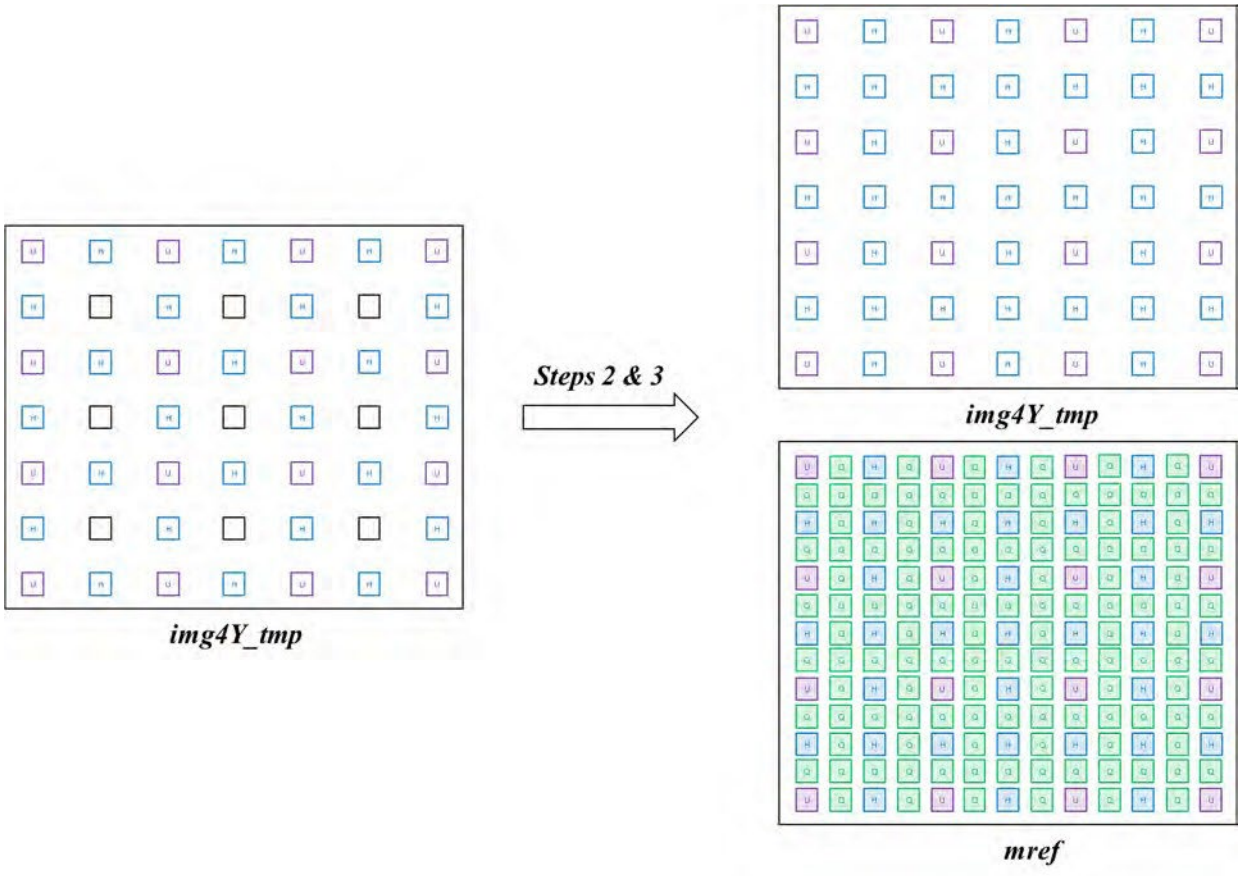


```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));      /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 563-592).

285. As I further describe below, in steps two and three, the ***oneforthpix()*** function interpolates values for additional ½ sub-pixels (**blue**) (some based on unit pixel values and others based on previously-interpolated ½ sub-pixels values) and stores these values in the data structure ***img4Y_tmp*** (below upper right). The ***oneforthpix()*** function uses the values for the ½ sub-pixels (**blue**) and unit pixels

(**purple**) in the data structure ***img4Y_tmp*** (below upper right) to interpolate values for ¼ sub-pixels (**green**) and stores the ¼ and ½ sub-pixel values (**green**) and (**blue**), in the data structure ***mref*** (below lower right).



```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2         ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

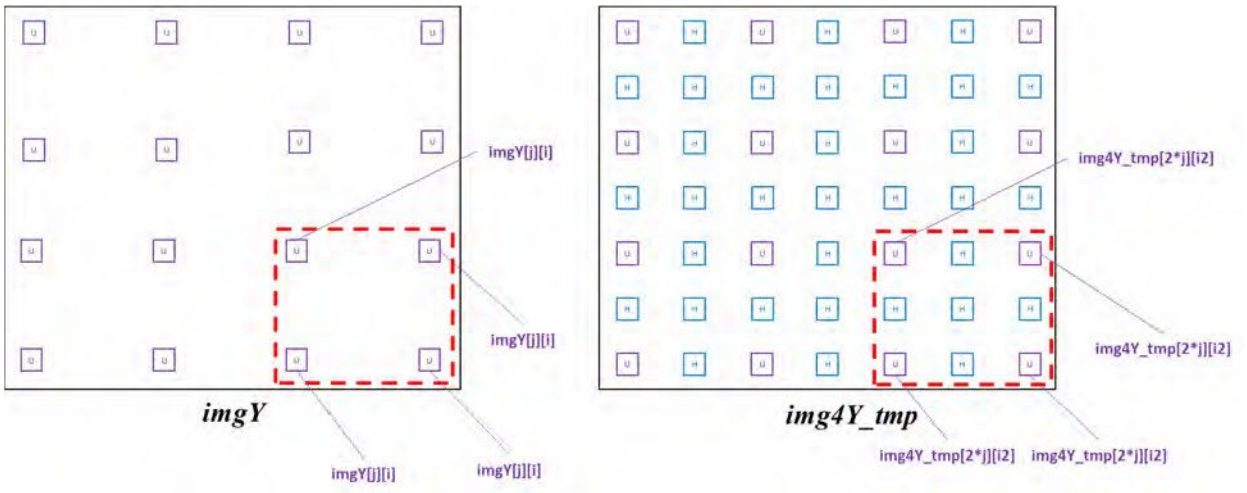
```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                          )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                          )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                  ][i/2                  ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                  ] +
612                       img4Y_tmp[j/2                  ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                          + 2048)/4096));
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-625).

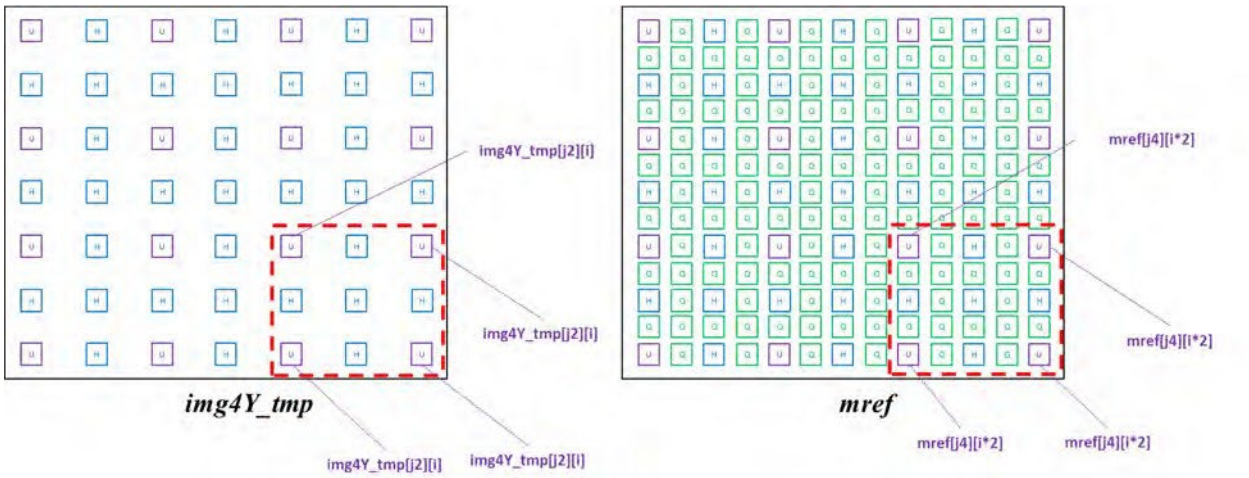
286. As I have explained above, and shown below in more detail, the data structures ***img4Y_tmp*** and ***mref*** include ½ sub-pixels (**blue**) and/or ¼ sub-pixels

(**green**) that are located within a rectangular bounded region (**red**) that is defined by four corner unit pixels (**purple**) and that has no intermediate pixels between the corners. The unit pixels (**purple**) and sub-pixels (**blue** and **green**) are, additionally, arranged in rows and columns.

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                      img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                      img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```



```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE FOURTH TAP[0][0]*(img4Y tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```



TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 554-592). The pixels are configured for display. TML6, 1-2 (/tml6pre.zip/lencod/readme.txt ("picture formats")). Similarly, if a motion vector points to a sub-pixel, the sub-pixels are also configured for display. TML6, 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632), 16-24 (/tml6pre.zip/lencod/src/block.c, ll. 734-1180 ("motion_search"

function)), 115-116 (/tml6pre.zip/lencod/src/macroblock.c, ll. 769-842 ("Motion vector search")).

287. The '273 patent explains, and thus confirms my assessment, that the TML6 source code is an interpolator for interpolating pixels and sub-pixels to determine values for the sub-pixels. EX-1001, Figures 13(a)-13(c), 8:38-9:36 ("[I]nterpolation schemes have been developed as part of the work ongoing in the ITU-Telecommunications Standardization Sector, Study Group 16, Video Coding Experts Group (VCEG), Question[] 6… These approaches were proposed for incorporation into ITU-T recommendation H.26L and have been implemented in test models (TML) for the purposes of evaluation and further development…. The test model corresponding to…. Question 6 is known as Test Model 6 (TML6)…."), 11:25-13:18 (describing TML6's interpolation of ½ and ¼ sub-pixels).

288. In its description of TML6 with reference to Figure 13(a) (reproduced below), the '273 patent further notes that TML6 interpolates ½ sub-pixels "b" and "c" (**blue**) and ¼ sub-pixels "d" to "g" (**green**) and that these sub-pixels are located within a rectangular bounded region (**red**) defined by four corner pixels "A" (**purple**) with no intermediate pixels between the corners:

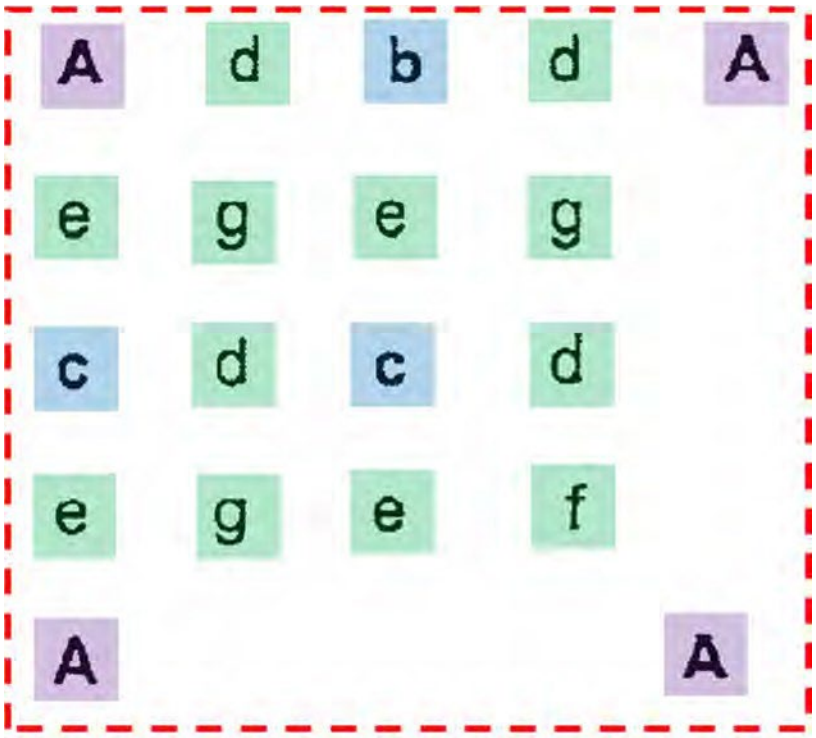


Fig. 13(a)

EX-1001, Figure 13(a), 8:38-9:27 ("[S]eparate notation, defined in FIG. 13a, will be used in the discussion of the sub-pixel value interpolation scheme used in TML6…. [T]he letter A represents the location of pixels in the image data representing a frame of a video sequence…."), 11:25-13:18 (describing TML6's interpolation of ½ sub-pixels b and c, and ¼ sub-pixels d-g using "pixels A" with reference to Figure 13(a)). The figure above illustrates that the unit pixels "A" (**purple**) and sub-pixels "b" to "g" (**blue** and **green**) are arranged in rows and columns. I agree all these things were known in the prior art.

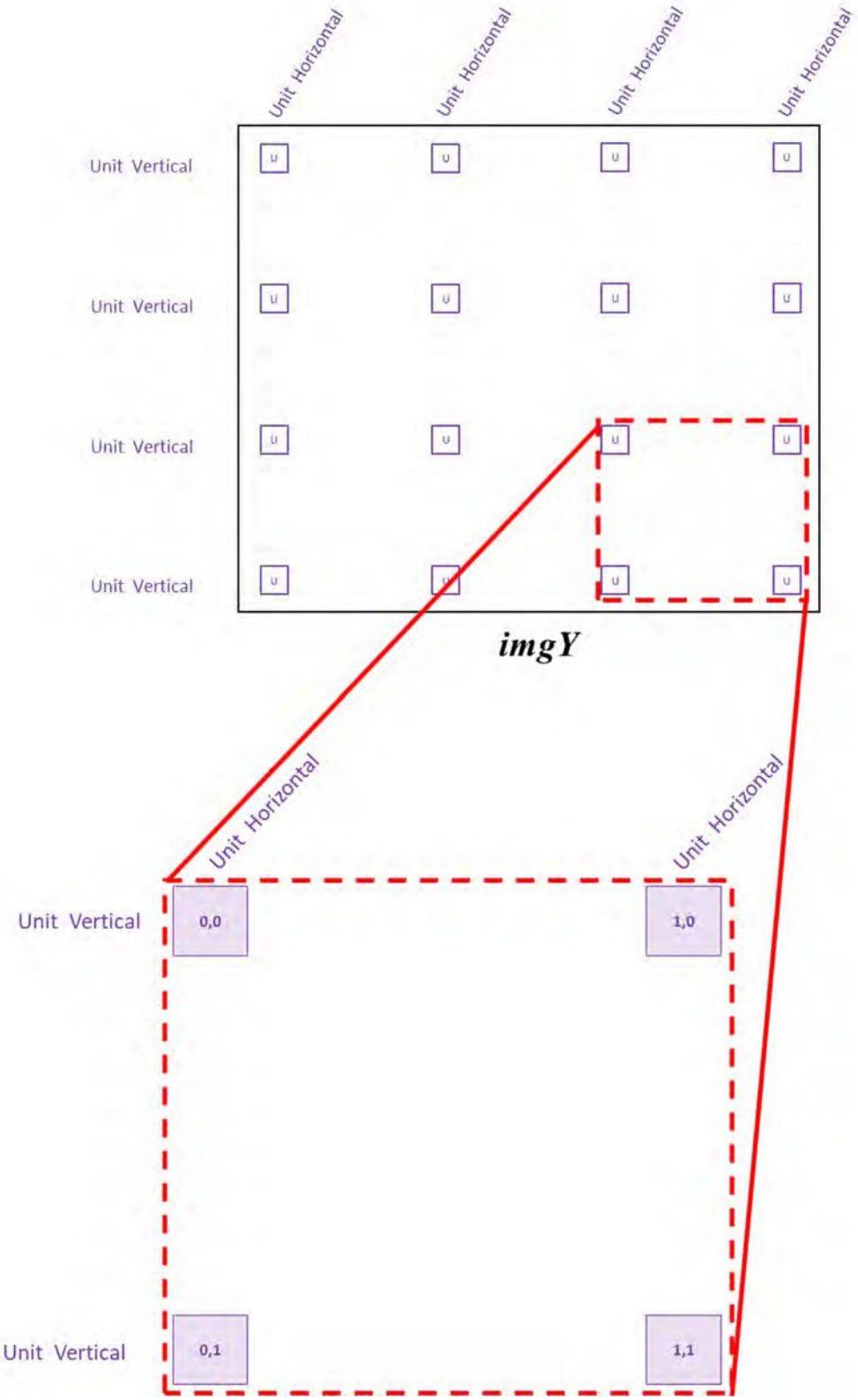
289. I understand that the term "interpolator" is part of the preamble and therefore may not be limiting. However, if the Board determines that "interpolator" is limiting in some way, TML6 and Fandrianto disclose interpolating sub-pixels from other pixels and/or previously interpolated sub-pixels and are "interpolators." Moreover, a POSITA would have understood that the TML6 source code is designed to run on a processor or be implemented in hardware such as a video encoder and decoder that interpolate sub-pixels and are thus an interpolator. Elements 75[pre][b]-75[c], *infra*.

b) Element 75[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator comprising:"

