

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CISCO SYSTEMS, INC.,
Petitioner

U.S. Patent No. 8,649,426

**DECLARATION OF DR. IMMANUEL FREEDMAN,
UNDER 37 C.F.R. § 1.68 IN SUPPORT OF PETITION FOR
INTER PARTES REVIEW**

TABLE OF CONTENTS

I.	Introduction.....	4
II.	Qualifications and Professional Experience.....	6
III.	Level of Ordinary Skill in the Art	11
IV.	Relevant Legal Standards	12
V.	Background.....	14
A.	Video Coding	14
B.	Known Challenges	15
C.	Known Solutions	17
VI.	Overview of the '426 Patent.....	19
VII.	Claim Construction.....	22
VIII.	Identification of how the Claims are Unpatentable.....	23
A.	Ground 1: Claims 1-5 and 7-13 are obvious in view of Perlman.	23
1.	Summary of Perlman.....	23
2.	Claim 1	25
3.	Claim 2	64
4.	Claim 3	67
5.	Claim 4	68
6.	Claim 5	69
7.	Claim 7	72
8.	Claim 8	76
9.	Claim 9	79
10.	Claim 10	80

11.	Claim 11	82
12.	Claim 12	83
13.	Claim 13	93
B.	Ground 2: Claim 6 is obvious in view of Perlman and Oguz.	96
1.	Summary of Oguz	96
2.	Reasons to Combine Oguz with Perlman.....	98
3.	Claim 6	100
IX.	Conclusion	103

I, Immanuel Freedman, do hereby declare as follows:

I. INTRODUCTION

1. I am making this declaration at the request of Cisco Systems, Inc. in the matter of the *Inter Partes* Review of U.S. Patent No. 8,649,426 (“the ’426 patent”) to White *et al.*

2. I am being compensated for my work in this matter at my usual hourly rate for this work. I am also being reimbursed for reasonable and customary expenses associated with my work and testimony in this investigation. My compensation is not contingent on the outcome of this matter or the specifics of my testimony.

3. I have been asked to provide my opinions regarding whether claims 1-13 (“the Challenged Claims”) of the ’426 patent are unpatentable as they would have been obvious to a person having ordinary skill in the art (“POSITA”) at the time of the alleged invention, in light of the prior art. It is my opinion that all of the limitations of the Challenged Claims would have been obvious to a POSITA.

4. In the preparation of this declaration, I have studied:

- a. the ’426 patent, Ex.1001;
- b. the prosecution history of the ’426 patent (“’426 File History”), Ex.1002;

- c. U.S. Patent Publication No. 2009/0118017 to Perlman *et al.* (“Perlman”), Ex.1006;
- d. U.S. Patent Publication No. 2008/0151997 to Oguz (“Oguz”), Ex.1008.
5. In forming the opinions expressed below, I have considered:
- the documents listed above;
- the relevant legal standards, including the standard for obviousness, and any additional authoritative documents as cited in the body of this declaration; and
- my own knowledge and experience based upon my work in the field of optical communication as described below, as well as the following materials:
- a. U.S. Patent No. 5,745,181 to Wilkinson (“Wilkinson”), Ex.1005;
- b. U.S. Patent No. 5,729,279 to Fuller (“Fuller”), Ex.1007;
- c. IEEE 100: The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition, 2000, Ex.1009;
- d. U.S. Patent No. 5,553,073 to Barraclough (“Barraclough”), Ex.1014;
- e. U.S. Patent No. 6,665,728 to Graumann (“Graumann”), Ex.1015;
- f. U.S. Patent No. 6,466,248 to Spann (“Spann”), Ex.1016.
6. Unless otherwise noted, all emphasis in any quoted material has been

added.

II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE

7. My qualifications and professional experience are described in my *Curriculum Vitae*, a copy of which can be found in Exhibit 1004. The following is a brief summary of my relevant qualifications and professional experience.

8. I earned a Bachelor of Science degree in Physics from the University of Durham, England, in 1979. I obtained a Doctorate in Physics from the University of Durham, England in 1986. Between obtaining my undergraduate and doctoral degree, I developed a microcomputer system for detecting coalmine fires and heatings as a scientist for the National Coal Board and worked as a software engineer for Laser-Scan Ltd. in Cambridge, England.

9. After obtaining my Doctorate, I served as a Research Assistant at University College London from September 1986 to June 1987, where I developed digital image processing algorithms to improve image and stereo-matching quality for a digital terrain modeling system, including software and algorithms for affine transformation, edge filtering, kriging interpolation, and image stereo-matching with sub-pixel acuity. I continued my work with digital image processing as a Research Associate at the University of Maryland, from June 1987 to September 1988. During my time at the University of Maryland, I designed algorithms for filtering, segmenting, clustering, and path planning based on digital images

organized by quad-tree data structures.

10. From September 1988 to June 1994, I worked as a Senior Systems Engineer for the Hughes STX Corporation. As part of my work, I developed and applied methods for calibration and determination of spacecraft attitude and the comparison of sky maps from the Cosmic Background Explorer (COBE) mission with sky maps from other missions based on scientific data stored in a spatially-referenced database using a quad-tree data structure. In this role, I led the Systems Engineering and end-to-end development of a novel system for compressing imaging and ancillary data that combined scientific modeling with statistical data compression. I was also charged with designing and developing evaluation tools to ensure user-transparent, system-wide compression of a 380-GB dynamic database at an image quality acceptable to end-user scientists. In public recognition of my work, I received National Aeronautics and Space Administration Group Achievement Awards in 1990 and 1992.

11. After June 1994, I began a six-month stint as a contract Software Engineer for the Federal National Mortgage Association in Washington D.C., for which I developed a graphical user interface to monitor and validate loan servicer input for a Loss Mitigation Project. I then served as an Independent Consultant to Optivision, Inc. for the next six months, where I researched and developed rate control algorithms and software based on the MPEG-2 Test Model 5 for the

OPTIVideo™ MPEG-2 video encoder, as well as adaptive quantization algorithms based on the then-JPEG-3 draft standard. In this role, I researched and developed algorithms to improve the quality of gray scale image compression for the medical imaging DICOM Standard by providing a lossless hybrid algorithm encoding image residuals with a diagonal Golomb code based an Enhanced Universal Trellis Coded Quantization algorithm.

12. Between December 1995 and March 1996, I served as a Senior Staff Engineer/Firmware Engineer for General Instrument Inc., Comstream Inc., and Armor Safe Technologies Inc. At Comstream, I worked on integrating an MPEG-2 set top box with OpenTV interactive television middleware programmed in the Microtec C language ported to a Motorola 68340 processor under the pSOS operating system.

13. From January 1996-97, I was the sole proprietor of Anugraha, where I researched and developed algorithms and processes to compress fine art photography at an image quality acceptable to artists based on the JPEG imaging standard implemented with image pre-processing and adaptive quantization. For the next year or so, I worked as an engineering contractor or consultant for various companies, working primarily on image processing systems and digital interactive television set-top boxes.

14. In October 1998, I began a six-month engagement with Rockwell

Collins Inc., where I worked as a Lead Systems Engineer tasked with harmonizing requirements for an MPEG-2 in-flight entertainment system. I then worked for Sun Microsystems Inc. as a Software Engineer until November 1999. During my time at Sun Microsystems Inc., I developed a Distributed Component Object Model (DCOM) software interface between a TV control graphical user interface and a Microsoft broadcast application programming interface (API) with the goal of improving the visual quality of interactive TV displays derived from UDP/IP datagrams synchronized with MPEG-2 audio/video packet data.

15. For the next 22 months, from January 2000 to October 2002, I worked as the Chief Systems Engineer for Media Logic Systems Ltd. During my time at Media Logic Systems, I designed and developed a live interactive television system (iSeeTV) in which customers communicate with human sales agents in video-enabled call centers. To create this system, I researched and developed tools and encoder systems to improve image quality at prescribed latency and bit rate for distributing live video and audio streams encoded via low latency methods. To perform the above, I was required to understand and implement video codec systems employing the MPEG-2 Simple Profile at Main Level (CATV), MPEG-4 Visual Profile with background sprite coding, and the H.263+ Standard (now known as H.264).

16. Since November 2002, I have been an engineering contractor, and

more recently an independent consultant in mathematical modeling, for several companies, such as Cyra Technologies Inc., and Amgen Inc. I also served as a senior research fellow at Merck & Co., Inc., a manager at GlaxoSmithKline Inc., a director at Daiichi Sankyo, Inc., a senior director at Praxis Precision Medicines, and currently serve as a director at Takeda Pharmaceuticals. During this time, I have developed mathematical models and simulations related to various systems, signals, and images. Specifically, I have focused on analyzing, processing, storing, and deriving information from biomedical imaging and other data. Using the information derived from these data, I have created a variety of models related to biology and the effects of drugs on the human body. In recognition of my work, I have received GlaxoSmithKline R&D Recognition Awards in 2012, 2013, and 2016, a Daiichi Sankyo recognition award in 2021, and Takeda recognition awards in 2022 and 2023.

17. In addition to my over thirty years of relevant industry experience, I have authored many publications relating to video and imaging coding. In 2003, I authored a chapter entitled “Video Compression” for the *Internet Encyclopedia*. In 2004 I authored the chapter entitled “Video” for the *Berkshire Encyclopedia of Human-Computer Interaction*. And in 2007 I authored a chapter titled “Video Compression” for the *Handbook of Computer Networks*.

18. I am also a Senior Member of the Institute of Electrical and

Electronics Engineers (“IEEE”) and serve as the current Philadelphia Chapter Chair of the Communications & Information Theory Societies and I am a former Chair of the American Association of Pharmaceutical Scientists Pharmacology Imaging Community. I also served as the 2019 Vice Chair of the IEEE P2673 Intelligence Augmentation for Medical Imaging Standards Working Group. I also have been registered to practice as a patent agent for the United States Patent and Trademark Office since 2002 (Reg. No. 51,704).

19. From 2017, I have also volunteered, first as a Research Scholar, and currently as a Voluntary Researcher, with the State University of New York at Buffalo. In this role, I have provided senior authorship and mentorship for doctoral candidates in areas relating to computer modeling and estimation.

III. LEVEL OF ORDINARY SKILL IN THE ART

20. I understand there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including (1) the levels of education and experience of persons working in the field at the time of the invention; (2) the sophistication of the technology; (3) the types of problems encountered in the field; and (4) the prior art solutions to those problems.

21. A person of ordinary skill in the art (“POSITA”) in the field of the ’426 patent, as of its earliest possible filing date of September 18, 2008, would have been someone knowledgeable and familiar with video codecs. A POSITA

would have been familiar with the encoding and transmission of video data, including MPEG and ITU standards that existed as of the late 2000s. That person would have a bachelor's degree in electrical engineering, computer science, physics, or equivalent training, and approximately two years of experience working in the field of video codecs. Lack of work experience can be remedied by additional education, and vice versa.

22. For purposes of this Declaration, in general, and unless otherwise noted, my statements and opinions, such as those regarding my experience and the understanding of a POSITA generally (and specifically related to the references I consulted herein), reflect the knowledge that existed in the field as of the priority date of the '426 patent. Unless otherwise stated, when I provide my understanding and analysis below, it is consistent with the level of a POSITA prior to the priority date of the '426 patent.

IV. RELEVANT LEGAL STANDARDS

23. I am not an attorney. In preparing and expressing my opinions and considering the subject matter of the '426 patent, I am relying on certain basic legal principles that counsel have explained to me. These principles are discussed below.

24. I understand that prior art to the '426 patent includes patents and printed publications in the relevant art that predate the priority date of the alleged

invention recited in the '426 patent. For purposes of this Declaration, I am applying September 18, 2008 as the earliest possible priority date of the '426 patent.

25. I have been informed that a claimed invention is unpatentable under 35 U.S.C. § 103 if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. I have also been informed by counsel that the obviousness analysis takes into account factual inquiries including the level of ordinary skill in the art, the scope and content of the prior art, and the differences between the prior art and the claimed subject matter.

26. I have been informed by counsel that the Supreme Court has recognized several rationales for combining references or modifying a reference to show obviousness of claimed subject matter. Some of these rationales include the following: (a) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a

reasonable expectation of success; and (f) some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

V. BACKGROUND

A. Video Coding

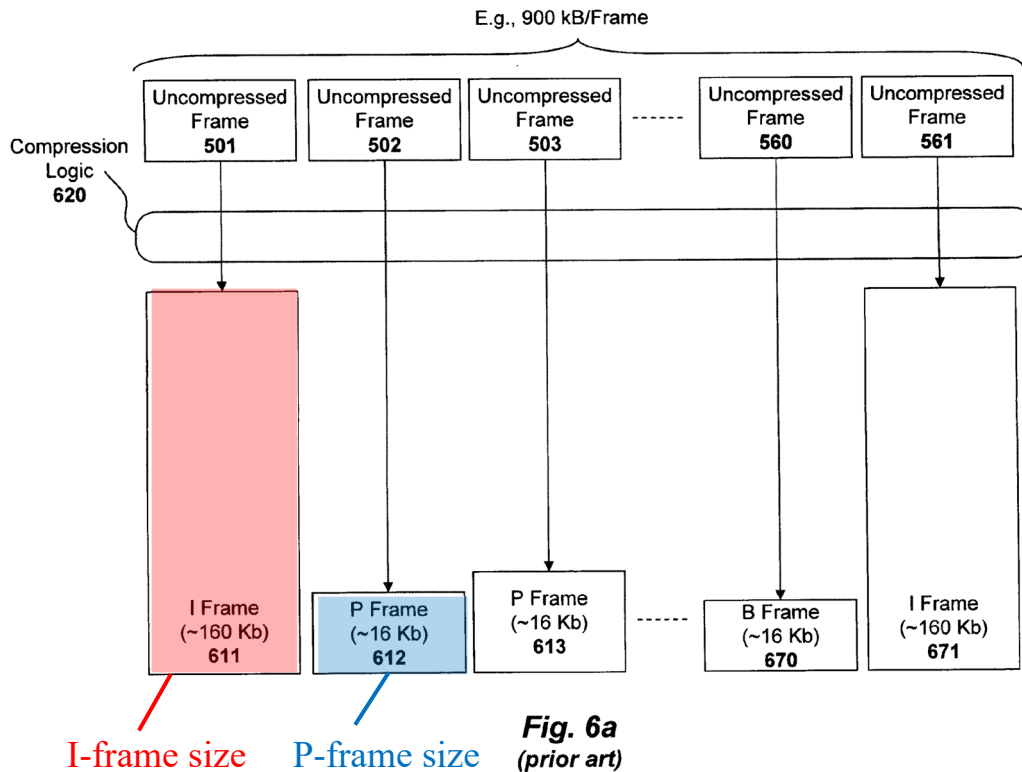
27. Before the '426 patent was filed, it was known to stream video data over the Internet. *See e.g.*, Ex.1006, [0009]-[0010]. Before video data is transmitted over a network, it is typically compressed using a codec, and there were multiple standardized codecs that existed before the '426 patent was filed. Various “types of compression, such as H.264, or Windows Media VC9, MPEG2 and MPEG4 are all more efficient at compressing a video stream because they exploit the similarities between successive frames.” Ex.1006, [0156].

28. A video stream comprises video data that represents a series of frames that make up a video sequence. To encode a video stream, video codecs will designate different frames in different ways. Frames may be designated as reference frames (also referred to as I-frames) or non-reference frames (also referred to as P-frames). In MPEG coding, for example, includes “I-frames which are coded without reference to other frames” and “P-frames which are predictive frames encoded using motion compensation from the previous I-frame.” Ex.1007,

19:46-50. In other words, a reference frame, or I-frame, includes all the data needed to produce a full image. *See* Ex.1006, [0062], [0157]. A P-frame, however, includes data for only the parts of the frame that have changed since a previous frame. *See* Ex.1006, [0062], [0157]. In this manner, the data representing a P-frame is typically much smaller than the data representing an I-frame. I-frames and P-frames were “well understood and documented.” Ex.1005, 1:26.