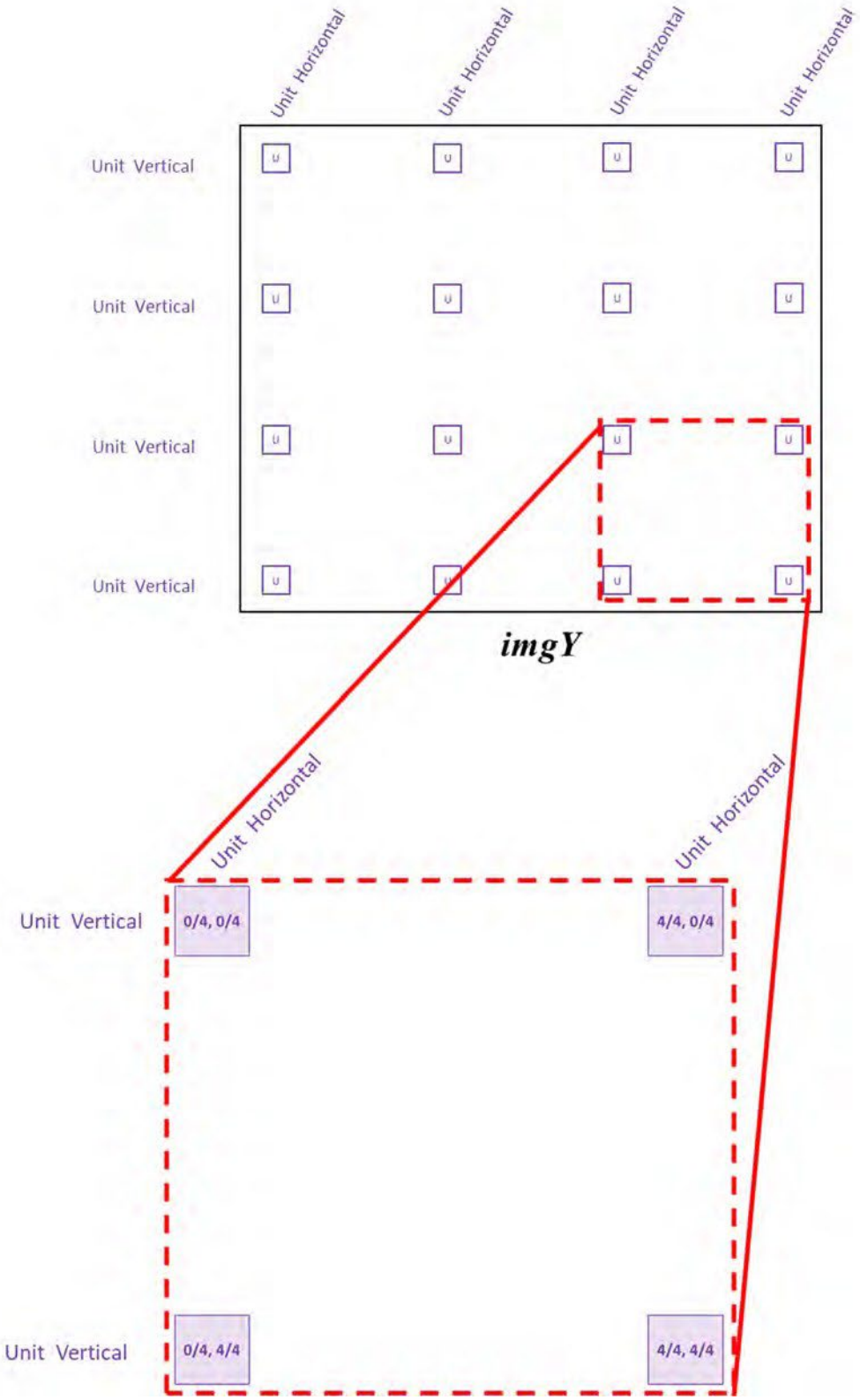
290. I understand that this limitation forms part of the preamble of this claim and therefore may not be limiting. To the extent that the Board treats Element 75[pre][b] as limiting, the prior art teaches it.

291. TML6 explains that pixel and sub-pixel locations are representable mathematically within the rectangular bounded region.

292. The following is described above for Element 75[pre][a]. The data structure ***imgY*** stores values for unit pixels U (**purple**). And, as illustrated below, four of the pixels U (**purple**) are located in the rectangular bounded region (**red**).

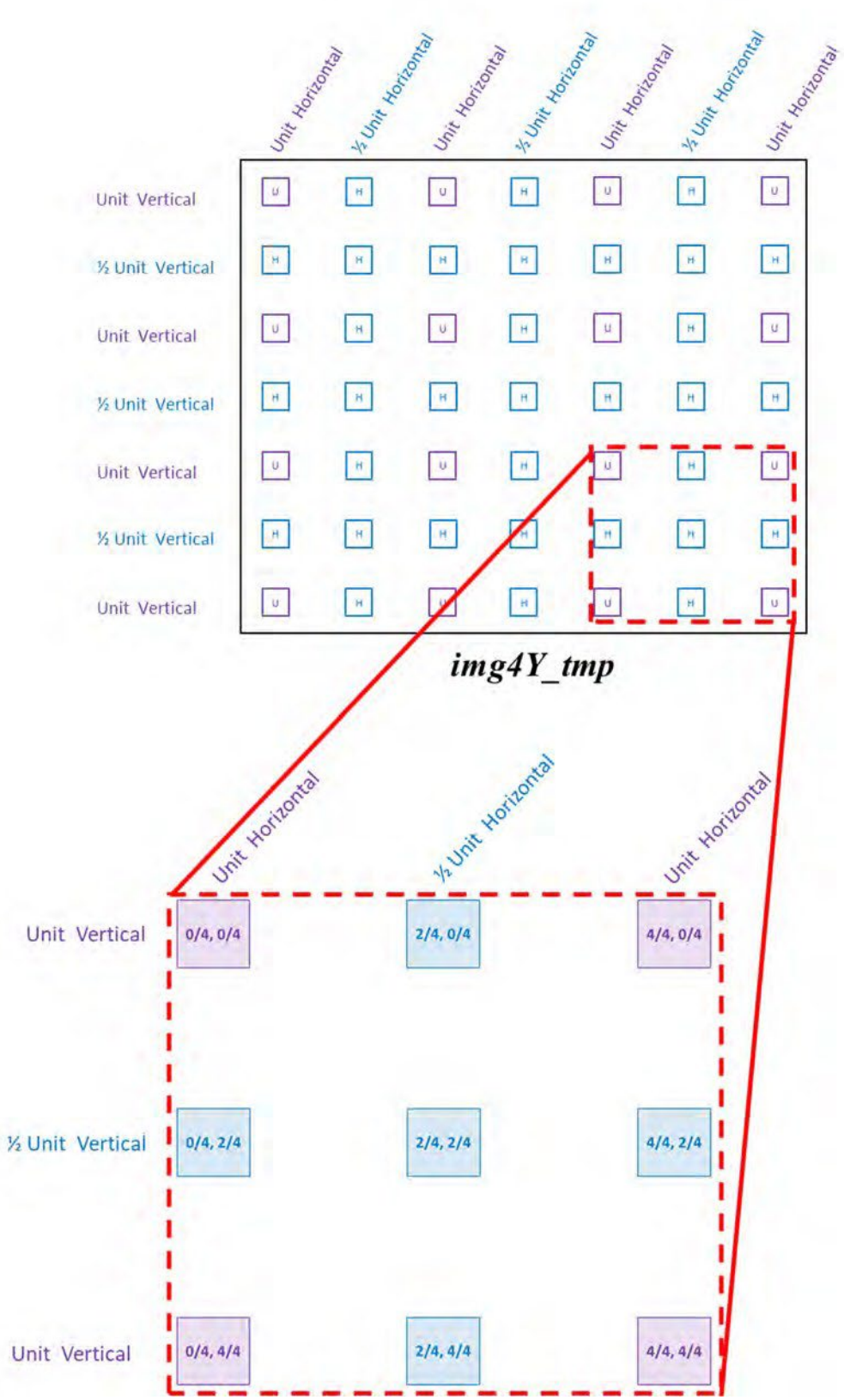

Unit Horizontal
Unit Horizontal
Unit Horizontal
Unit Horizontal
Unit Vertical
Unit Vertical
Unit Vertical
Unit Vertical
imgY
Unit Horizontal
Unit Horizontal
Unit Vertical
0,0
1,0
Unit Vertical
0,1
1,1

Within this bounded region (**red**), the four pixels U (**purple**) can be represented mathematically using the Cartesian co-ordinate system—the upper left pixel U has the co-ordinates (0,0), the upper right pixel U has the co-ordinates (1,0), the lower left pixel U has the co-ordinates (0,1), and the lower right pixel U has the co-ordinates (1,1). These co-ordinates can also be represented with the notation "0/4" (which equals 0) and "4/4" (which equals 1), as illustrated below.


Unit Horizontal
Unit Horizontal
Unit Horizontal
Unit Horizontal
Unit Vertical
Unit Vertical
Unit Vertical
Unit Vertical
imgY
Unit Horizontal
Unit Horizontal
Unit Vertical
0/4, 0/4
4/4, 0/4
Unit Vertical
0/4, 4/4
4/4, 4/4

See Element 75[pre][a], *supra*.

293. Additionally, the ***oneforthpix()*** function interpolates ½ sub-pixels (**blue**) based on the values of the unit pixels U (**purple**) in the data structure ***imgY*** and stores these pixel values in the data structure ***img4Y_tmp***. *See* Element 75[pre][a], *supra*. Since the ½ sub-pixels (**blue**) are located halfway between the unit pixels U (**purple**) in the vertical direction, or the horizontal direction, or both, they can be expressed as x,y co-ordinates 0/4, 2/4 (= 1/2), and/or 4/4 as illustrated below.


Unit Horizontal
½ Unit Horizontal
Unit Horizontal
½ Unit Horizontal
Unit Horizontal
½ Unit Horizontal
Unit Horizontal
Unit Vertical
½ Unit Vertical
Unit Vertical
½ Unit Vertical
Unit Vertical
½ Unit Vertical
Unit Vertical
U
H
img4Y_tmp
Unit Horizontal
½ Unit Horizontal
Unit Horizontal
Unit Vertical
½ Unit Vertical
Unit Vertical
0/4, 0/4
2/4, 0/4
4/4, 0/4
0/4, 2/4
2/4, 2/4
4/4, 2/4
0/4, 4/4
2/4, 4/4
4/4, 4/4

294. Likewise, the ***oneforthpix()*** function interpolates ¼ sub-pixels (**green**) based on the unit pixels U (**purple**) and the previously-interpolated ½ sub-pixels (**blue**) and stores the pixel and sub-pixel values in the data structure ***mref***, as noted above. *See* Element 75[pre][a], *supra*. Since the ¼ sub-pixels (**green**) are located a quarter-way between the unit pixels U (**purple**) in the vertical direction, or the horizontal direction, or both, they can be expressed with the x,y co-ordinates, as illustrated below.


Unit Horizontal
¼ Unit Horizontal
½ Unit Horizontal
¼ Unit Horizontal
Unit Horizontal
¼ Unit Horizontal
½ Unit Horizontal
¼ Unit Horizontal
Unit Horizontal
¼ Unit Horizontal
½ Unit Horizontal
¼ Unit Horizontal
Unit Horizontal
Unit Vertical
¼ Unit Vertical
½ Unit Vertical
¼ Unit Vertical
Unit Vertical
¼ Unit Vertical
½ Unit Vertical
¼ Unit Vertical
Unit Vertical
¼ Unit Vertical
½ Unit Vertical
¼ Unit Vertical
Unit Vertical
mref
Unit Horizontal
¼ Unit Horizontal
½ Unit Horizontal
¼ Unit Horizontal
Unit Horizontal
Unit Vertical
¼ Unit Vertical
½ Unit Vertical
¼ Unit Vertical
Unit Vertical
0/4, 0/4
1/4, 0/4
2/4, 0/4
3/4, 0/4
4/4, 0/4
0/4, 1/4
1/4, 1/4
2/4, 1/4
3/4, 1/4
4/4, 1/4
0/4, 2/4
1/4, 2/4
2/4, 2/4
3/4, 2/4
4/4, 2/4
0/4, 3/4
1/4, 3/4
2/4, 3/4
3/4, 3/4
4/4, 3/4
0/4, 4/4
1/4, 4/4
2/4, 4/4
3/4, 4/4
4/4, 4/4

See Element 75[pre][a], *supra*.

295. TML6 thus discloses, as shown above, that the locations of the pixels (**purple**) and sub-pixels (**blue** and **green**) can be represented mathematically within the rectangular bounded region (**red**) using the co-ordinate notation $K/2^N$ for the "horizontal co-ordinates" and notation $L/2^N$ for the "vertical co-ordinates." The horizontal co-ordinates include: 0/4 (K=0; N=2), 1/4 (K=1; N=2), 2/4 (K=2; N=2), 3/4 (K=3; N=2), and 4/4 (K=4; N=2). The vertical co-ordinates include: 0/4 (L=0; N=2), 1/4 (L=1; N=2), 2/4 (L=2; N=2), 3/4 (L=3; N=2), and 4/4 (L=4; N=2).

296. As further noted above, K and L are also integers having respective values between zero and 2^N (*i.e.*, 2^2=4). Although claim 75 states that K and L are "positive integers having respective values between zero and 2^N," it also states that K and L can equal zero. Elements 75[pre][b]-75[c], *infra*. Therefore, while zero is not a positive integer, for my analysis, I assume that K and L can equal zero, as set forth in the claim.

297. Furthermore, the value of "N" is equal to 2, as noted above, which is a positive integer greater than one. And TML6 discloses interpolating up to ¼ sub-pixels, and thus, discloses that that N corresponds to a particular degree of sub-pixel value interpolation (*i.e.*, interpolating ¼ ($1/2^N$; N=2) sub-pixels). Moreover, TML6 is source code that a POSITA would have understood is designed to execute on a processor, such as a GPU. *See generally*, TML6. Thus,

TML6 teaches causing an apparatus to perform the interpolating steps, as discussed below with respect to Elements 1[a]-1[c], because it would cause the processor to execute the commands that perform the interpolation. *Id*.

298. In its description of TML6 with reference to Figure 13(a) (reproduced below), the '273 patent explains that, in TML6, the locations of the pixels (**purple**), ½ sub-pixels (**blue**), and ¼ sub-pixels (**green**) can be represented mathematically within the rectangular bounded region (**red**) using the co-ordinates $K/2^N$, $L/2^N$. This is illustrated below. K and L are integers that have values between zero and four (i.e., 2^N; N=2), and N equals two, which is an integer greater than one and that represents a particular degree of sub-pixel value interpolation (i.e., interpolating ¼ ($1/2^N$; N=2) sub-pixels).

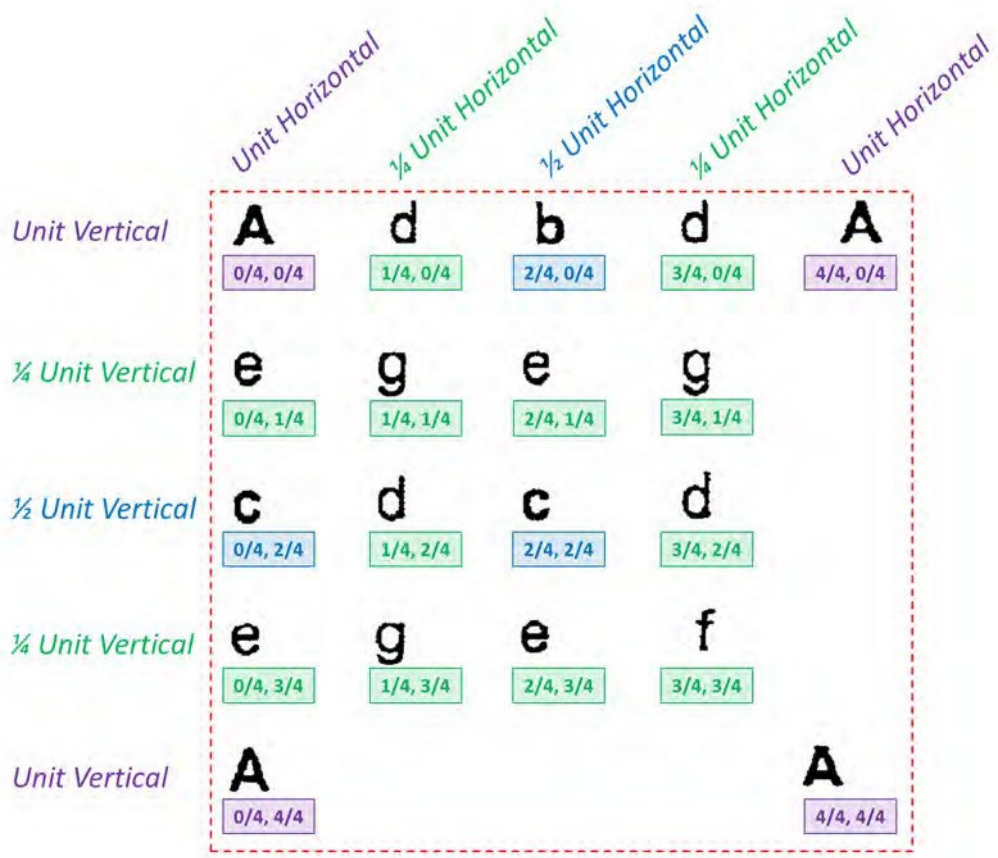


Fig. 13(a)