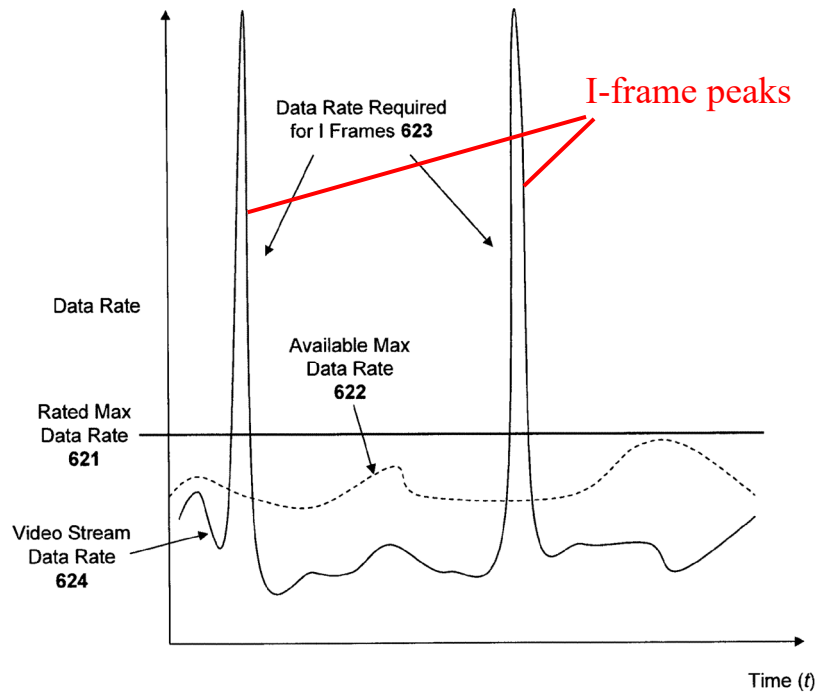
B. Known Challenges

29. While standard compression techniques can significantly reduce the amount of data required to transmit a video stream, they “typically come[] at the cost of higher latency.” Ex.1006, [0062]. This is because an “I frame is typically much larger than a B or P frame (often 10 times larger), and as a result, it takes proportionately longer to transmit at a given data rate.” Ex.1006, [0062]. Fig. 6a illustrates the size of I-frames compared to P-frames in kilobytes (Kb). This concept is illustrated below.



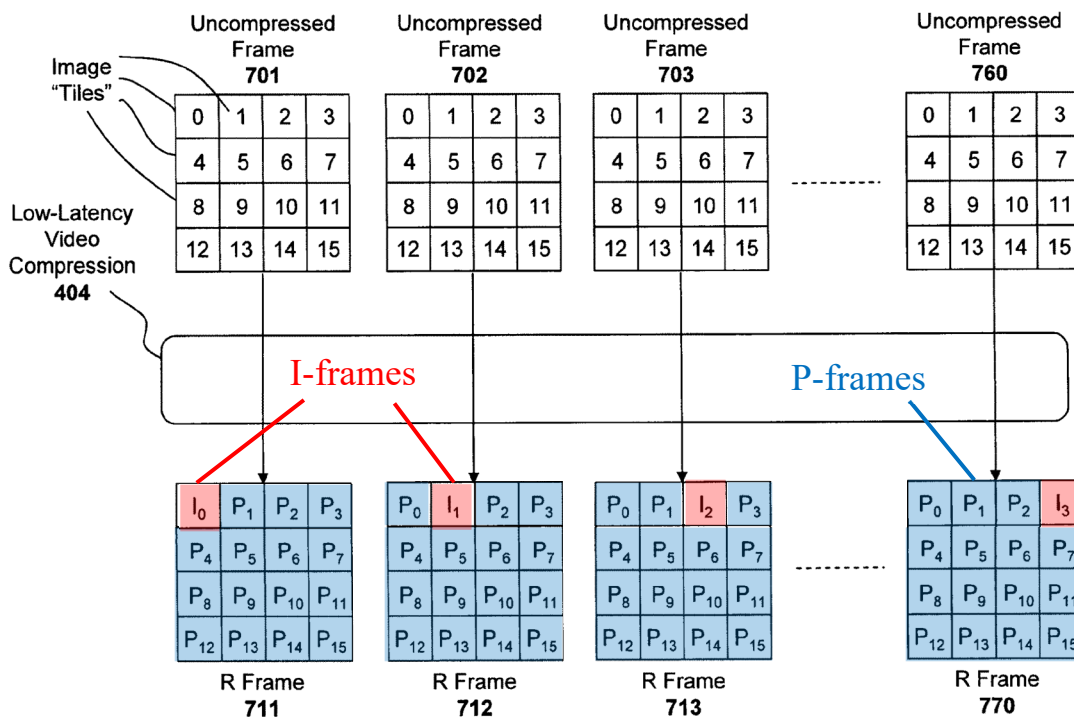
Ex.1006, Fig. 6a (annotated).

30. As a result, the data rate required to transmit a video stream will spike when transmitting the I-frames. “One problem with the foregoing types of compression is that although the average data rate is relatively low (e.g., 1.1 Mbps), a single I frame may take several frame times to transmit.” Ex.1006, [0159]. As illustrated in the figure below, “[t]he video stream data rate 624 is below the available max data rate 621 except for the I frames, where the peak data rate required for the I frames 623 far exceeds the available max data rate 622.” Ex.1006, [0161].

**Fig. 6b****Ex.1006, Fig. 6b (annotated).**

C. Known Solutions

31. One known solution to this problem was to independently encode different segments of a video frame and distribute the I-frames for each segment in time. For example, Perlman describes a technique that “subdivides each uncompressed frame 701-760 into a series of ‘tiles’ and individually encodes each tile as either an I frame or a P frame.” Ex.1006, [0168]. Then, “the I tiles and P tiles for tiles are progressively interleaved in time over successive frames.” Ex.1006, [0169]. This is illustrated below.



Ex.1006, Fig. 7a (annotated).

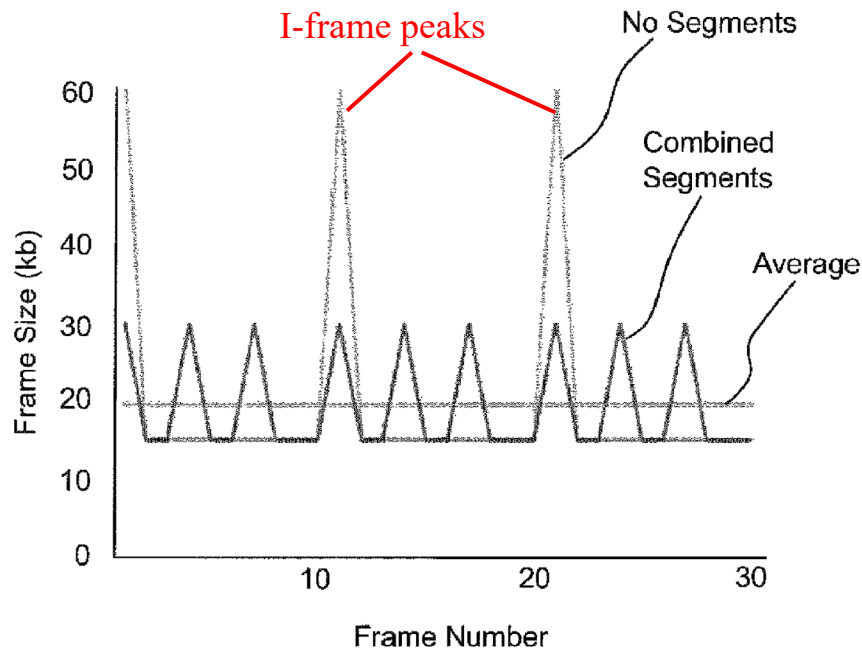
32. As shown in the figure above, an uncompressed image frame (e.g., 701) is divided into a four-by-four grid of 16 tiles. Each tile is then encoded as either an I-frame or a P-frame to produce an overall compressed frame (e.g., 711). In successive compressed frames, different tiles are designated as the I-frame, as shown from Frame 711 to 712 and 712 to 713. Only one tile in each overall compressed frame is designated to be an I-frame. Thus, the I-frames are distributed in time.