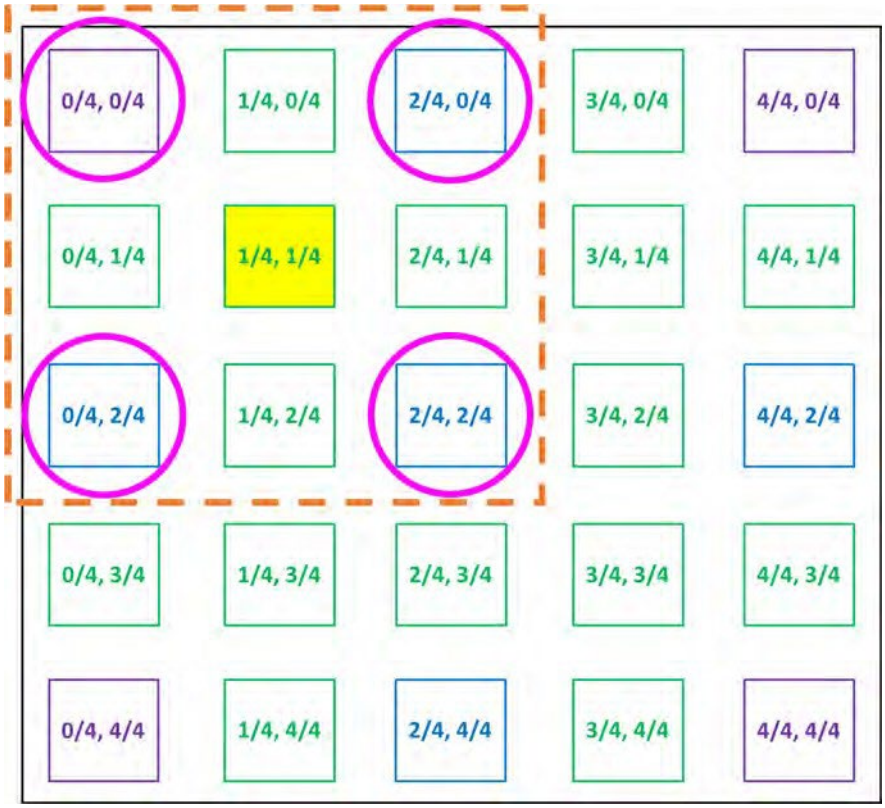
EX-1001, Figures 13(a)-13(c), 8:38-9:53 ("A separate notation, defined in FIG. 13a, will be used in the discussion of the sub-pixel value interpolation scheme used in TML6…. [T]he letter A represents the location of pixels in the image data representing a frame of a video sequence…. The term unit 'horizontal location" is used to describe the location of any sub-pixel that is constructed in a column of the original image data….'[H]alf horizontal location' is used to describe the location of any sub-pixel that is constructed in a column that lies at half pixel resolution.

Sub-pixels b, c, and e…fall into this category….'[Q]uarter horizontal location' refers to any sub-pixel that is constructed in a column which lies at quarter-pixel resolution, such as…sub-pixels d and g in FIG. 13a….''), 11:25-13:18 (describing TML6's interpolation of ½ and ¼ sub-pixels). I agree all these things were known in the prior art.

c) Element 75[a]: "circuitry configured to interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel;"

299. TML6 and Fandrianto combined discloses or renders obvious Element 75[a]. Both prior art references disclose circuitry that is configured to interpolate a sub-pixel value with odd values of both K and L (*e.g.*, K=1; L=1 or 3). For example, TML6's ***oneforthpix()*** function interpolates the highlighted ¼ sub-pixels

(**green**)[21] located at the co-ordinates (¼,¼) (*i.e.*, $K/2^N$, $L/2^N$ (K=1, L=1, N=2)) ("¼,¼ sub-pixel"), as illustrated below:



[21] The illustrations in Element 75[a] and elsewhere in my declaration show sub-pixels within one rectangular bounded region of TML6 and Fandrianto. However, corresponding sub-pixels in other rectangular bounded regions are interpolated in the same way. TML6, 39 (/tml6pre.zip/lencod/src/lencod.c, ll. 1338-1344), 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632); Fandrianto, Figure 5, 5:26-32, 8:58-9:32, 10:51-58.

TML6, 58 (/tml6pre.zip/lencod/src/image.c ll. 593-625); Element 75[pre][a], *supra*. In my analysis that follow, the pixels and sub-pixels positioned at co-ordinates (x,y) are referred to as "x,y pixels" and "x,y sub-pixels."

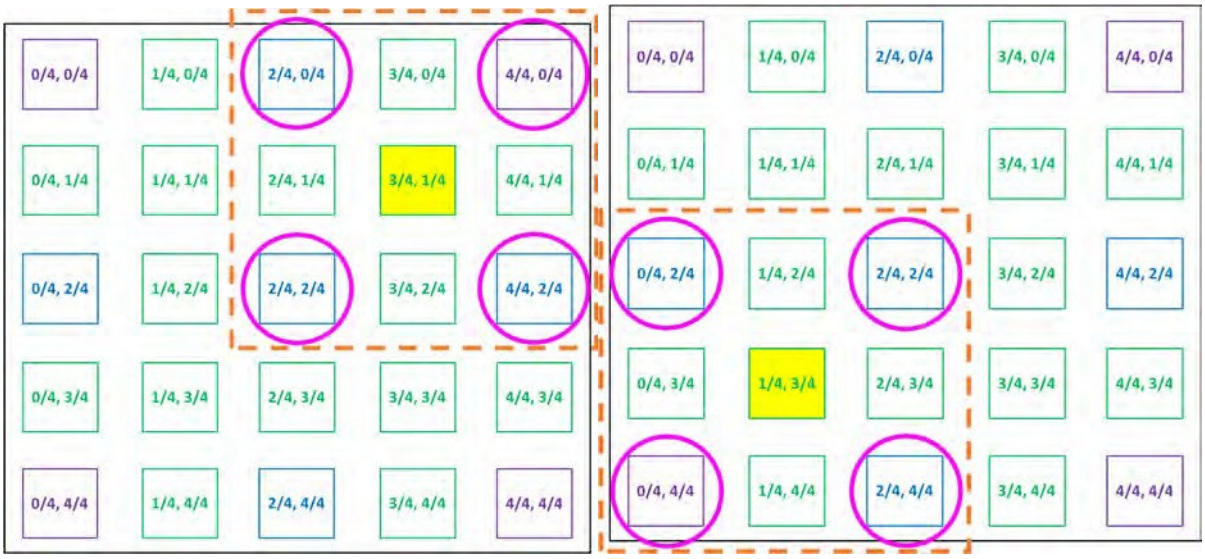
300. In the above illustration, the highlighted ¼,¼ sub-pixel (**green**) is interpolated according to a weighted average of the four values of the pixel (**purple**) and the three ½ sub-pixels (**blue**) referenced in Element 75[c] (and circled above), with weights equal to "1." Specifically, the highlighted ¼,¼ sub-pixel (**green**) is interpolated using a weighted average of the following: (1) The "nearest-neighbouring pixel" (*i.e.*, the 0/4,0/4 pixel); (2) The 2/4,2/4 sub-pixel "situated at the co-ordinates ½,½" (*i.e.*, ½,½=2/4,2/4)); (3) The 0/4,2/4 sub-pixel "having co-ordinates with even values of both K and L, including zero" (*i.e.*, co-ordinates $K/2^N$, $L/2^N$, where K=0; L=2; N=2) and "diagonally-opposed" to the 2/4,0/4 sub-pixel; and (4) The 2/4,0/4 sub-pixel "having co-ordinates with even values of both K and L, including zero" (*i.e.*, co-ordinates $K/2^N$, $L/2^N$, where K=2; L=0; N=2) and "diagonally-opposed" to the 0/4,2/4 sub-pixel.

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                    ][i/2                    ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                    ] +
612                       img4Y_tmp[j/2                    ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
```

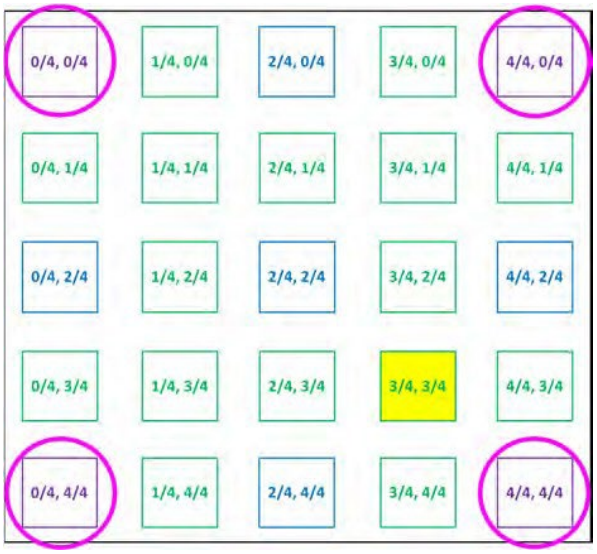
TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-625). The sub-pixels and pixel described above are located within a quadrant (**orange**) of the rectangular bounded region. This quadrant is defined by corner pixels having co-ordinates ½,½ (*i.e.*, 2/4,2/4 sub-pixel (**blue** and circled)) and the nearest-neighbouring pixel (*i.e.*, 0/4,0/4 pixel (**purple** and circled)).

301. Similarly, as illustrated below, in TML6, the ***oneforthpix()*** function interpolates the highlighted ¼,¾ sub-pixel (**green**) on the right and located at the co-ordinates (¼,¾) (*i.e.*, $K/2^N$, $L/2^N$ (K=1, L=3, N=2)), and the highlighted ¾,¼ sub-pixel (**green**) on the left and located at the co-ordinates (¾,¼) (*i.e.*, $K/2^N$, $L/2^N$ (K=3, L=1, N=2)) in the same fashion as the ¼,¼ sub-pixel, using a weighted

average of the four circled values including the nearest-neighbouring pixel (**purple**) and the three sub-pixels (**blue**). These highlighted sub-pixels (**green**), circled pixels (**purple**), and circled sub-pixels (**blue**) in the figures below likewise correspond to the "nearest-neighbouring pixel" (*i.e.*, 4/4,0/4 or 0/4,4/4 pixel), "the sub-pixel situated at the co-ordinates ½,½" (*i.e.*, 2/4,2/4 sub-pixel), and the "pair of diagonally-opposed sub-pixels having co-ordinates with even values of both K and L, including zero" (*i.e.*, 2/4,0/4 and 4/4,2/4 sub-pixels or 0/4,2/4 and 2/4,4/4 sub-pixels) recited in Element 75[a] for the reasons I discuss above.



TML6, 58-59 (/tml6pre.zip/lencod/src/image.c, ll. 593-632). Finally, the highlighted ¾,¾ sub-pixel (**green**) is interpolated using the nearest four unit pixels (**purple**).


0/4, 0/4
1/4, 0/4
2/4, 0/4
3/4, 0/4
4/4, 0/4
0/4, 1/4
1/4, 1/4
2/4, 1/4
3/4, 1/4
4/4, 1/4
0/4, 2/4
1/4, 2/4
2/4, 2/4
3/4, 2/4
4/4, 2/4
0/4, 3/4
1/4, 3/4
2/4, 3/4
3/4, 3/4
4/4, 3/4
0/4, 4/4
1/4, 4/4
2/4, 4/4
3/4, 4/4
4/4, 4/4