33. Thus, before the '426 patent was filed, it was known to encode separate segments of a video frame independently and to interleave the I-frames in time. Doing so was simply a known solution to the known problem of the I-frames

causing spikes in the bandwidth required to transmit video data, thus leading to higher latency.

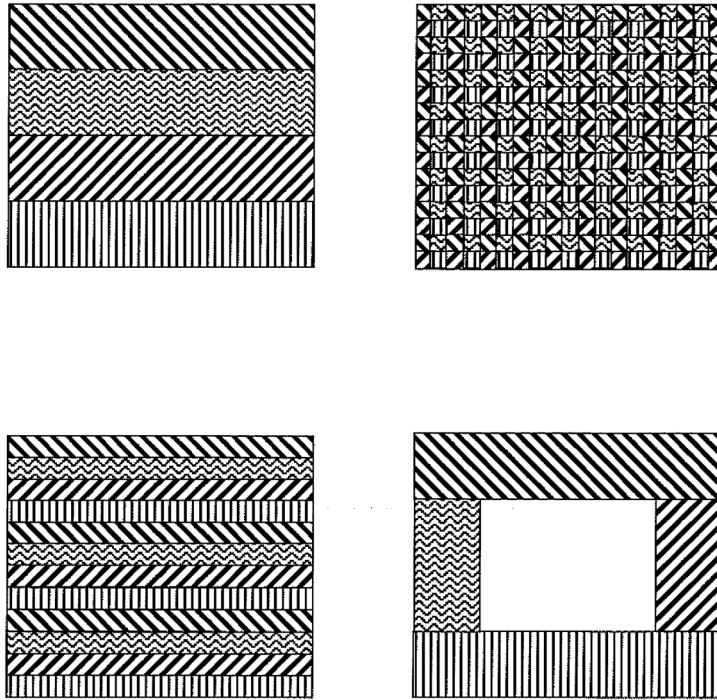
VI. OVERVIEW OF THE '426 PATENT

34. The '426 patent describes and claims nothing more than the known solutions to the known problems described above. The '426 patent explains that one “problem that is associated with modern video encoding techniques is the high peak/average data bursts caused by sending the I-frame via the communication network.” Ex.1001, 3:29-32. The '426 patent illustrates the same I-frame peaks as was known in the art: “Referring now to FIG. 2, it is apparent that for the prior art approach to video encoding the peak to average bit rate ratio of the video bitstream is very high, due to large I-frames, which causes large bursts of traffic on the network and often results in packet loss.” Ex.1001, 5:11-16. “[T]hus, a significant latency results from the databursts 21 unless the available bandwidth is sufficiently high.” Ex.1001, 5:22-24.



Ex.1001, Fig. 2 (annotated).

35. The '426 patent also describes the same solution that was known in the prior art—encoding different segments of the video frame separately, with only one portion encoded as an I-frame at a time. The '426 patent describes and claims “dividing each frame into a plurality of segments” and “encoding the segments independently of one another.” Ex.1001, 3:63-65. The '426 patent provides various examples of how to divide the frame into multiple segments, as shown below.



Ex.1001, Fig. 5 (partial).

36. The '426 patent also describes distributing the I-frames for each of the different segments in time: “This is shown diagrammatically in FIG. 4, where the I-frames for segment (0), segment (1) and segment (2) are offset one from another.” Ex.1001, 6:12-14.

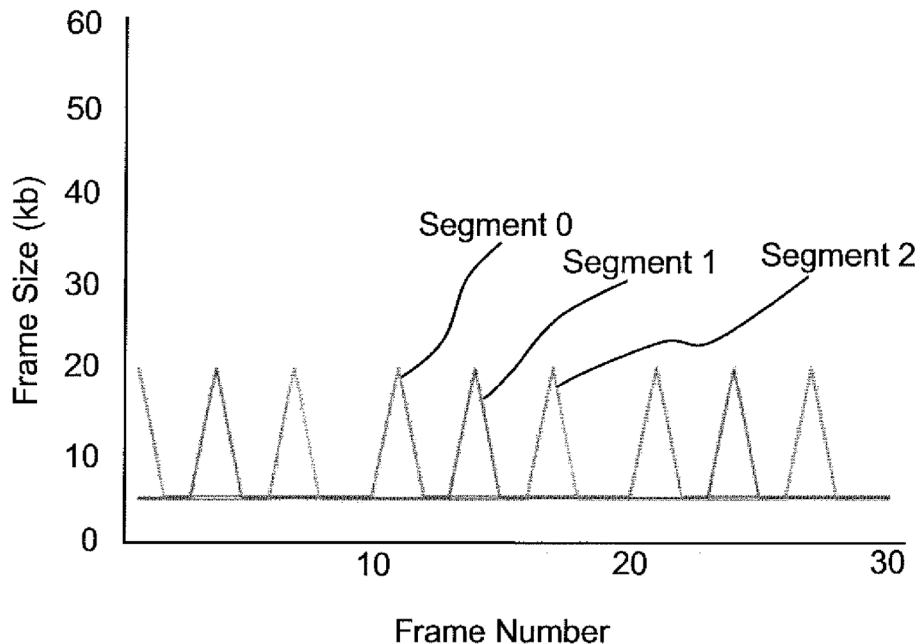


Figure 4

Ex.1001, Fig. 4.

37. However, as I referenced in Section V.C. above and will explain in further detail below, the concept of encoding different segments of a frame independently and distributing the I-frames in time was already known at the time the '426 patent was filed.

VII. CLAIM CONSTRUCTION

38. It is my understanding that in order to properly evaluate the '426 patent, the terms of the claims must first be interpreted. It is my understanding that for the purposes of this *inter partes* review, the claims are to be construed under the so-called *Phillips* standard, under which claim terms are given their ordinary and customary meaning as would be understood by one of ordinary skill in the art

in light of the specification and prosecution history, unless the inventor has set forth a special meaning for a term. I do not believe any of the terms require express construction.

VIII. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

39. I have been asked to provide my opinion as to whether the Challenged Claims of the '426 patent would have been obvious in view of the prior art. The discussion below provides a detailed analysis of how the prior art references identified below teach the limitations of the Challenged Claims of the '426 patent.

40. As part of my analysis, I have considered the scope and content of the prior art and any differences between the alleged invention and the prior art. I describe in detail below the scope and content of the prior art, as well as any differences between the alleged invention and the prior art, on an element-by-element basis for each Challenged Claims of the '426 patent.

41. As described in detail below, the alleged invention of the Challenged Claims would have been obvious in view of the teachings of the identified prior art references as well as the knowledge of a POSITA.

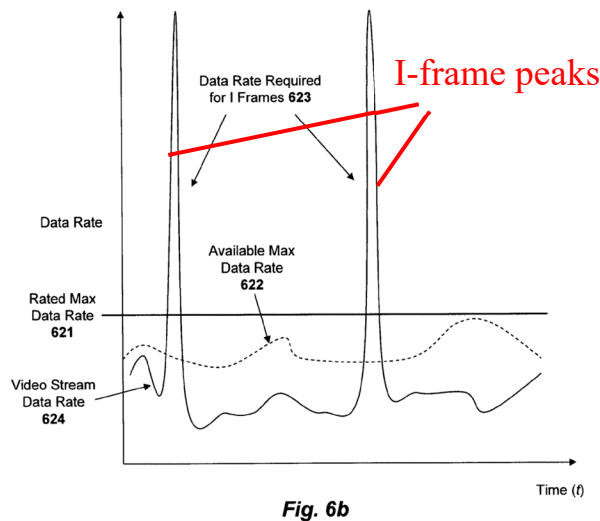
A. Ground 1: Claims 1-5 and 7-13 are obvious in view of Perlman.

1. Summary of Perlman.

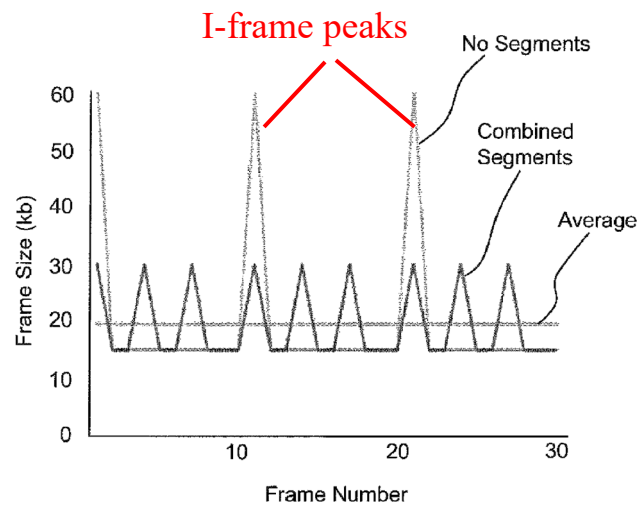
42. Perlman, like the '426 patent, describes a technique for handling the problem of high latency due to I-frame data bursts. Perlman explains that using

standard compression techniques “typically comes at the cost of higher latency.”

Ex.1006, [0062]. This is because an “I frame is typically much larger than a B or P frame (often 10 times larger), and as a result, it takes proportionately longer to transmit at a given data rate.” Ex.1006, [0062]. This concept is illustrated below, along with a similar diagram from the ’426 patent.

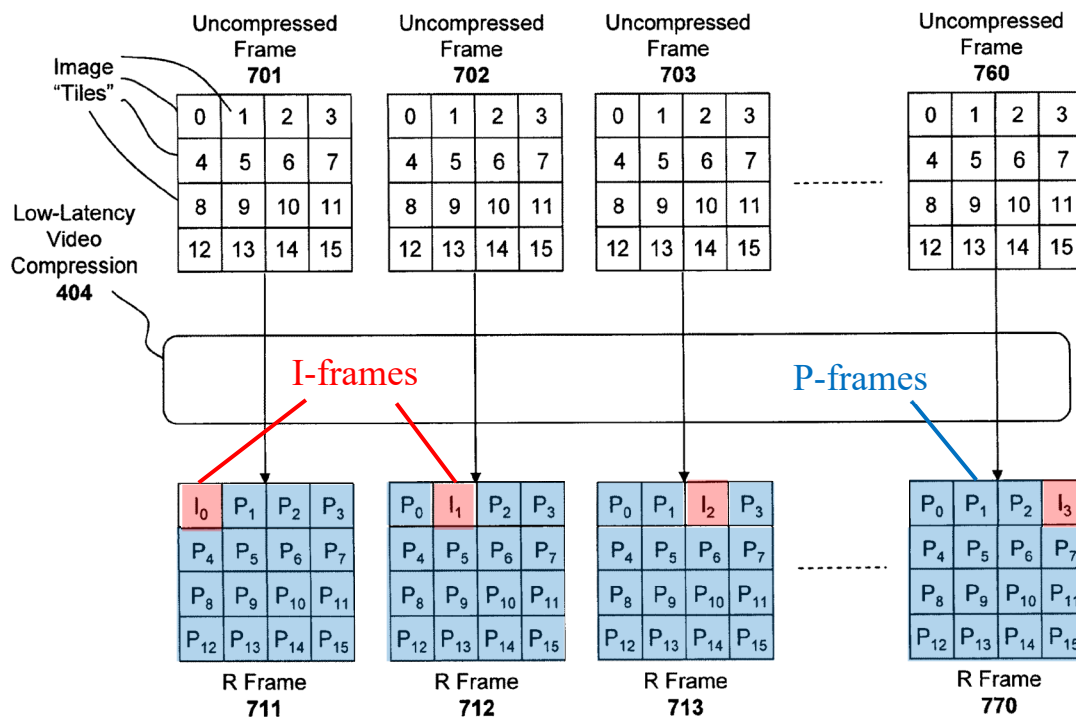


Ex.1006, Fig. 6b (annotated).



Ex.1001, Fig. 2 (annotated).

43. Like the ’426 patent, Perlman describes a technique that “subdivides each uncompressed frame 701-760 into a series of ‘tiles’ and individually encodes each tile as either an I frame or a P frame.” Ex.1006, [0168]. Then, “the I tiles and P tiles for tiles are progressively interleaved in time over successive frames.” Ex.1006, [0169]. This is illustrated below.



Perlman, Fig. 7 (annotated).

44. Thus, Perlman also provides evidence that it was known to encode separate segments of a video frame independently and distribute the I-frames in time. Again, doing so was simply a known solution to the known problem of the I-frames causing spikes in the bandwidth required to transmit video data, thus leading to higher latency.

2. Claim 1

[1.0] A method of encoding in real time a video signal that is associated with a videoconference or telepresence session, the method comprising:

45. **First**, Perlman describes a method for compressing and transmitting video streams in real time (“*encoding in real time a video signal*”).

To accomplish the foregoing goals, one embodiment takes a new approach to video compression which decreases the latency and the peak bandwidth requirements for transmitting video.

Prior to the description of these embodiments, an analysis of current video compression techniques will be provided with respect to FIG. 5 and FIGS. 6 a-b. Of course, these techniques may be employed in accordance with underlying principles if the user is provided with sufficient bandwidth to handle the data rate required by these techniques. Note that audio compression is not addressed herein other than to state that it is implemented simultaneously and in synchrony with the video compression. Prior art audio compression techniques exist that satisfy the requirements for this system.

Ex.1006, [0153].

Many PCs and mobile phones are equipped with video cameras and have the capability to do real-time video compression,

particularly when the image is small. Also, small cameras are available that can be attached to a television, and it is not difficult to implement real-time compression either in software or using one of many hardware compression devices to compress the video. Also, many PCs and all mobile phones have microphones, and headsets are available with microphones.

Ex.1006, [0258].

46. A POSITA would have understood that compressing video data using a codec is “encoding” the video data as claimed. Indeed, Perlman expressly correlates the two words “encodes” and “compresses” with one another:

In one embodiment, **the low-latency compression logic 404 divides up the video frame into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame** (i.e., the tile is compressed as if it is a separate video frame of 1/16th the size of the full image, and the compression used for this “mini” frame is I frame compression) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively. With each successive video frame, the tile to be encoded as an I tile is changed. Thus, in a given frame time, only one tile of the tiles in the video frame is an I tile, and the remainder of the tiles are P tiles. For example, in FIG. 7 a, tile 0 of uncompressed frame 701 is encoded as I tile I₀ and the remaining 1-15 tiles are encoded as P tiles P₁ through P₁₅ to produce R frame 711. In the next uncompressed video frame 702, tile 1 of uncompressed frame 701 is encoded as I tile I₁ and the remaining tiles 0 and 2 through 15 are encoded as P tiles, P₀ and P₂ through P₁₅, to produce R frame 712. Thus, the I tiles and P tiles for tiles are progressively interleaved in time over successive frames. The process continues until an R tile 770 is generated with the last tile in the matrix encoded as an I tile (i.e., I₁₅). The process then starts over, generating another R frame such as frame 711 (i.e., encoding an I tile for tile 0) etc. Although not illustrated in FIG. 7 a, in one embodiment, the first R frame of the video sequence of R frames contains only I tiles (i.e., so that subsequent P frames have reference image data from which to calculate motion). Alternatively, in one embodiment, the startup sequence uses the same I tile pattern as normal, but does not include P tiles for those tiles that have not yet been

encoded with an I tile. In other words, certain tiles are not encoded with any data until the first I tile arrives, thereby avoiding startup peaks in the video stream data rate 934 in FIG. 9 a, which is explained in further detail below. Moreover, as described below, various different sizes and shapes may be used for the tiles while still complying with these underlying principles.

Ex.1006, [0169].

47. **Second**, Perlman explains that its method can be used for video conferencing. While much of Perlman's disclosure is in the context of video game streaming applications, Perlman describes that its techniques for video compression also apply to video conference streaming ("*a video signal that is associated with a videoconference or telepresence session*"):

Another feature of the hosting service 210 is the ability to for multiple users to collaborate while viewing live video, even if using widely disparate devices for viewing. This is useful both when playing games and **when using applications.**

Ex.1006, [0257].