```
593     /* 1/4 pix */
594     /* luma */
595     ie2=(img->width-1)*4;
596     je2=(img->height-1)*4;
597
598     for (j=0;j<je2+4;j+=2)
599       for (i=0;i<ie2+3;i+=2)
600         mref[img->frame_cycle][j][i+1]=max(0,min(255,(int)(img4Y_tmp[j/2
          ][i/2]+img4Y_tmp[j/2][min(ie2/2+1,i/2+1)]+1024)/2048));
601
602           for (i=0;i<ie2+4;i++)
603           {
604               for (j=0;j<je2+3;j+=2)
605               {
606                   if( i%2 == 0 )
607                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int
                        )(img4Y_tmp[j/2][i/2]+img4Y_tmp[min(je2/2+1,j/2+1
                        )][i/2]+1024)/2048));
608                   else
609                       mref[img->frame_cycle][j+1][i]=max(0,min(255,(int)(
610                       img4Y_tmp[j/2                    ][i/2                    ] +
611                       img4Y_tmp[min(je2/2+1,j/2+1)][i/2                    ] +
612                       img4Y_tmp[j/2                    ][min(ie2/2+1,i/2+1)] +
613                       img4Y_tmp[min(je2/2+1,j/2+1)][min(ie2/2+1,i/2+1)]
                        + 2048)/4096));
614
615                   /* "funny posision" */
616                   if( ((i&3) == 3)&&(((j+1)&3) == 3))
617                   {
618                       mref[img->frame_cycle][j+1][i] = (mref[img->
                        frame_cycle][j-2          ][i-3          ] +
619                          mref[img->frame_cycle][min(je2,j+2)][i-3
                                ] +
620                          mref[img->frame_cycle][j-2          ][min(ie2,i
                          +1)] +
621                          mref[img->frame_cycle][min(je2,j+2)][min(ie2,i
                          +1)] + 2 )/4;
622                   }
623               }
624           }
```

TML6, 58 (/tml6pre.zip/lencod/src/image.c, ll. 593-624).

302. The ’273 patent explains, with reference to Figure 13(a), that TML6 interpolates the value for the ¼,¼ sub-pixel “g” (**green**) by using a weighted average of the same unit pixel A (**purple**) and three ½ sub-pixels b, c, and c (**blue**).

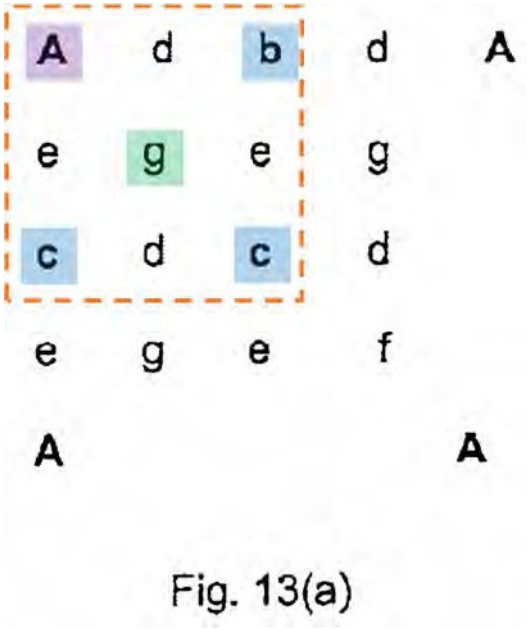


Fig. 13(a)

EX-1001, Figure 13(a), 12:34-40 ("Values for … g are computed using the value of the nearest original pixel A and the intermediate values of the three nearest neighbouring ½ resolution sub-pixels [b, c, and c], according to $g=(1024A+32b+32c_1+c_2+2048)/4096$"), 12:41-45 ("¼ resolution sub-pixel f is interpolated by averaging the values of the 4 closest pixels….").

303. As I describe above, TML6 interpolates the highlighted ¼,¼, ¼,¾, and ¾,¼ sub-pixels in the figures shown above by using a weighted average of (1) "the value of a nearest-neighbouring pixel," (2) "the value of the sub-pixel situated at co-ordinates ½,½," and (3-4) "the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of both K and L, including zero." I do note that, per one interpretation of the claim, TML6 potentially may not teach interpolating the ¼,¼, ¼,¾, ¾,¼ or ¾,¾ sub-pixel

according to a predetermined choice of both (1) a weighted average the values of a pair of the nearest-neighbouring pixel and the sub-pixel located at co-ordinates ½,½ and (2) a weighted average of the values of the pair of diagonally-opposed sub-pixels having co-ordinates with even values of both K and L, including zero. If the claim is interpreted in this manner, and if the Board determines that TML6 does not suggest this feature, the combined teachings of TML6 and Fandrianto do, as I describe below.

304. For example, as shown below in Figure 5, Fandrianto shows four unit pixels X (**purple**) within a rectangular bounded region (**red**)) and discloses interpolating a value for the highlighted ¼,¾ sub-pixel Q2 (**green**).

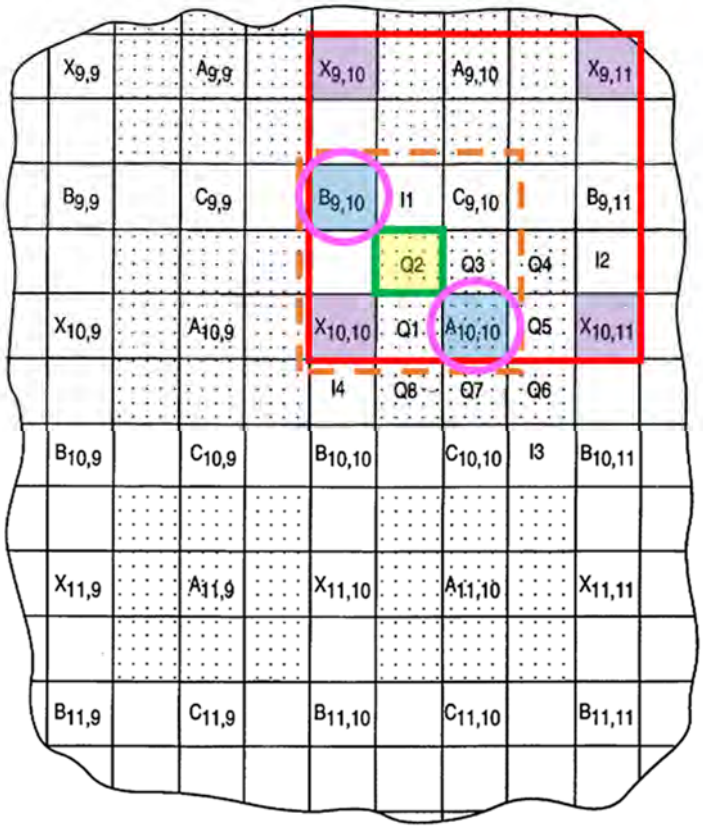


FIG. 5

Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 8:58-9:32 ("The second quarter pixel estimation search block of Q2 pixels is generated from a vertical interpolation of the Q1 pixels with the results of a horizontal interpolation of the data in the B block 74 and the C block 76, with the results being stored in the A block 72 (the

Q2 pixels progressively overwriting the Q1 pixels) for the absolute displaced block difference calculation, in accordance with the following expression.

$\boldsymbol{Q2_{r,c}=(Q1_{r,c}+I1_{r,c})/2}$ where $\boldsymbol{I1_{r,c}=(B_{r-1,c}+C_{r-1,c})/2}$. Consider, for example, the interpolation of the Q2 pixel to the upper left of pixel $A_{10,10}$. A collection of pixels containing the pixel group $B_{9,8}$-$B_{9,15}$ in row nine of the B block 74 is addressed on, say, port A of the DPCM memory 130 and the pixel group $B_{9,8}$-$B_{9,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the A port of the ALU 154. A1 about the same time, a collection of pixels containing the pixel group $C_{9,8}$-$C_{9,15}$ in row nine of the C block 76 is addressed on, say, port B of the DPCM memory 130 and the pixel group $C_{9,8}$-$C_{9,15}$ is selected by shifter 144 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed to the DP memory 124, where it is stored as pixels $I1_{9,8}$-$I1_{9,15}$ in the ninth row of I block 84, a free 16x16 block of memory in the DP memory 124. Next, the pixel group $I1_{9,8}$-$I1_{9,15}$ in row nine of the I block 84 is addressed on, say, port A of the DP memory 124 and presented through the multiplexer 152 to the A port of the ALU 154. At about the same time, a collection of pixels containing the pixel group $Q1_{10,8}$-$Q1_{10,15}$ in row ten of the A block 72 is addressed on, say, port A of the DPCM memory 130 and the pixel group $Q1_{10,8}$-$Q1_{10,15}$ is selected by shifter 140 and presented through the

multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed back to the DPCM memory, where it is stored as pixels $Q1_{10,8}$-$Q1_{10,15}$ in the tenth row of the A block 72."), 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels Q1-Q8 may be practiced, if desired…. As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74.").

305. This ¼,¾ sub-pixel Q2 (**green**) further has co-ordinates $K/2^N$, $L/2^N$ (K=1, L=3, N=2) with odd values of both K and L. Moreover, this sub-pixel Q2 is interpolated by using a weighted average of the values of a pair of diagonally-opposed sub-pixels (*i.e.*, the 0/4,2/4 sub-pixel B and the 2/4,4/4 sub-pixel A (both **blue** and circled)). The 0/4,2/4 sub-pixel B and the 2/4,4/4 sub-pixel A have co-ordinates $K/2^N$, $L/2^N$ (K=0 or 2, L=2 or 4, N=2) with even values of both K and L, including zero. Fandrianto, 8:58-9:32 ("The [] quarter pixel estimation search block of *Q2* pixels is generated … in accordance with the following expression $Q2_{r,c}=(Q1_{r,c}+I1_{r,c})/2$" [*i.e.*, a weighted average (with weights equaling "1") for a 'vertical interpolation' embodiment using sub-pixels Q1 and I1]" (emphasis added)); 10:51-58 (In another embodiment, "the Q2 pixels may be derived from a *diagonal interpolation* of the data in the A block… and the B block

[*i.e.*, a weighted average $\underline{Q2_{r,c}=(A_{r1,c}+B_{r,c})/2}$ using sub-pixels A and B (both **blue** and circled)]" (emphasis added)); 8:36-10:58. As illustrated above, these diagonally-opposed 0/4,2/4 sub-pixel B and 2/4,4/4 sub-pixel A are located within a quadrant (**orange** box) of the rectangular bounded region (**red** box). This quadrant is defined by corner pixels having co-ordinates ½,½ (*i.e.*, the 2/4,2/4 sub-pixel $C_{9,10}$) and the nearest-neighbouring pixel (*i.e.*, 0/4,4/4 pixel $X_{10,10}$). *Id.*

306. Fandrianto also teaches alternatively interpolating the highlighted ¼,¾ sub-pixel Q2 (**green**) using a weighted average of the value of a nearest-neighbouring pixel (*i.e.*, 0/4,4/4 sub-pixel X) (**purple** and circled) and the value of the sub-pixel C located at co-ordinates ½,½ (**blue** and circled), as shown below.

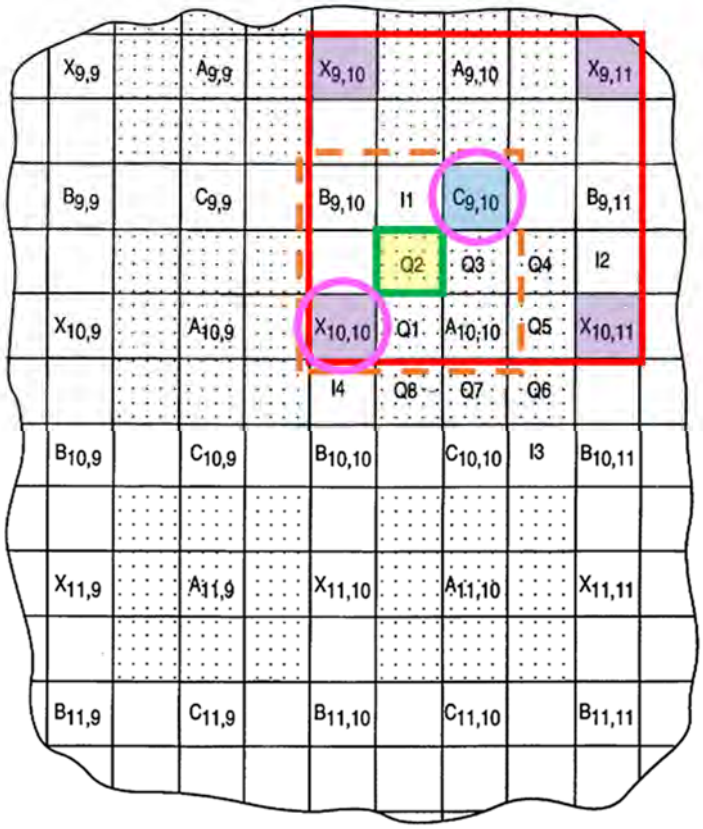


FIG. 5

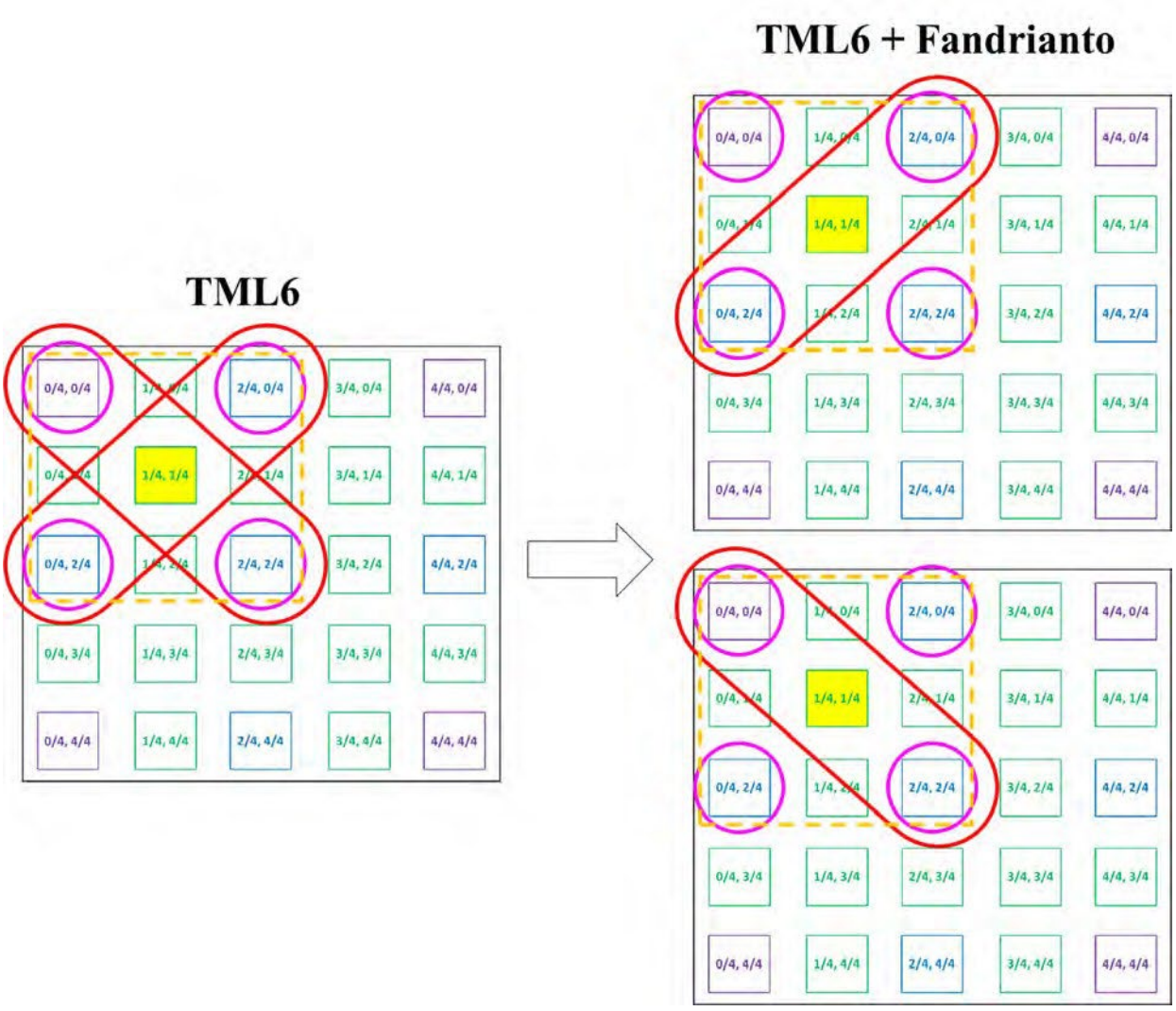
Fandrianto, Figure 5, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 7:52-55 ("[W]hen ***diagonal interpolation*** is also used, all quarter pixel estimation search blocks [including sub-pixel Q2 (**green**)] are interpolated from the full [unit pixel X (**purple**)] and half pixel [C (**blue**)] estimation search blocks."), 8:58-9:32 ("The second quarter

pixel estimation search block of Q2 pixels is generated from a vertical interpolation of the Q1 pixels with the results of a horizontal interpolation of the data in the B block 74 and the C block 76, with the results being stored in the A block 72 (the Q2 pixels progressively overwriting the Q1 pixels) for the absolute displaced block difference calculation, in accordance with the following expression. $\boldsymbol{Q2_{r,c}=(Q1_{r,c}+I1_{r,c})/2}$ where $\boldsymbol{I1_{r,c}=(B_{r-1,c}+C_{r-1,c})/2}$ Consider, for example, the interpolation of the Q2 pixel to the upper left of pixel $A_{10,10}$. A collection of pixels containing the pixel group $B_{9,8}$-$B_{9,15}$ in row nine of the B block 74 is addressed on, say, port A of the DPCM memory 130 and the pixel group $B_{9,8}$-$B_{9,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the A port of the ALU 154. A1 about the same time, a collection of pixels containing the pixel group $C_{9,8}$-$C_{9,15}$ in row nine of the C block 76 is addressed on, say, port B of the DPCM memory 130 and the pixel group $C_{9,8}$-$C_{9,15}$ is selected by shifter 144 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed to the DP memory 124, where it is stored as pixels $I1_{9,8}$-$I1_{9,15}$ in the ninth row of I block 84, a free 16x16 block of memory in the DP memory 124. Next, the pixel group $I1_{9,8}$-$I1_{9,15}$ in row nine of the I block 84 is addressed on, say, port A of the DP memory 124 and presented through the multiplexer 152 to the A port of the ALU 154. At about the same time,

a collection of pixels containing the pixel group $Q1_{10,8}$-$Q1_{10,15}$ in row ten of the A block 72 is addressed on, say, port A of the DPCM memory 130 and the pixel group $Q1_{10,8}$-$Q1_{10,15}$ is selected by shifter 140 and presented through the multiplexer 152 to the B port of the ALU 154. The ALU 154 sums the pixel groups, divides by two, and rounds the result. An eight pixel result at the output of the ALU 154 is routed back to the DPCM memory, where it is stored as pixels $Q1_{10,8}$-$Q1_{10,15}$ in the tenth row of the A block 72."), 10:51-58 ("It will be appreciated that a great many permutations of the order of derivation of pixels Q1-Q8 may be practiced, if desired…. As a further example, the Q2 pixels may be derived from a diagonal interpolation of the data in the A block 72 and the B block 74."). Fandrianto, therefore, additionally discloses that the ¼,¾ sub-pixel Q2 is diagonally interpolated using the nearest-neighbouring pixel $X_{10,10}$ and the sub-pixel $C_{9,10}$ with co-ordinates ½,½ as recited in Element 75[c]. *Id*. As with the other disclosed interpolations, this interpolation is a weighted average of two values of the pixel $X_{10,10}$ and sub-pixel $C_{9,10}$, with weights equal to "1." *Id*., Figure 5, 6:18-7:2, 8:36-10:50.

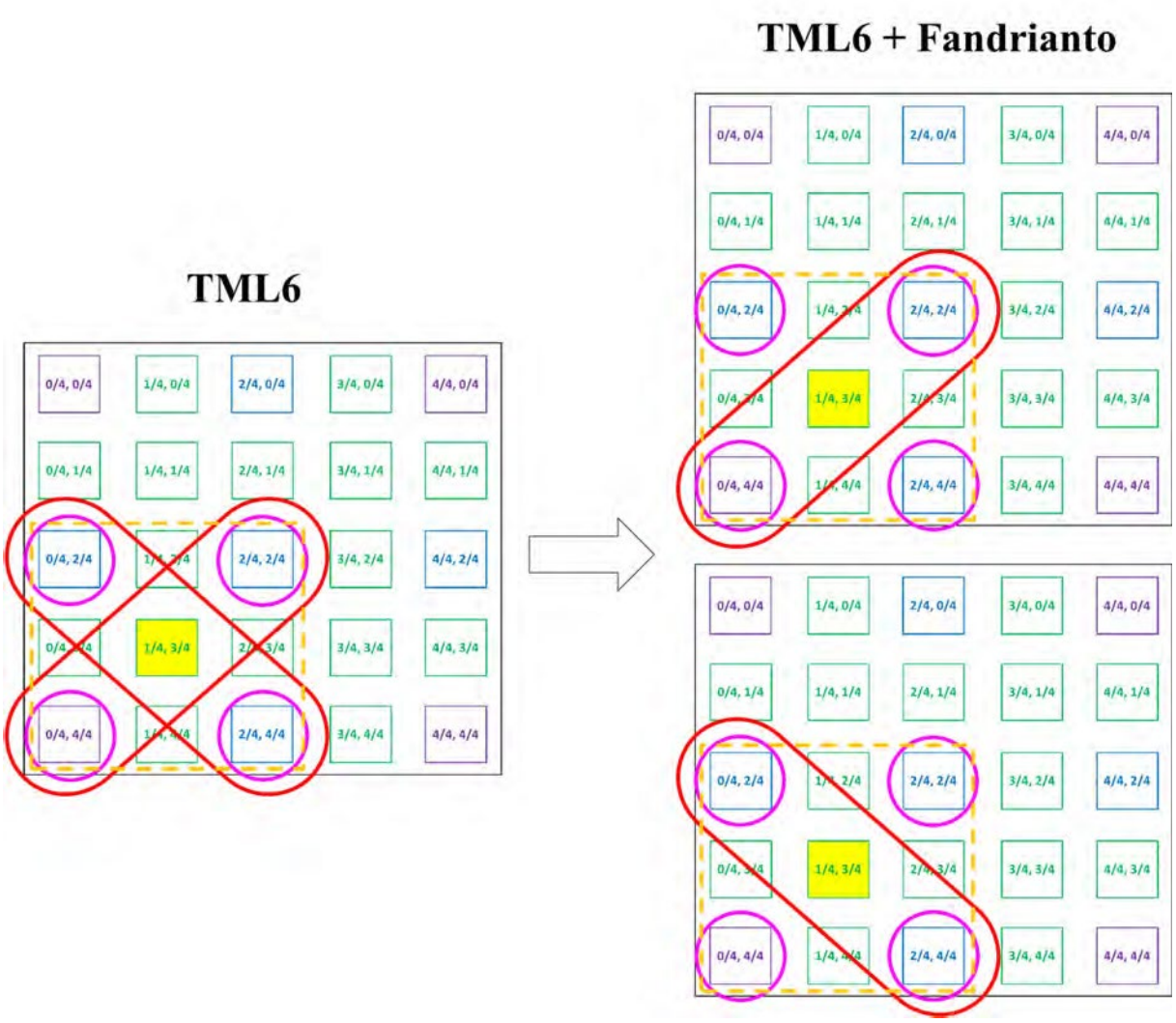
307. As I explain in Section IX.A.1 (motivation to combine), a POSITA would have been motivated to use the interpolation method, as set forth in Fandrianto, for interpolating the ¼,¾ sub-pixel Q2 (**green**) to also interpolate the ¼,¼ sub-pixel, ¼,¾ sub-pixel, ¾,¼ sub-pixel and/or ¾,¾ sub-pixel in TML6. For

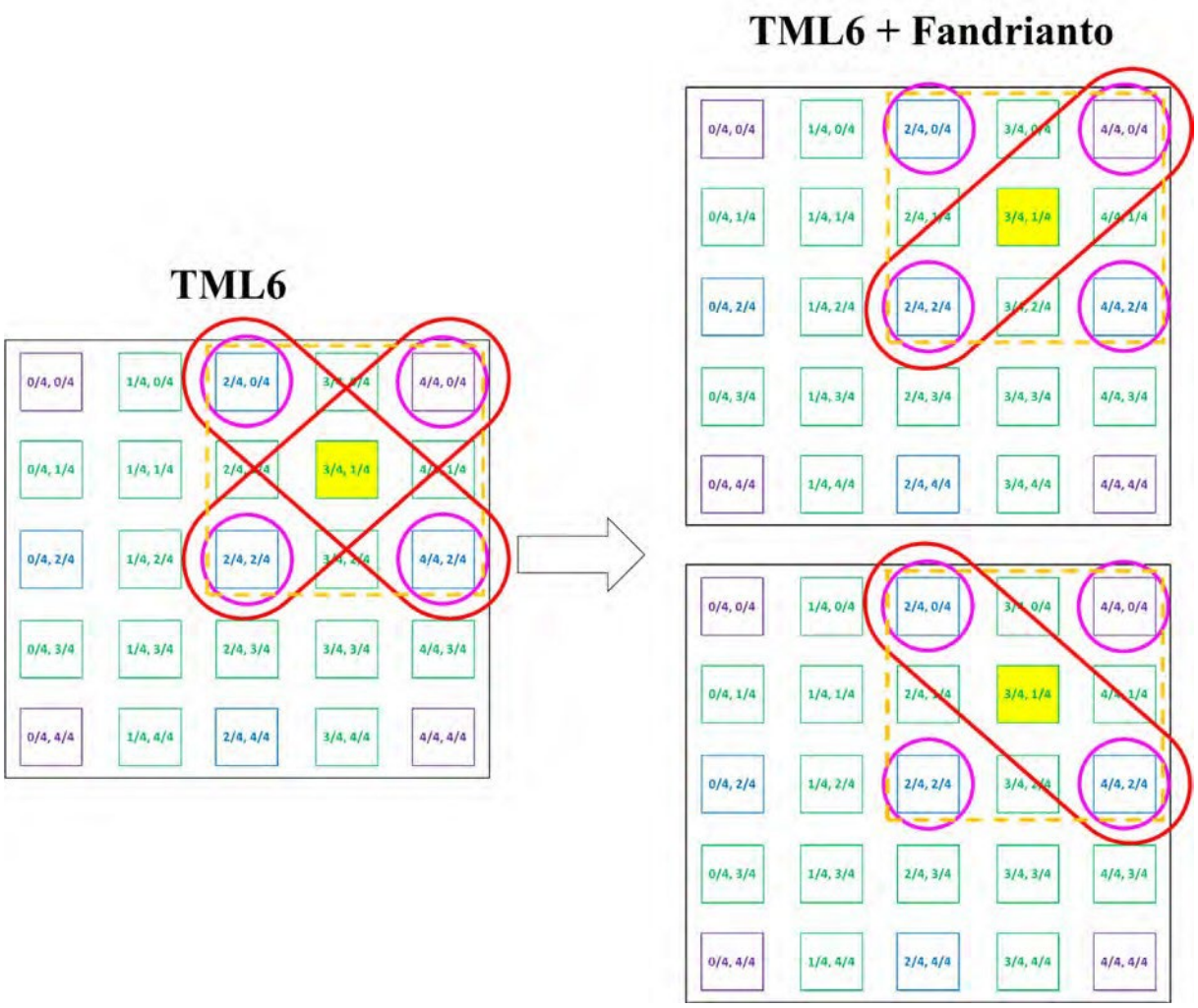
example, a POSITA would have known how to modify the code of TML6 to use Fandrianto's interpolation method or by using Fandrianto's technique on its hardware. The combined teachings of TML6 and Fandrianto, as illustrated below, thus result in a system or software that interpolates TML6's ¼,¼ sub-pixel (**green**) using (as shown on the bottom right) a weighted average of the value of a nearest-neighbouring 0/4,0/4 pixel (**purple**) and the value of the 2/4,2/4 sub-pixel (**blue**) located at co-ordinates ½,½, *or* (as shown on the upper right) a weighted average of the values of a pair of diagonally-opposed sub-pixels (*i.e.*, the 0/4,2/4 and 2/4,0/4 sub-pixels) (**blue**) having co-ordinates with even values of both K and L, including zero (*i.e.*, K=0 or 2, L=0 or 2), located within a quadrant (**orange** box) of the rectangular bounded region defined by corner pixels having co-ordinates ½,½ and the nearest-neighbouring pixel, as in Fandrianto.

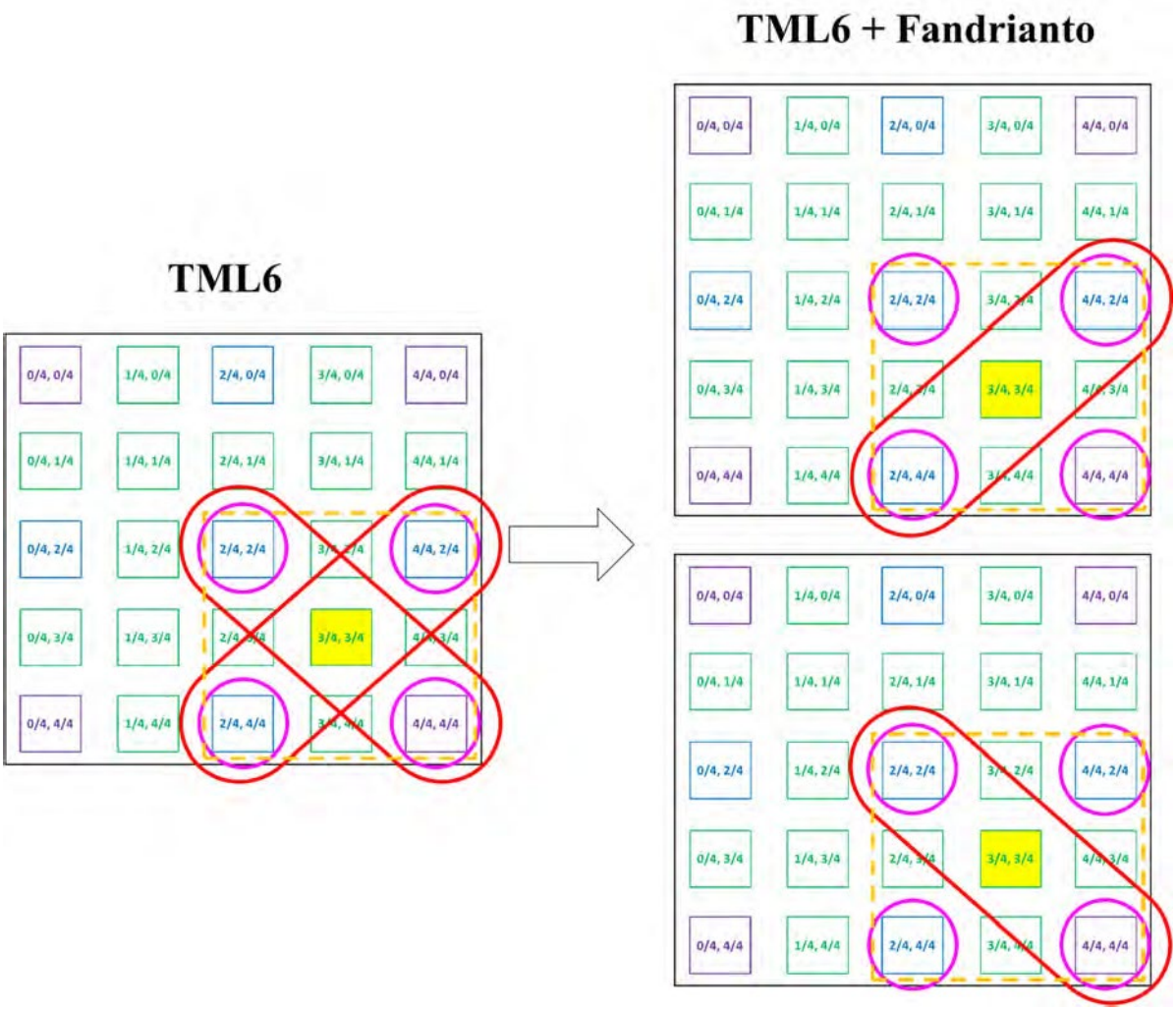


See supra Section IX.A.1 (motivation to combine TML6 and Fandrianto).

Moreover, as illustrated below, the combined teachings of TML6 and Fandrianto teach interpolating TML6's ¼,¾ sub-pixel (**green**), and the ¾,¼ sub-pixel (**green**), and the ¾,¾ sub-pixel (**green**) in the same way.


TML6
0/4, 0/4
1/4, 0/4
2/4, 0/4
3/4, 0/4
4/4, 0/4
0/4, 1/4
1/4, 1/4
2/4, 1/4
3/4, 1/4
4/4, 1/4
0/4, 2/4
2/4, 2/4
3/4, 2/4
4/4, 2/4
1/4, 3/4
3/4, 3/4
4/4, 3/4
0/4, 4/4
2/4, 4/4
3/4, 4/4
4/4, 4/4
TML6 + Fandrianto


TML6
TML6 + Fandrianto
0/4, 0/4
1/4, 0/4
2/4, 0/4
3/4, 0/4
4/4, 0/4
0/4, 1/4
1/4, 1/4
3/4, 1/4
0/4, 2/4
1/4, 2/4
2/4, 2/4
4/4, 2/4
0/4, 3/4
1/4, 3/4
2/4, 3/4
3/4, 3/4
4/4, 3/4
0/4, 4/4
1/4, 4/4
2/4, 4/4
3/4, 4/4
4/4, 4/4



See supra Section IX.A.1 (motivation to combine TML6 and Fandrianto).

308. Furthermore, on the same basis discussed above in Section IX.A.1, *supra*, a POSITA would have found it obvious to interpolate TML6's ¼,¼, ¼,¾, ¾,¼, and/or ¾,¾ sub-pixels using a predetermined choice of the two interpolation methods depicted above and disclosed in Fandrianto, such that that one or more these sub-pixels are interpolated with each of the two methods. Fandrianto is express in its teaching that a finite number of interpolation methods can be used to

interpolate the ¼,¼, ¼,¾, ¾,¼, and ¾,¾ sub-pixels, including the option of vertical interpolation and the option of two diagonal interpolations. It would, therefore, have been obvious for a POSITA to try these limited options, including making a predetermined choice between these two diagonal-interpolation options, to interpolate any of the ¼,¼, ¼,¾, ¾,¼, and ¾,¾ sub-pixels. Moreover, modifying the TML6 code or Fandrianto's hardware to implement this choice would have been straightforward, and a POSITA would have reasonably expected such a modification to yield the predictable result of interpolating ¼,¼, ¼,¾, ¾,¼, and/or ¾,¾ sub-pixels based on values of nearby pixels and/or sub-pixels.

309. Element 75[a] also states that the interpolator has circuitry configured to interpolate the sub-pixel values as set forth in the claim. TML6 discloses source code that is designed to run on a processor, and a processor clearly needs circuitry to run the code. *See* Fandrianto, 1:61-67 ("effective video coding requires integration of… integrated circuit technology, and digital storage media…."). And the '273 patent explains that TML6 has such circuitry. EX-1001, 12:65-13:17 ("TML6… requires more silicon area in ASICs and requires more computations in some CPUs"). TML6 explicitly or implicitly teaches circuitry to run the interpolations in the ***oneforthpix()*** function, as laid out in Element 75[a].

310. Fandrianto also sets forth an interpolator having circuitry configured to interpolate sub-pixels, as shown below.

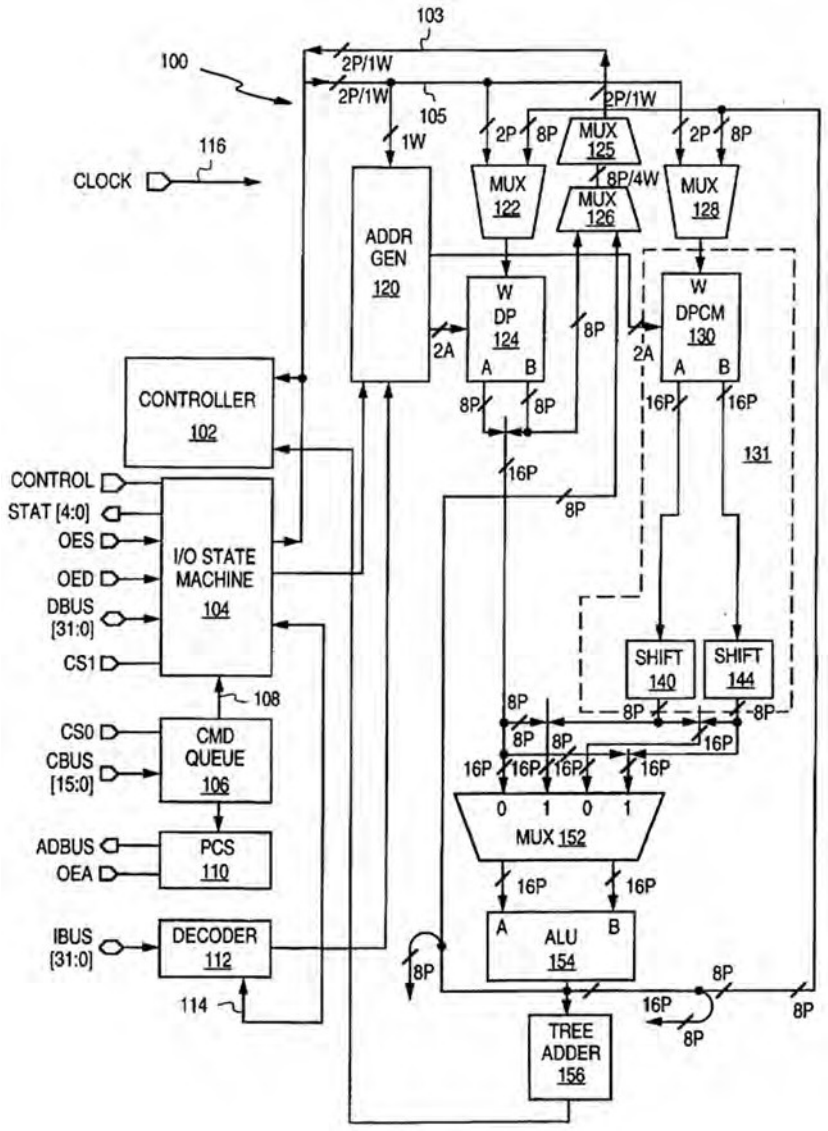


FIG. 2

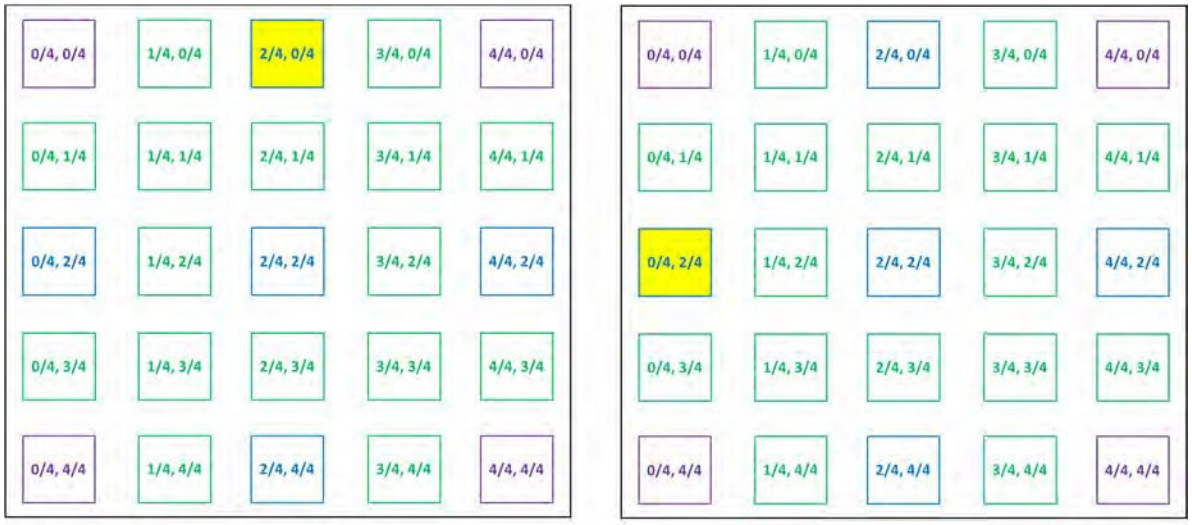
Fandrianto, Figures 2-5, 1:61-67, 2:48-58, 3:55-4:53, 5:26-7:2, 7:27-10:58 ("Various interpolation techniques may be used…[t]he following [interpolation] technique is suitable for the architecture of FIG. 2").

311. The combined teachings of TML6 and Fandrianto, therefore, disclose an interpolator that includes circuitry configured to interpolate sub-pixels as stated in Elements 1[a] and 75[a].

d) Element 75[b]: "circuitry configured to interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

312. As illustrated below, the ***oneforthpix()*** function of TML6 interpolates values for the 2/4,0/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$) with K equal to an even value and L equal zero (K=2, L=0, N=2) and for the 0/4,2/4 sub-pixel having co-ordinates ($K/2^N$, $L/2^N$) with K equal to zero and L equal to an even value (K=0, L=2, N=2).[22] TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 538-592).

[22] In my analysis of Element 75[b], the various illustrations show sub-pixels within one rectangular bounded region of TML6 and Fandrianto. However corresponding sub-pixels in other rectangular bounded regions of pixels and sub-pixels are interpolated in the same manner. TML6, 39 (/tml6pre.zip/lencod/src/lencod.c, ll. 1338-1344), 57-59 (/tml6pre.zip/lencod/src/image.c, ll. 538-632); Fandrianto, Figure 5, 5:26-6:48. Accordingly, TML6 and Fandrianto interpolate multiple



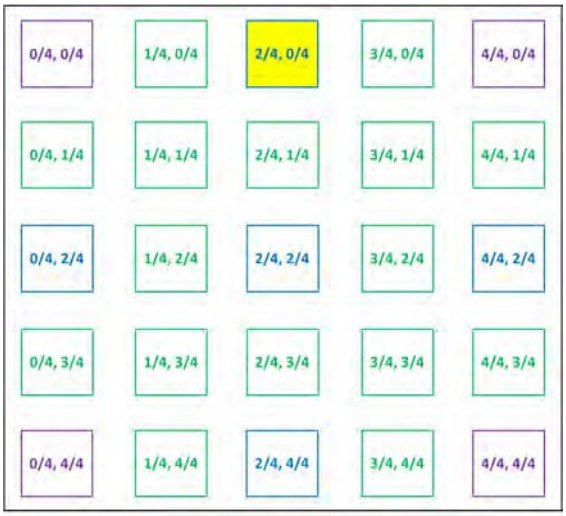
As I explained above for Element 75[a], these 2/4,0/4 and 0/4,2/4 sub-pixels are used to interpolate the values of the sub-pixels having co-ordinates ¼,¼, ¼,¾, ¾,¼ and ¾,¾ with odd values of both K and L (K=1 or 3, L=1 or 3). Element 75[a], *supra*. The ***oneforthpix()*** function of TML6 also interpolates values for these

---

sub-pixels with even values of K and L, as claimed. If the claim requires interpolating, in one rectangular bounded region, multiple sub-pixels having co-ordinates with K equal to an even value and L equal zero and multiple sub-pixels having co-ordinates with K equal to zero and L equal to an even value, the scope of the claim is unclear, since there is only one such sub-pixel when N=2. However, TML6 or the combination of TML6 and Fandrianto shows this claimed feature to the same extent that the '273 patent discloses this claimed feature.

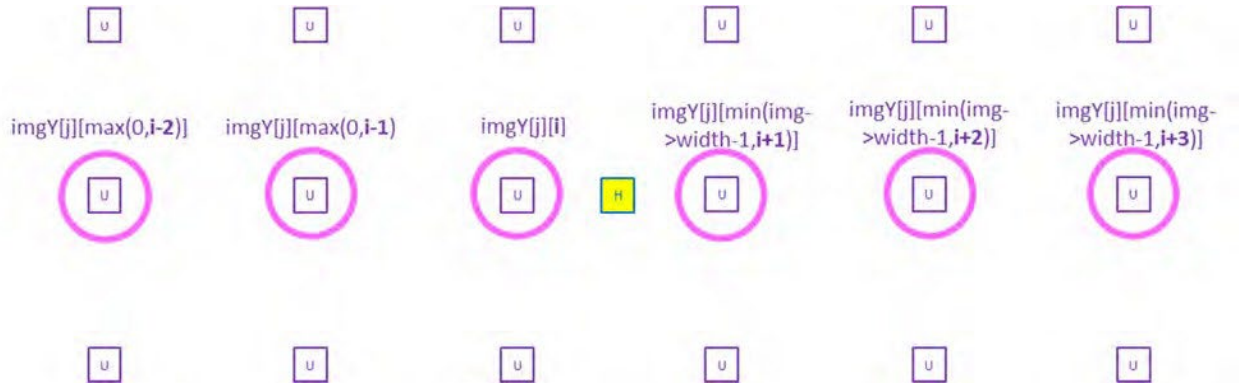
2/4,0/4 and 0/4,2/4 sub-pixels using the values of the unit pixels (**purple**), which are located in rows and columns, respectively. TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 538-592).

313. In particular, the ***oneforthpix()*** function interpolates the values for the 2/4,0/4 sub-pixels (**blue**) (highlighted below) with values of the unit pixels (**purple**) that are stored in the data structure ***imgY*** and that are in the same row as the highlighted sub-pixel.



(2/4,0/4)
K=2, L=0

As further illustrated below, these highlighted 2/4,0/4 sub-pixels (**blue**) are interpolated in the first step of the ***oneforthpix()*** function by using a weighted sum of the values of the three adjacent circled unit pixels (**purple**) to the left of the highlighted sub-pixel and the three adjacent circled unit pixels (**purple**) to the right of the highlighted sub-pixel.