Many PCs and mobile phones are equipped with video cameras and have the capability to do real-time video compression, particularly when the image is small. **Also, small cameras are available that can be attached to a television, and it is not difficult to implement real-time compression either in software or using one of many hardware compression devices to compress the video.** Also, many PCs and all

mobile phones have microphones, and headsets are available with microphones.

Ex.1006, [0258].

Such cameras and/or microphones, combined with local video/audio compression capability (**particularly employing the low latency video compression techniques described herein**) will enable a user to transmit video and/or audio from the user premises 211 to the hosting service 210, together with the input device control data. When such techniques are employed, then a capability illustrated in FIG. 19 is achievable: a user can have his video and audio 1900 appear on the screen within another user's game or application. This example is a multiplayer game, where teammates collaborate in a car race. **A user's video/audio could be selectively viewable/hearable only by their teammates. And, since there would be effectively no latency, using the techniques described above the players would be able to talk or make motions to each other in real-time without perceptible delay.**

Ex.1006, [0259].

The example of FIG. 19 shows how such collaboration is used in a game, **but such collaboration can be an immensely powerful tool for applications.** Consider a situation where a large building is being designed for New York city by architects in Chicago for a real estate developer based in New York, but the decision involves a financial investor who is traveling and happens to be in an airport in Miami, and a decision needs to be made about certain design elements of the building in terms of how it fits in with the buildings near it, to satisfy both the investor and the real estate developer. **Each party will hear**

the voice of the other parties (the conference calling will be handled by any of many widely available conference calling software package in the app/game server(s) 1521-1525) and, through actuation of a button on a user input device, a user will be able to make video appear of themselves using their local camera. As the meeting proceeds, the architects will be able to show what the build looks like as they rotate it and fly by it next to the other building in the area, with extremely photorealistic 3D rendering, and the same video will be visible to all parties, at the resolution of each party's display device. **Still, for most of the purposes of the meeting, where the investor is watching the architects control the building fly-by or for talking of video teleconferencing, even cellular latency should be acceptable.**

Ex.1006, [0261]; *see also* [0262]-[0263].

48. Perlman illustrates in Figure 19 a videogame screen that includes a videoconference view of the user “Player MASTERDAVE.” The videogame screen itself is “*a video signal that is associated with a videoconference*”:



Videoconference video of user Player MASTERDAVE

Perlman, Fig. 19 (annotated).

49. Perlman thus explicitly states that video conferencing systems, whether in game or other applications, can benefit from Perlman's compression techniques, "particularly employing the low latency video compression techniques described herein." Ex.1006, [0259]; *See also* [0261]-[0263]. In addition, a POSITA would have found it obvious to apply Perlman's low latency compression techniques to video conferencing because reducing latency improves personal interactions in a real-time collaboration. *See e.g.*, Ex.1014, 2:7-10 (low latency is desired "in order for a natural two way conversation to be feasible"); Ex.1016, 1:32-33 ("During real time videoconferencing, longer latency is considered a negative"). Indeed, if the latency is high, then "it may be difficult for parties to

interrupt another in a manner that resembles a normal, in-person conference.”

Ex.1015, 1:53-59.

50. Thus, because Perlman describes a method for compressing video data in real time, and that video stream can be used for real time video conferencing, Perlman renders obvious a “*method of encoding in real time a video signal that is associated with a videoconference or telepresence session*” as recited in the claim.

[1.1] capturing a stream of video data comprising a series of sequential video frames;

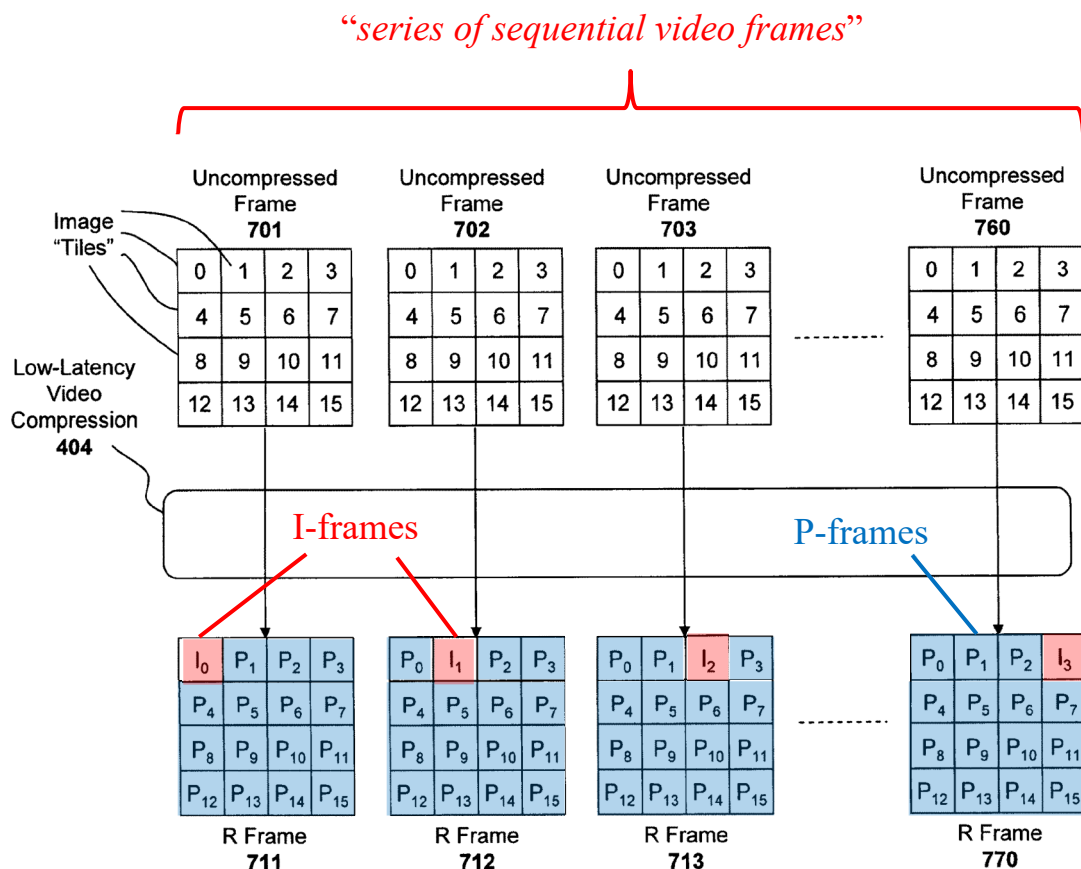
51. Perlman describes a process by which video compression logic 404 receives (i.e., “*capture[s]*”) graphical and audio output from application software (“*stream of video data*”).

The graphical and audio output of the executed game or **application software—i.e., new sequences of video images—are provided to the low-latency video compression logic 404 which compresses the sequences of video images according to low-latency video compression techniques, such as those described herein and transmits a compressed video stream**, typically with compressed or uncompressed audio, back to the client 415 over the Internet 410 (or, as described below, over an optimized high speed network service that bypasses the general Internet). Low-latency video decompression logic 412 on the client 415 then decompresses the video and audio streams and renders the decompressed video stream, and typically plays the decompressed audio stream, on a display device 422 Alternatively, the audio can be played on speakers separate from the display device 422

or not at all. Note that, despite the fact that input device 421 and display device 422 are shown as free-standing devices in FIGS. 2 a and 2 b, they may be integrated within client devices such as portable computers or mobile devices.

Ex.1006, [0117].

52. Second, consistent with how video streams work, Perlman describes that the received video stream comprises a sequence of uncompressed frames 701, 702, 703 (“a series of sequential video frames”). Ex.1006, [0169]. Fig. 7a of Perlman illustrates a sequence of frames, as shown below.



Ex.1006, Fig. 7a (annotated).

The low-latency compression logic 404 employed in one embodiment uses several different techniques to address the range of problems with streaming low-latency compressed video, while maintaining high quality. First, the low-latency compression logic 404 generates only I frames and P frames, thereby alleviating the need to wait several frame times to decode each B frame. In addition, as illustrated in FIG. 7 a, in one embodiment, the low-latency compression logic 404 subdivides each uncompressed frame 701-760 into a series of “tiles” and individually encodes each tile as either an I frame or a P frame. The group of compressed I frames and P frames are referred to herein as “R frames” 711-770. In the specific example shown in FIG. 7 a, each uncompressed frame is subdivided into a 4×4 matrix of 16 tiles. However, these underlying principles are not limited to any particular subdivision scheme.

Ex.1006, [0168].

53. Thus, because Perlman describes receiving a stream of video data that includes a sequence of video images, Perlman renders obvious “*capturing a stream of video data comprising a series of sequential video frames*” as claimed.

[1.2] *separating the video frames into a plurality of frame segments, wherein each frame segment represents a part of a complete video frame;*

54. First, Perlman’s technique divides the video frames into a plurality of tiles (“*separating the video frames into a plurality of frame segments*”).

The low-latency compression logic 404 employed in one embodiment uses several different techniques to address the range of problems with

streaming low-latency compressed video, while maintaining high quality. First, the low-latency compression logic 404 generates only I frames and P frames, thereby alleviating the need to wait several frame times to decode each B frame. In addition, as illustrated in FIG. 7 a, in one embodiment, **the low-latency compression logic 404 subdivides each uncompressed frame 701-760 into a series of “tiles” and individually encodes each tile as either an I frame or a P frame.** The group of compressed I frames and P frames are referred to herein as “R frames” 711-770. In the specific example shown in FIG. 7 a, each uncompressed frame is subdivided into a 4×4 matrix of 16 tiles. However, these underlying principles are not limited to any particular subdivision scheme.

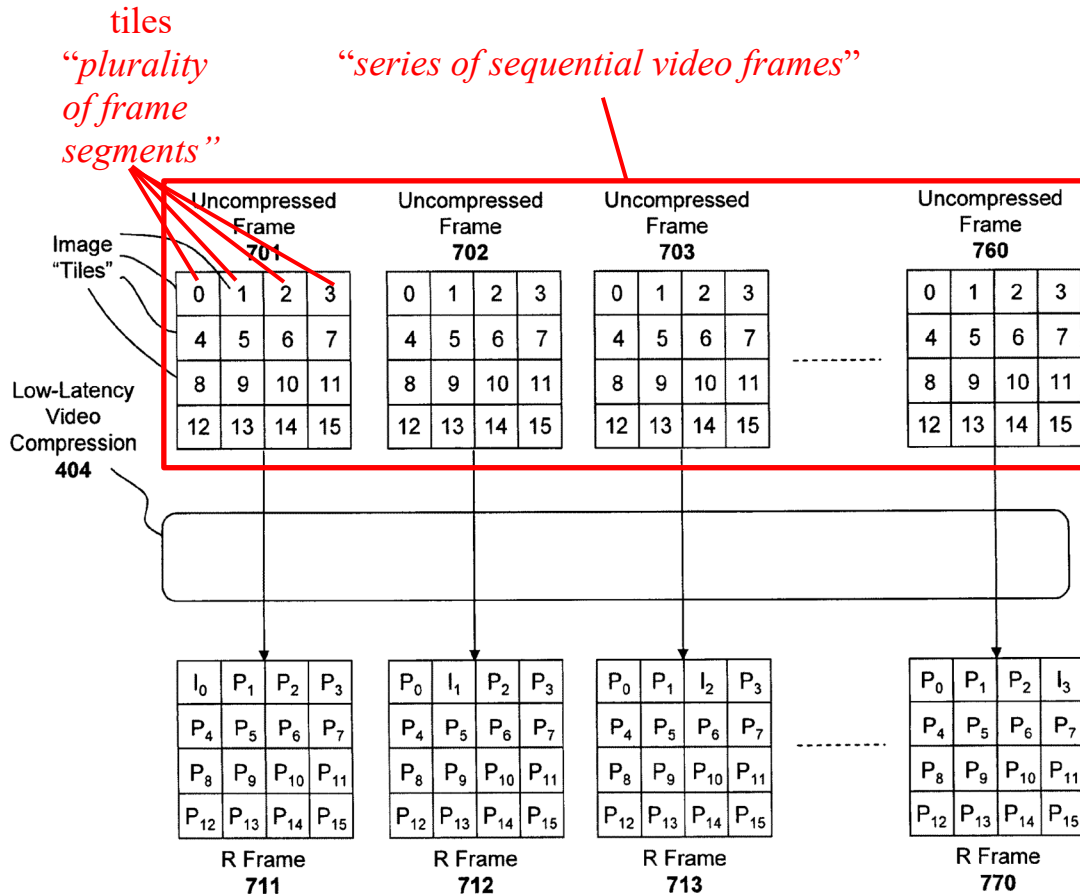
Ex.1006, [0168].

55. Second, Perlman also explains that each tile represents 1/16th of the full image (“*each frame segment represents a part of a complete video frame*”).

In one embodiment, the low-latency compression logic 404 divides up the video frame **into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame (i.e., the tile is compressed as if it is a separate video frame of 1/16th the size of the full image,** and the compression used for this “mini” frame is I frame compression) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively. With each successive video frame, the tile to be encoded as an I tile is changed. Thus, in a given frame time, only one tile of the

tiles in the video frame is an I tile, and the remainder of the tiles are P tiles. For example, in FIG. 7 a, tile 0 of uncompressed frame 701 is encoded as I tile I_0 and the remaining 1-15 tiles are encoded as P tiles P_1 through P_{15} to produce R frame 711. In the next uncompressed video frame 702, tile 1 of uncompressed frame 701 is encoded as I tile I_1 and the remaining tiles 0 and 2 through 15 are encoded as P tiles, P_0 and P_2 through P_{15} , to produce R frame 712. Thus, the I tiles and P tiles for tiles are progressively interleaved in time over successive frames. The process continues until an R tile 770 is generated with the last tile in the matrix encoded as an I tile (i.e., I_{15}). The process then starts over, generating another R frame such as frame 711 (i.e., encoding an I tile for tile 0) etc. Although not illustrated in FIG. 7 a, in one embodiment, the first R frame of the video sequence of R frames contains only I tiles (i.e., so that subsequent P frames have reference image data from which to calculate motion). Alternatively, in one embodiment, the startup sequence uses the same I tile pattern as normal, but does not include P tiles for those tiles that have not yet been encoded with an I tile. In other words, certain tiles are not encoded with any data until the first I tile arrives, thereby avoiding startup peaks in the video stream data rate 934 in FIG. 9 a, which is explained in further detail below. Moreover, as described below, various different sizes and shapes may be used for the tiles while still complying with these underlying principles.

Ex.1006, [0169].



Ex.1006, Fig. 7a (annotated).

56. Thus, because Perlman describes dividing the frames into a plurality of tiles, each tile representing a part of a full image, Perlman renders obvious “separating the video frames into a plurality of frame segments, wherein each frame segment represents a part of a complete video frame” as claimed.

[1.3] encoding each frame segment independently and in parallel to generate parallel encoded data streams, one for each of the plurality of frame segments,

57. First, Perlman discloses encoding each tile individually and in parallel using independent processes (“independently and in parallel”).

The low-latency compression logic 404 employed in one embodiment uses several different techniques to address the range of problems with streaming low-latency compressed video, while maintaining high quality. First, the low-latency compression logic 404 generates only I frames and P frames, thereby alleviating the need to wait several frame times to decode each B frame. In addition, as illustrated in FIG. 7 a, in one embodiment, **the low-latency compression logic 404 subdivides each uncompressed frame 701-760 into a series of “tiles” and individually encodes each tile as either an I frame or a P frame. The group of compressed I frames and P frames are referred to herein as “R frames” 711-770.** In the specific example shown in FIG. 7 a, each uncompressed frame is subdivided into a 4×4 matrix of 16 tiles. However, these underlying principles are not limited to any particular subdivision scheme.