```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                    img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                    img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 563-575, 588). In this weighted sum, the weights assigned to each unit pixel (**purple**) value are stored in a data structure ***ONE_FOURTH_TAP*** (defined in the file ***image.h*** in the ***lencod*** folder).

```
563     for (j=0; j < img->height; j++)
564       {
565           for (i=0; i < img->width; i++)
566           {
567               i2=i*2;
568               is=(ONE_FOURTH_TAP[0][0]*(imgY[j][i          ]+imgY[j][min(
                  img->width-1,i+1)])+
569                   ONE_FOURTH_TAP[1][0]*(imgY[j][max(0,i-1)]+imgY[j][min(
                    img->width-1,i+2)])+
570                   ONE_FOURTH_TAP[2][0]*(imgY[j][max(0,i-2)]+imgY[j][min(
                    img->width-1,i+3)]));
571               img4Y_tmp[2*j][i2  ]=imgY[j][i]*1024;     /* 1/1 pix pos */
572               img4Y_tmp[2*j][i2+1]=is*32;               /* 1/2 pix pos */
573           }
574     }
575
```

```
28   /* TAPs used in the oneforthpix()routine */
29    const int ONE_FOURTH_TAP[3][2] =
30     {
31        {20,20},
32        {-5,-4},
33        { 1, 0},
34     };
35
```

TML6, 57 (/tml6pre.zip/lencod/src/image.c, ll. 568-570), 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34); *id.*, 57 (/tml6pre.zip/lencod/src/image.c, l. 572) (as shown at line 572, the weighted sum is multiplied by 32, *i.e.*, each of the weights from the data structure ***ONE_FOURTH_TAP*** in the weighted sum is multiplied by a scaling factor, 32). A value including this weighted sum is stored in the data structure ***img4Y_tmp*** and then, after being rounded and brought into the dynamic range, is stored in the data structure ***mref***.

```
572                 img4Y_tmp[2*j][i2+1]=is*32;              /* 1/2 pix pos */
```

```
588                 mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                    img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 572, 588).

314. The ’273 patent, in conjunction with Figure 13(b), explains that TML6 interpolates these ½ sub-pixels in this manner.

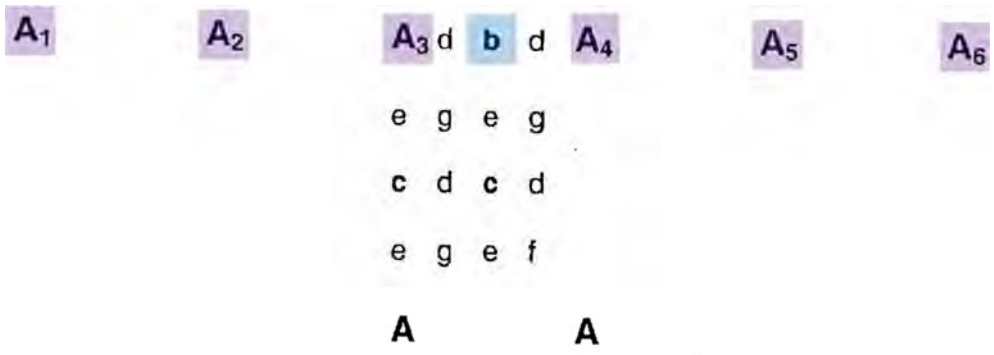
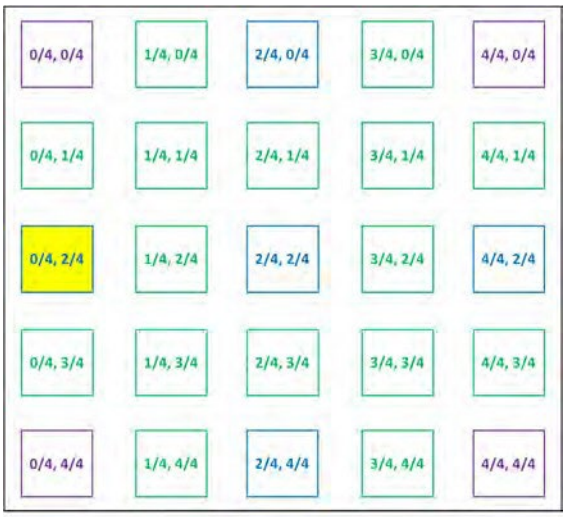


Figure13(b)

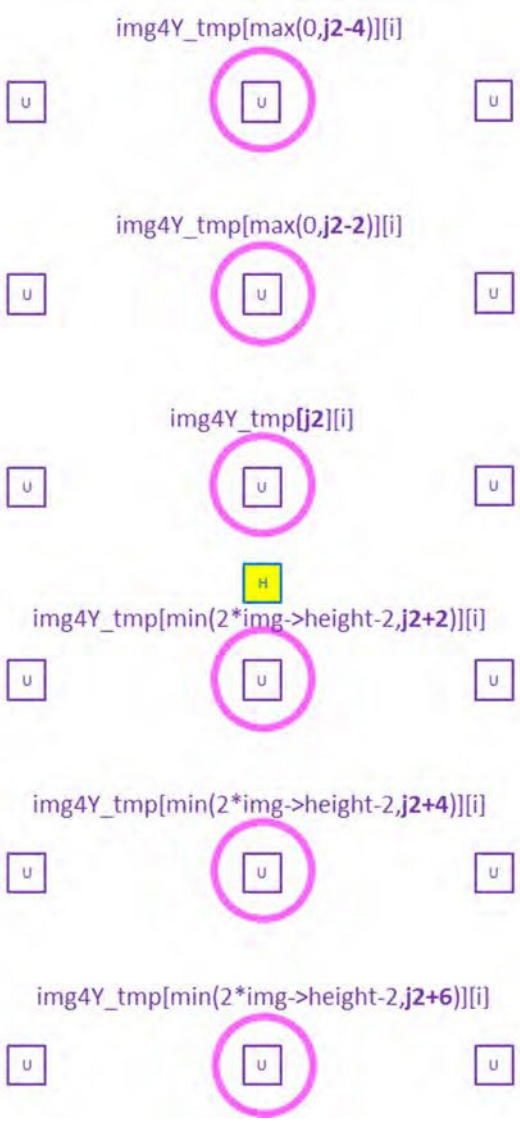
The '273 patent notes that the value for the 2/4,0/4 sub-pixel "b" (**blue**) above is interpolated "based upon the values of the 6 pixels (A_1 to A_6) [**purple**] situated in a row at unit horizontal locations and unit vertical locations symmetrically about b, as shown in FIG. 13*b*, according to the formula $b=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$." EX-1001, Figures 13(a)-13(b), 11:35-45.

315. The ***oneforthpix()*** function similarly interpolates the values for the 0/4,2/4 sub-pixels (**blue**) (highlighted below) by using unit pixels (**purple**) values that are stored in the data structure ***imgY*** and that are in the same column as the highlighted sub-pixels.



(0/4,2/4)
K=0, L=2

As illustrated below, these highlighted 0/4,2/4 sub-pixels (**blue**) are interpolated in the second step of the ***oneforthpix()*** function by using a weighted sum of the values of the three adjacent circled unit pixels (**purple**) above the highlighted sub-pixel and the three adjacent circled unit pixels (**purple**) below the highlighted sub-pixel.


img4Y_tmp[max(0,j2-4)][i]
U
U
U
img4Y_tmp[max(0,j2-2)][i]
U
U
U
img4Y_tmp[j2][i]
U
U
U
H
img4Y_tmp[min(2*img->height-2,j2+2)][i]
U
U
U
img4Y_tmp[min(2*img->height-2,j2+4)][i]
U
U
U
img4Y_tmp[min(2*img->height-2,j2+6)][i]
U
U
U

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).

316. The weights in this weighted sum that are assigned to these values of the unit pixels (**purple**) are stored in the data structure ***ONE_FOURTH_TAP***, which I discuss above.

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE FOURTH TAP[0][0]*(img4Y tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

```
28    /* TAPs used in the oneforthpix()routine */
29     const int ONE_FOURTH_TAP[3][2] =
30       {
31         {20,20},
32         {-5,-4},
33         { 1, 0},
34       };
35
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592), 3 (/tml6pre.zip/lencod/inc/image.h, ll. 28-34). A value including this weighted sum is stored in the data structure ***img4Y_tmp***. Then, it is rounded and brought into the dynamic range, the value is stored in the data structure ***mref***.

```
587               img4Y tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).

317. The ’273 patent, with reference to Figure 13(c), explains that TML6 interpolates these ½ sub-pixels in the same manner.

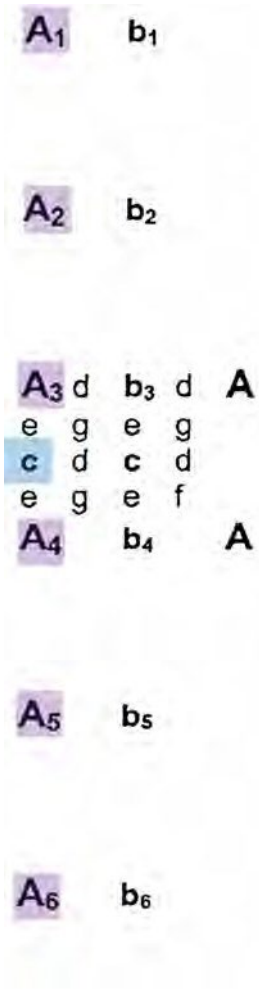


Figure 13(c)

The ’273 patent notes that value for the 0/4,2/4 sub-pixel “c” (**blue**) (above) is interpolated “based upon the values of the 6 pixels (A_1 to A_6) [**purple**] situated in a column at unit horizontal locations and unit vertical locations symmetrically about c [**blue**], according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$.” EX-1001, Figures 13(a)-13(c), 11:46-11:63 (“Values for the ½ resolution sub-pixels labelled

c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula $c=(A_1-5A_2+20A_3+20A_4-5A_5+A_6)$. The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to $c=(c+16)/32$. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator / denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range $[0, 2^n-1]$.")

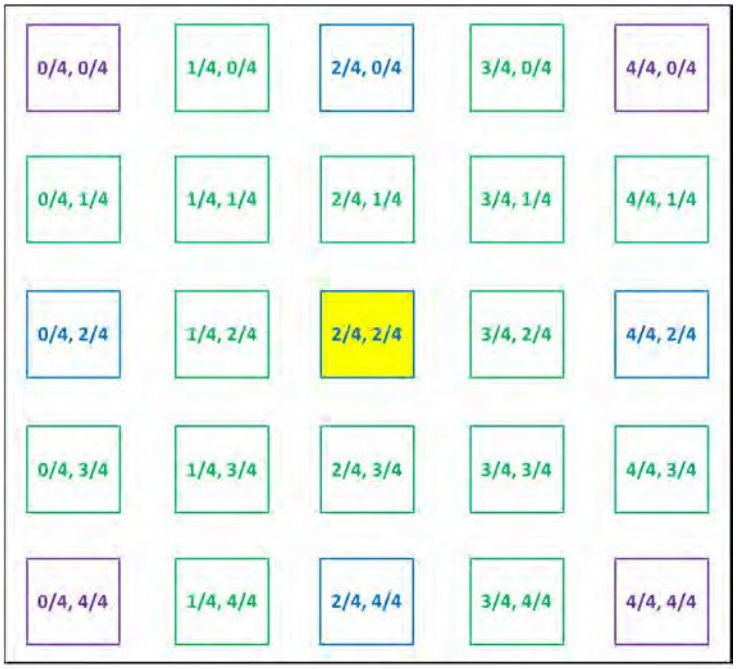
318. For the same reasons that I discuss above for Elements 1[b] and 75[a], TML6 and Fandrianto also disclose circuitry configured to interpolate values for the claimed sub-pixels of Element 75[b]. 1[b], 75[a], *supra*.

e) Element 75[c]: "circuitry configured to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal

to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

319. As shown below, TML6's ***oneforthpix()*** function interpolates values for 2/4,2/4 sub-pixels (**blue**) (highlighted below) having co-ordinates ($K/2^N$, $L/2^N$, where K=L=N=2) with even values of both K and L.[23]

[23] As I mentioned in my analysis of Element 75[b] above, the combined teachings of TML6 and Fandrianto disclose interpolating these sub-pixels in other rectangular bounded regions. Element 75[b], *supra*. If claim requires interpolating *multiple* such sub-pixels in one rectangular bounded region with even values of K and L (as claimed), the scope of the claim is unclear, since there is only *one* such sub-pixel when N=2. However, TML6 or the combination of TML6 and Fandrianto show this claimed feature to the same extent that the '273 patent discloses this claimed feature.

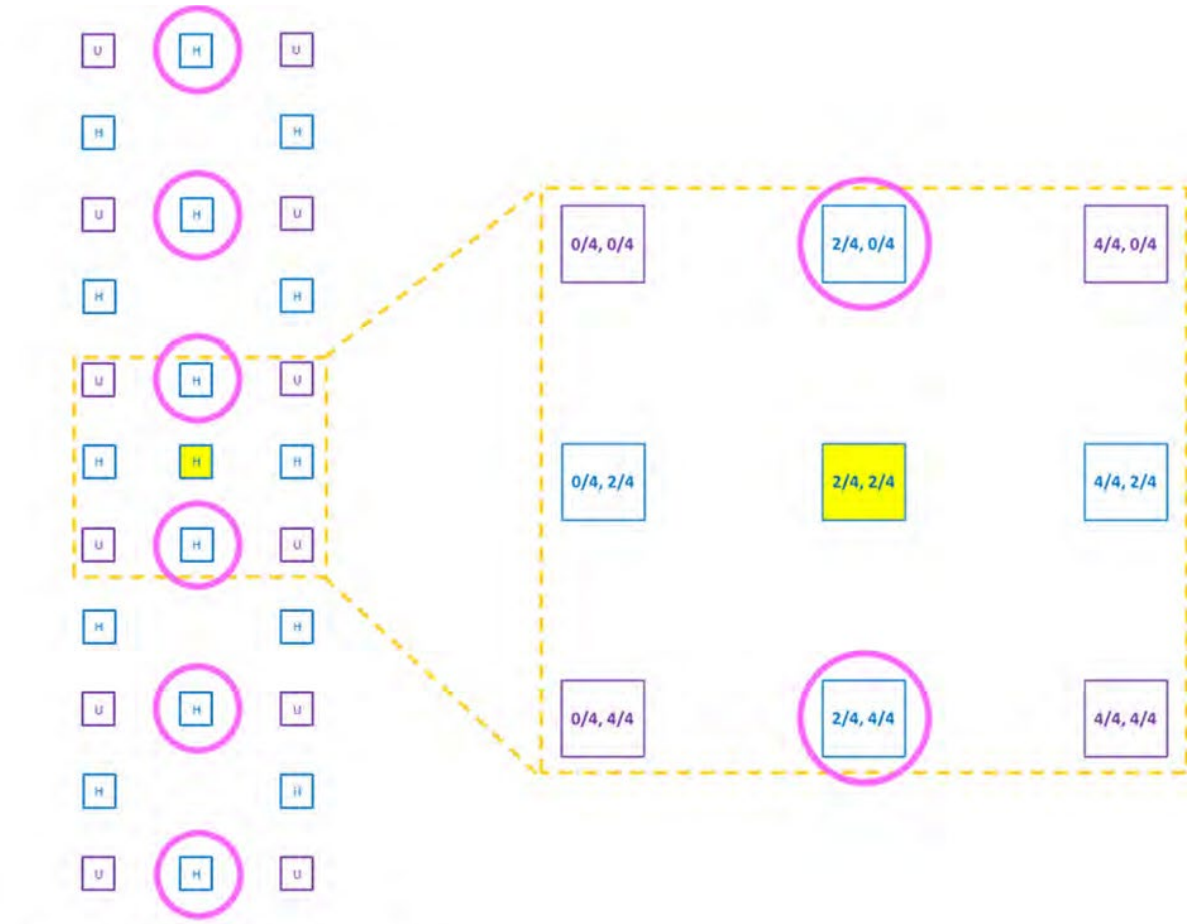


TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592). As I discuss with respect to Element 75[a], these 2/4,2/4 sub-pixels are "situated at co-ordinates ½,½" and are used in the interpolation of the values for the sub-pixels having co-ordinates ¼,¼, ¼,¾, ¾,¼, and/or ¾,¾ with odd values of K and L. Element 75[a], *supra*.

320. As illustrated below, the highlighted 2/4,2/4 sub-pixels (**blue**) are interpolated in the second step of the ***oneforthpix()*** function using a weighted sum (hereinafter referred to as the "first weighted sum") of: (1) The values of the circled 2/4,0/4 sub-pixel and 2/4,4/4 sub-pixel (**blue**) (in the **orange** box) having co-ordinates ($K/2^N$, $L/2^N$, where K=2, L=0 or 4, N=2) with K equal to an even

value and L equal to zero,[24] and (2) The values of the circled 2/4,0/4 sub-pixel and 2/4,4/4 sub-pixel (in the **green** boxes) having corresponding co-ordinates in immediately adjacent rectangular bounded regions.

[24] Under one claim interpretation, Element 75[c] possibly may require interpolating sub-pixels (with even values of both K and L) using a weighted sum of: (1) *at least two sub-pixels* having co-ordinates with K equal to an even value and *L equal to zero* (both within the same bounded region), and (2) corresponding sub-pixels in adjacent rectangular bounded regions. However, the '273 patent discloses interpolating the 2/4,2/4 sub-pixel c using a weighted sum of: (1) *only one sub-pixel* having co-ordinates with K equal to an even value (*i.e.*, 2) and *L equal to zero* (*i.e.*, the 2/4,0/4 sub-pixel b_3) *and* a second 2/4,4/4 sub-pixel b_4 having co-ordinates with K equal to an even value (*i.e.*, 2) and L equal to *four* (and *not* zero), and (2) corresponding sub-pixels b_1 and b_2 and sub-pixels b_5 and b_6 in adjacent rectangular bounded regions. EX-1001, Figures 14(a)-14(c), 45:2-43 ("$c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6+512)$"). TML6 or the combination of TML6 and Fandrianto discloses interpolating these sub-pixels in the same way as described below and as confirmed by the '273 patent. *Id.*, Figure 13(c), 11:46-63 ("$c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$").



First Weighted Sum

```
576     for (i=0; i < img->width*2; i++)
577       {
578           for (j=0; j < img->height; j++)
579           {
580               j2=j*2;
581               j4=j*4;
582               /* change for TML4, use 6 TAP vertical filter */
583               is=(ONE_FOURTH_TAP[0][0]*(img4Y_tmp[j2          ][i]+
                  img4Y_tmp[min(2*img->height-2,j2+2)][i])+
584                   ONE_FOURTH_TAP[1][0]*(img4Y_tmp[max(0,j2-2)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+4)][i])+
585                   ONE_FOURTH_TAP[2][0]*(img4Y_tmp[max(0,j2-4)][i]+
                      img4Y_tmp[min(2*img->height-2,j2+6)][i]))/32;
586
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
590           }
591       }
592
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592). A value having this first weighted sum is stored in the data structure ***img4Y_tmp***. Then, after it is rounded and brought into the dynamic range, the value is stored in the data structure ***mref***.

```
587               img4Y_tmp[j2+1][i]=is;                    /* 1/2 pix */
588               mref[img->frame_cycle][j4  ][i*2]=max(0,min(255,(int)((
                  img4Y_tmp[j2][i]+512)/1024)));     /* 1/2 pix */
589               mref[img->frame_cycle][j4+2][i*2]=max(0,min(255,(int)((is+
                  512)/1024))); /* 1/2 pix */
```

TML6, 57-58 (/tml6pre.zip/lencod/src/image.c, ll. 576-592).

321. The ’273 patent, with reference to Figure 13(c), notes that TML6 interpolates the value for the 2/4,2/4 sub-pixels c (**blue**) in the same way.

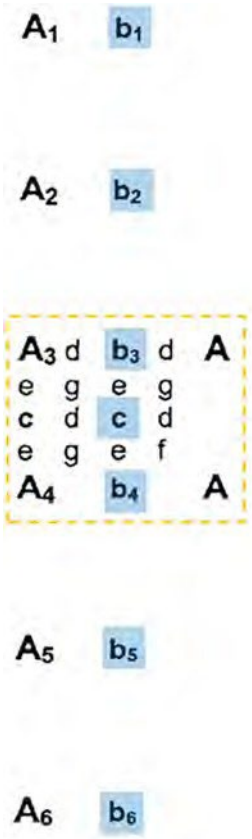


Figure 13(c)

EX-1001, Figures 13(c), 11:46-11:63 ("Values for the ½ resolution sub-pixels labelled c are obtained by first calculating intermediate values c. Referring to FIG. 13*c*, an intermediate value c for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is calculated based upon the values of the 6 pixels (A_1 to A_6) situated in a column at unit horizontal locations and unit vertical locations symmetrically about c, according to the formula c (A_1-$5A_2$+$20A_3$+$20A_4$-$5A_5$+A_6). The final value for the ½ resolution sub-pixel c located at unit horizontal and half vertical location is then calculated according to

c=(c+16)/32. Similarly, an intermediate value c for the ½ resolution sub-pixel c at half horizontal and half Vertical location is calculated according to $c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6)$. A final value for this ½ resolution sub-pixel is then calculated according to (c+512)/1024. Again, the operator/denotes division with truncation and the values calculated for ½ resolution sub-pixels c are further clipped to lie in the range [0, 2^n-1].").

322. Fandrianto also explains that the values for the highlighted 2/4,2/4 sub-pixel C (**blue**, below) are interpolated *either* by (1) averaging the circled 2/4,0/4 sub-pixel A above and the 2/4,4/4 sub-pixel A below the highlighted sub-pixel ("vertical interpolation") *or* (2) averaging the circled 0/4,2/4 sub-pixel B to the left and the 4/4,2/4 sub-pixel B to the right of the highlighted 2/4,2/4 sub-pixel C ("horizontal interpolation").

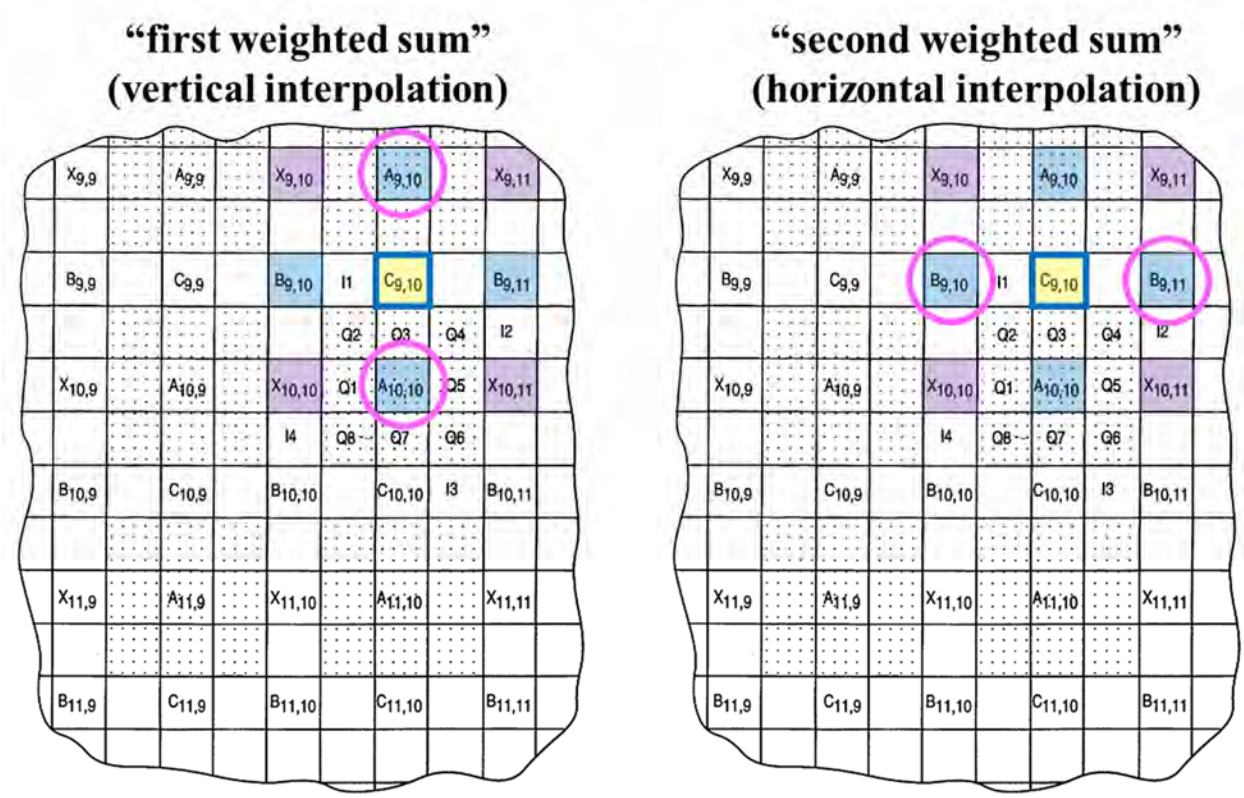


FIG. 5 FIG. 5

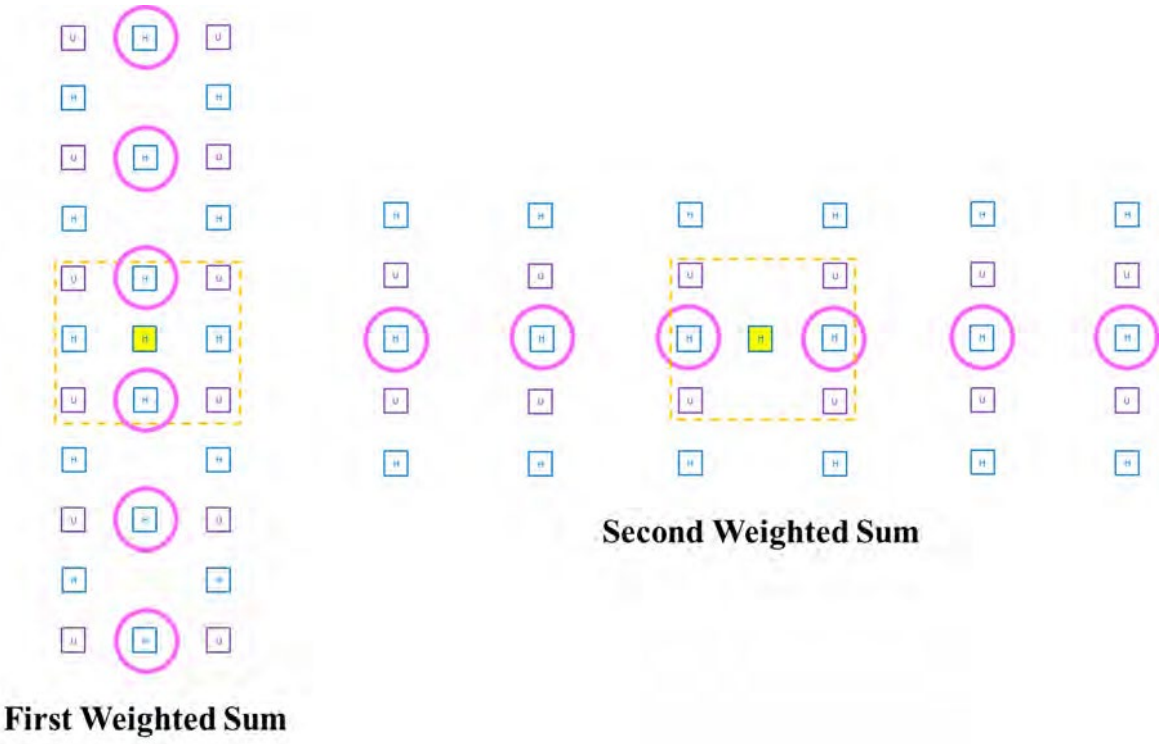
Fandrianto, 5:26-32 ("For improved prediction accuracy, half-pixel estimation and quarter-pixel estimation are performed after the integral pixel estimation. In motion vector searching with one-half or one-quarter pixel accuracy, the DPCM memory 130 is used to store a search window 24 that is in large part an interpolated search matrix generated from a best match search block from a less accurate estimation operation."), 6:4-12 ("Pixels $A_{r,c}$ (r=0, …,17; c=0, …, 16) of the A block 72 are horizontally interpolated at half-pixel locations, from the X block 70. Pixels $B_{r,c}$ (r=0, …, 16; c=0, …, 17) of the B block 74 are vertically interpolated at half-pixel locations, from the X block 70. Pixels $C_{r,c}$ (r=0, …, 16;

c=0, …, 16) are pixels interpolated at half-pixel locations, preferably vertically from the A matrix 72 or horizontally from the B matrix 74, but may also be interpolated diagonally from the X matrix 70."), 6:49-6:54 ("The C block 76 is formed by progressive interpolation of preferably ***either*** the A block 72 vertically, or the B block 74 horizontally."), 6:18-48, 8:36-10:50. Accordingly, as set forth in Fandrianto, the highlighted 2/4,2/4 sub-pixels are interpolated by using either the "first weighted sum" of the values of sub-pixels having co-ordinates with K equal to an even value 2 and L equal to zero ("A block") *or* another weighted sum ("second weighted sum") of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value ("B block"),[25] with respective weights in each weighted sum equal to "1." *Id*.

323. It would have been obvious for a POSITA to modify the teachings of TML6 based on the teachings of Fandrianto, for analogous reasons as discussed above in Section IX.A.1, *supra*, such that the ***oneforthpix()*** function interpolates the 2/4,2/4 sub-pixels using a predetermined choice of either the first weighted sum

[25] As I have previously explained above in footnotes 23-24, the combined teachings of TML6 and Fandrianto disclose the claimed feature in the same manner that the embodiments of '273 patent discloses this feature. EX-1001, Figures 14(a)-14(c), 45:25-43 ("$c=(b_1-5b_2+20b_3+20b_4-5b_5+b_6+512)/1024$").

of TML6 (summing values of sub-pixels having co-ordinates with K equal to an even value (two) and L equal to zero) or the second weighted sum of TML6 as modified by Fandrianto (summing values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value (two)).



Specifically, it is expressly taught by Fandrianto that a finite number of interpolation techniques can be used to interpolate the 2/4,2/4 sub-pixels, including vertical interpolation using a first weighted sum (as also used in TML6), and horizontal interpolation using a second weighted sum. Fandrianto, 6:4-12 ("Pixels $A_{r,c}$ (r=0, …,17; c=0, …,16) of the A block 72 are horizontally interpolated at half-pixel locations, from the X block 70. Pixels $B_{r,c}$ (r=0, …, 16; c=0, …, 17) of the B

block 74 are vertically interpolated at half-pixel locations, from the X block 70. Pixels $C_{r,c}$ (r=0, …, 16; c=0, …, 16) are pixels interpolated at half-pixel locations, preferably vertically from the A matrix 72 or horizontally from the B matrix 74, but may also be interpolated diagonally from the X matrix 70."), 6:49-7:2 ("The C block 76 is formed by progressive interpolation of preferably either the A block 72 vertically, or the B block 74 horizontally."). It would, therefore, have been obvious for a POSITA to try such finite options, including using a predetermined choice between the first and second weighted sums, to interpolate the 2/4,2/4 sub-pixels in TML6. Revising the TML6 code to implement this modification would also be straightforward, and a POSITA would reasonably expect this modification to yield the predictable result of interpolating the 2/4,2/4 sub-pixels vertically or horizontally. The '273 patent recognizes that interpolating the 2/4,2/4 sub-pixels "c", as illustrated in Figure 14(a) below, using either a vertical weighted sum of sub-pixels "b" or a horizontal weighted sum of sub-pixels "b" results in the same outcome.

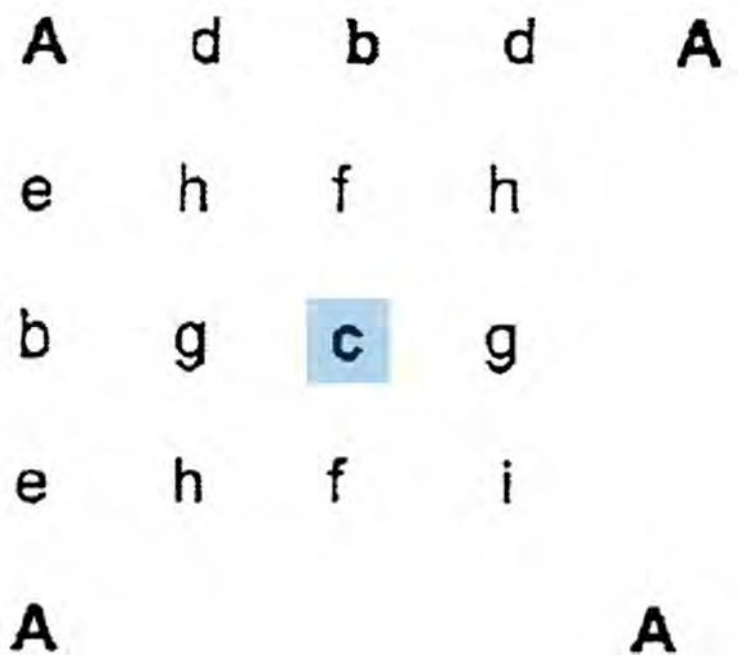


Fig. 14(a)

EX-1001, Figure 14(a), 42:18-52 ("It should be noted that the use of intermediate values b in the horizontal direction leads to the same result as using intermediate values b in the vertical direction."), 45:25-43 (same).

324. For the same reasons that I discuss above for Elements 1[c] and 75[a], TML6 and Fandrianto also disclose circuitry configured to interpolate values for the claimed sub-pixels of Element 75[c]. 1[c], 75[a], *supra*.

10. Claims 76-78

a) Independent Claim 76:

Element 76[pre][a]: "An interpolator for quarter resolution sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 76[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation K/4, L/4, K and L being positive integers having respective values between zero and 4, the interpolator being configured to:"

Element 76[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with K and L equal to 1 or 3, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates 2/4, 2/4, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with K and L equal to zero, 2 or 4, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates 2/4, 2/4 and the nearest neighbouring pixel;"

Element 76[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to 2 and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to 2, used in the interpolation of the sub-pixels

having co-ordinates with K and L equal to 1 or 3, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 76[c]: "interpolate a sub-pixel value for the sub-pixel having co-ordinates with K and L equal to 2, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with K and L equal to 1 or 3, using a predetermined choice of either a weighted sum of the values of the sub-pixel having co-ordinates with K equal to 2 and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the value of the sub-pixel having co-ordinates with K equal to zero and L equal to 2 and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions"

325. Claim 76 is very similar to claim 75, but specifies that N is equal to 2. However, as I explained regarding Elements 75[pre][a]-75[c], TML6 and thus the combination of TML6 and Fandrianto discloses N=2. Thus, for similar reasons as Elements 75[pre][a]-75[c], TML6 and Fandrianto disclose Elements 76[pre][a]-76[c]. *See* Elements 75[pre][a]-75[c], *supra*. Further, as explained for Claim 1, TML6 discloses the case where N=2. Claim 1, *supra*.

b) Claims 77-78

(i) Dependent Claim 77: "A video encoder or still image encoder comprising an interpolator according to claim 76"

(ii) Dependent Claim 78: "A communications terminal comprising a video encoder or still image encoder according to claim 77"

326. TML6, Fandrianto, and VCEG-L45 disclose or render obvious a "video encoder" or a "communications terminal" as recited in claims 77 and 78 for similar reasons as I discussed above for claims 64 and 65. Claims 64-67, 76, *supra*.

11. Independent Claim 82

Element 82[pre][a]: "A computer program for sub-pixel value interpolation, embodied on a computer readable storage medium, the computer program being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 82[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular

degree of sub-pixel value interpolation, the computer program comprising:"

Element 82[a]: "computer program code for interpolating a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 82[b]: "computer program code for interpolating sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 82[c]: "computer program code for interpolating sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates

in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

327. Elements 82[pre][a]-82[c] are directed to a "computer program" of the method and interpolator set forth in Elements 1[pre][a]-1[c] and 33[pre][a]-33[c] and have very similar scope. Accordingly, the combined teachings of TML6 and Fandrianto disclose Elements 82[pre][a]-82[c] for the reasons that I explained above for Elements 1[pre][a]-1[c] and 33[pre][a]-33[c]. Elements 1[pre][a]-1[c], 33[pre][a]-33[c], *supra*. Moreover, a POSITA would have understood that the TML6 source code is designed to be stored on a computer readable storage medium and run on a processor. *Id*.

12. Independent Claim 83

Element 83[pre][a]: "A network apparatus comprising an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 83[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the

co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation, the interpolator being configured to:"

Element 83[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates ½, ½ and the nearest neighbouring pixel;"

Element 83[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 83[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and

L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

328. Elements 83[pre][a]-83[c] are directed to a "network apparatus" and have similar scope as Elements 75[pre][a]-75[c]. Thus, the combination of TML6, Fandrianto, and VCEG-L45 discloses these elements for the same reasons discussed above for Elements 75[pre][a]-75[c]. Elements 75[pre][a]-75[c], *supra*.

329. TML6 and Fandrianto disclose or render obvious an "interpolator" for the reasons I discussed for Claim 33. Claim 33, *supra*. I understand that the term "network apparatus" is part of the preamble and therefore may not be limiting. However, if the Board determines that either of these terms is limiting in some way, TML6, Fandrianto and/or VCEG-L45 disclose or render obvious this feature for the same reasons as a "communications terminal" and a "wireless terminal" recited in claims 66-67 at least because a "communications terminal" and a "wireless terminal" are apparatuses on a wireless network, and is thus each are a network apparatus. Claim 67, *supra*.

B. Ground 2: Claims 32, 52, and 79-81 are Obvious over TML6 in View of Fandrianto and the Knowledge of a POSITA

1. Independent Claim 32:

Element 32[pre][a]: "A method for eighth resolution sub-pixel value interpolation to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 32[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically with the rectangular bounded region using the co-ordinate notation K/8, L/8, K and L being positive integers having respective values between zero and 8, the method comprising causing an apparatus to:"

Element 32[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with K and L equal to 1, 3, 5 or 7, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates 4/8, 4/8, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with K and L equal to zero, 2, 4, 6 or 8, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates 4/8, 4/8 and the nearest neighbouring pixel;"

Element 32[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to

2, 4 or 6 and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L equal to 2, 4 or 6, used in the interpolation of the sub-pixels having co-ordinates with K and L equal to 1, 3, 5 or 7, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 32[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K and L equal to 2, 4 or 6, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with K and L equal to 1, 3, 5 or 7, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to 2, 4 or 6 and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to 2, 4 or 6 and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

330. Claim 32 is substantially the same as Claim 31 but is directed to "eighth resolution" rather than "quarter resolution…interpolation." The difference in resolution is reflected in both the coordinate notations and the sub-pixels recited in Claim 32. As discussed for Element 1[pre][b], TML6—and thus the combined TML6-Fandrianto system—discloses the case where N=2. A POSITA would have been motivated to modify the combined TML6-Fandrianto system described above to achieve eighth resolution sub-pixel value interpolation. Doing so would require

the trivial modification of setting N equal to 3 as opposed to 2, which is disclosed in TML6 as described for Element 1[pre][b] above.

331. As I discuss in the Technology Background, by the ’273 patent’s earliest priority date, sub-pixel interpolation was a well-known technique for enhancing the resolution of digital images that involves defining a finer grid within the original pixel grid to increase the perceived resolution and smoothness of the image. This can be accomplished using a basic form of interpolation known as linear interpolation, in which sub-pixel values are interpolated using the values of pixels or other sub-pixels along a horizontal, vertical, or diagonal line.

332. As I discuss for Element 1[pre][b], TML6 teaches a method of sub-pixel interpolation in which the “particular degree of sub-pixel value interpolation” is 2, i.e., N=2. A value of N=2 represents a grid that is divided into 2^2=4 sub-pixels per pixel in each dimension (horizontally and vertically). This means each original pixel is divided into a 4x4 grid of sub-pixels, resulting in a total of 16 sub-pixels per original pixel.

333. Increasing N to 3 means dividing each pixel into 2^3=8 sub-pixels in each dimension, thus creating an 8x8 grid of sub-pixels, resulting in 64 sub-pixels per original pixel. A POSITA would have known that with more sub-pixels, the interpolation becomes finer, leading to higher resolution and smoother transitions between pixel values. This can significantly improve the quality of the image,

especially in areas with fine details or where there are gradual changes in color or intensity. Accordingly, a POSITA would have known that higher values of N allow for more precise image processing tasks such as edge detection, anti-aliasing, and rendering.

334. Setting N=3 to achieve higher-order interpolation merely reflects the fundamental mathematical concept of increasing the number of interpolating points to enhance the resolution and smoothness of the approximated function. Indeed, the specific order of "3" (i.e., 4 points) was known by the earliest priority date to be desirable. *See, e.g.*, Press, 106-107 ("The number of points (minus one) used in an interpolation scheme is called the *order* of the interpolation. … [A]dding points *close* to the desired point usually does help[.] … **We enthusiastically endorse interpolations with 3 or 4 points[.]**"). Furthermore, it is axiomatic that powers of 2 inherently work well with digital systems, including image processing algorithms, because it aligns well with the binary arithmetic used in such systems. Setting N to 3 leverages the binary nature of digital computing, making computations straightforward and efficient.

335. Although increasing N from 2 to 3 results in 64 sub-pixels per pixel (compared to 16 sub-pixels with N=2) increases the computational load, this would have been manageable with processing power available as of the earliest priority date of the '273 patent. For example, I note that the draft successor to TML6—

TML8, published in July 2001—implemented ⅛ pixel accuracy "[f]or a higher complexity or higher coding efficiency profile." VCEG-N10, 1, 14. A POSITA would have thus known that N=3 struck a good balance between resolution improvement and resource consumption at the relevant time.

336. Finally, a POSITA would have had a reasonable expectation of success modifying the TML6-Fandrianto combination to achieve eighth resolution sub-pixel value interpolation. The combined system would work the same with N set to 3 as it would with N set to 2. Furthermore, as explained above, eighth resolution sub-pixel value interpolation was achievable on hardware available in 2001, making it a practical choice for high-resolution displays and imaging applications as of the '273 patent's earliest priority date.

337. Thus, the teachings and motivations found in the prior art would have led a POSITA to adopt N=3 to achieve eighth pixel resolution, using known techniques to enhance image fidelity efficiently.

2. Dependent Claim 52: "An interpolator according to claim 33, wherein N is set equal to 3.

338. The TML6-Fandrianto combination in view of the knowledge of a POSITA renders obvious the additional limitations of this claim for the reasons discussed for Claims 32-33. Claims 32-33, *supra*.

3. Claims 79-81

a) Independent Claim 79

Element 79[pre][a]: "An interpolator for eighth resolution sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns,"

Element 79[pre][b]: "pixel and sub-pixel locations in the rows and columns being representable mathematically with the rectangular bounded region using the co-ordinate notation K/8, L/8, K and L being positive integers having respective values between zero and 8, the interpolator being configured to:"

Element 79[a]: "interpolate a sub-pixel value for a sub-pixel having co-ordinates with K and L equal to 1, 3, 5 or 7, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates 4/8, 4/8, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with both K and L equal to zero, 2, 4, 6 or 8, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having co-ordinates 4/8, 4/8 and the nearest neighbouring pixel;"

Element 79[b]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to 2, 4 or 6 and L equal to zero and sub-pixels having co-ordinates with K equal to zero and L

equal to 2, 4 or 6, used in the interpolation of the sub-pixels having co-ordinates with both K and L equal to 1, 3, 5 or 7, using weighted sums of the values of pixels located in rows and columns respectively; and"

Element 79[c]: "interpolate sub-pixel values for sub-pixels having co-ordinates with K and L equal to 2, 4 or 6, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with K and L equal to 1, 3, 5 or 7, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to 2, 4 or 6 and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to 2, 4 or 6 and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions."

339. Claim 79 is substantially the same as Claim 33 but specifies "[a]n interpolator for eighth resolution…interpolation" rather than Claim 33's "interpolator for sub-pixel…interpolation." The specified resolution is reflected in both the coordinate notations and the specific sub-pixels recited in Claim 79. For the reasons discussed for Claims 32-33, the TML6-Fandrianto combination in view of the knowledge of a POSITA renders obvious "[a]n interpolator for eighth resolution sub-pixel…interpolation." Claims 32-33, *supra*.

b) Dependent Claim 80: "A video decoder or still image decoder comprising an interpolator according to claim 79."

340. The TML6-Fandrianto combination in view of the knowledge of a POSITA renders obvious the additional limitations of this claim for the reasons discussed for Claims 32-33 and 68-70. Claims 32-33, 68-70, *supra*.

c) Dependent Claim 81: "A communications terminal comprising a video decoder or still image decoder according to claim 80."

341. The TML6-Fandrianto combination in view of the knowledge of a POSITA renders obvious the additional limitations of this claim for the reasons discussed for Claims 32-33 and 68-70. Claims 32-33, 68-70, *supra*.

X. AVAILABILITY FOR CROSS EXAMINATION

342. In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross examination in the case and that cross examination will take place within the United States. If cross examination is required of me, I will appear for cross examination within the United States during the time allotted for cross examination.

XI. SUPPLEMENTATION OF CONCLUSIONS

343. I reserve the right to supplement my conclusions in the future to respond to any arguments that the Patent Owner raises and to take into account new information as it becomes available to me. I may supplement or amend my conclusions, and bases for them, in response to any opinions offered by the Patent Owner's experts. I may also supplement or amend my conclusions, and bases for them, in light of additional evidence, testimony, discovery, arguments on the part of the Patent Owner, or additional information I may become aware of after the date of this declaration. I may also prepare appendices, demonstrative appendices, summaries, tutorials, demonstrations, charts, drawings, tables, or animations to supplement or demonstrate my conclusions as appropriate.