Ex.1006, [0168].

58. Perlman refers to the collection of I-frame and P-frame tiles corresponding to an input video image as an “R frame.” Perlman explains that each of the tiles within an R frame are encoded independently and in parallel. Compressing each tile may be done using multi-core processors, FPGAs, or other components. *See* Ex.1006, [0212]. As an example, Perlman explains that a multi-core processor may be used such that each core implements the coding as “an independent process.”

FIG. 12 illustrates one particular embodiment in which a multi-core and/or multi-processor 1200 is used to compress 8 tiles in

parallel. In one embodiment, a dual processor, quad core Xeon CPU computer system running at 2.66 GHz or higher is used, with each core implementing the open source x264 H.264 compressor **as an independent process**. However, **various other hardware/software configurations may be used while still complying with these underlying principles. For example, each of the CPU cores can be replaced with an H.264 compressor implemented in an FPGA**. In the example shown in FIG. 12, cores 1201-1208 are used to concurrently process the I tiles and P tiles as eight independent threads. As is well known in the art, current multi-core and multi-processor computer systems are inherently capable of multi-threading when integrated with multi-threading operating systems such as Microsoft Windows XP Professional Edition (either 64-bit or the 32-bit edition) and Linux.

Perlman, [212].

In the embodiment illustrated in FIG. 12, **since each of the 8 cores is responsible for just one tile, it operates largely independently from the other cores, each running a separate instantiation of x264¹**. A PCI Express x1-based DVI capture card, such as the Sendero Video Imaging IP Development Board from Microtronix of Oosterhout, The Netherlands is used to capture uncompressed video at 640×480, 800×600, or 1280×720 resolution, and the FPGA on the card uses

¹ A POSITA would have understood, and Perlman explains, that x264 is an open source software library for compressing video. Ex.1006, [0156].

Direct Memory Access (DMA) to transfer the captured video through the DVI bus into system RAM. The tiles are arranged in a 4×2 arrangement 1205 (although they are illustrated as square tiles, in this embodiment they are of 160×240 resolution). Each instantiation of x264's is configured to compress one of the 8 160×240 tiles, and they are synchronized such that, after an initial I tile compression, each core enters into a cycle, each one frame out of phase with the other, to compress one I tile followed by seven P tiles, and illustrated in FIG. 12.

Perlman, [213].

59. Accordingly, Perlman's technique may use various hardware or software configurations for encoding each of the tiles "*independently and in parallel*" as claimed. A POSITA would have found it obvious to implement Perlman's various techniques—including the techniques described with respect to Figure 7a—using parallel processors (or other suitable hardware) to encode different tiles in parallel as Perlman describes with respect to Fig. 12. A POSITA would have found this beneficial because encoding the separate tiles in parallel—as opposed to in series—maintains Perlman's goal of reducing latency. *See* Ex.1006, [0169].

60. Second, the output of Perlman's parallel processors (or other type of encoders) produce "*parallel encoded data streams, one for each of the plurality of frame segments*" as illustrated in Fig. 12 below.

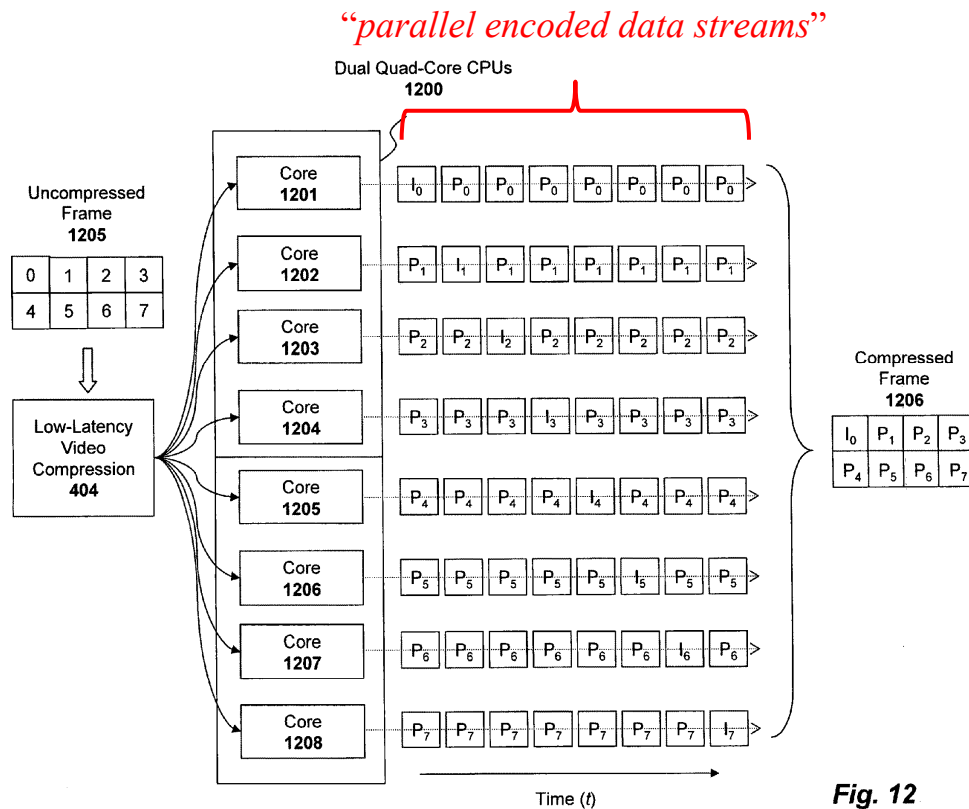
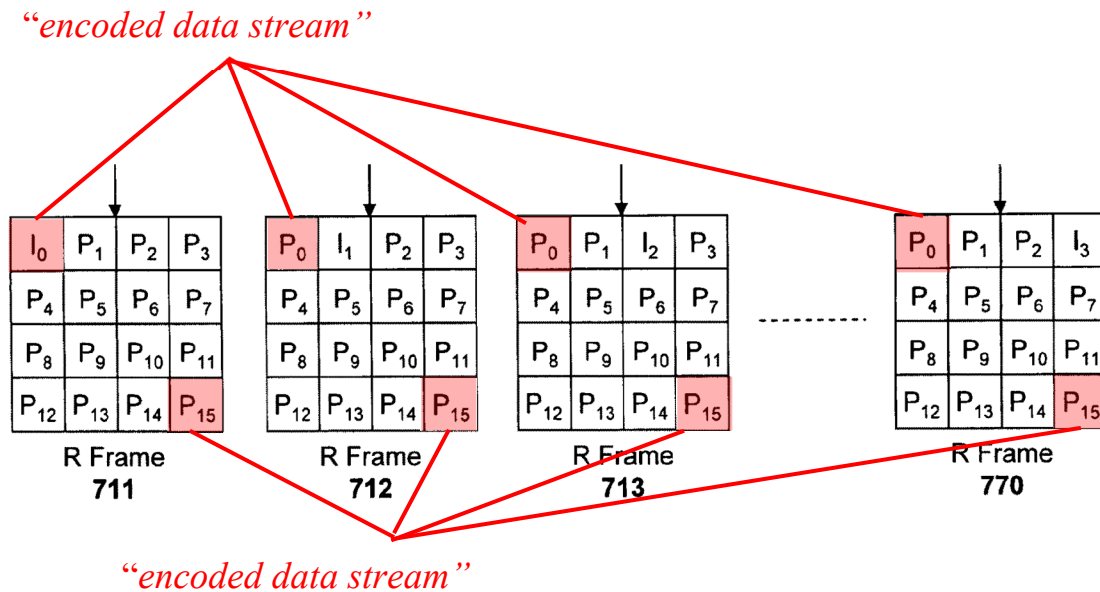


Fig. 12

Ex.1006, Fig. 12 (annotated).

61. Referring to the example in Fig. 7a (annotated below), one encoded data stream is represented by the tiles at the top left position (position “0”). Another encoded data stream is represented by the tiles at the bottom right position (position “15”). The tiles at each position thus represent “*parallel encoded data streams*” as claimed.



Ex.1006, Fig. 7a (partial, annotated).

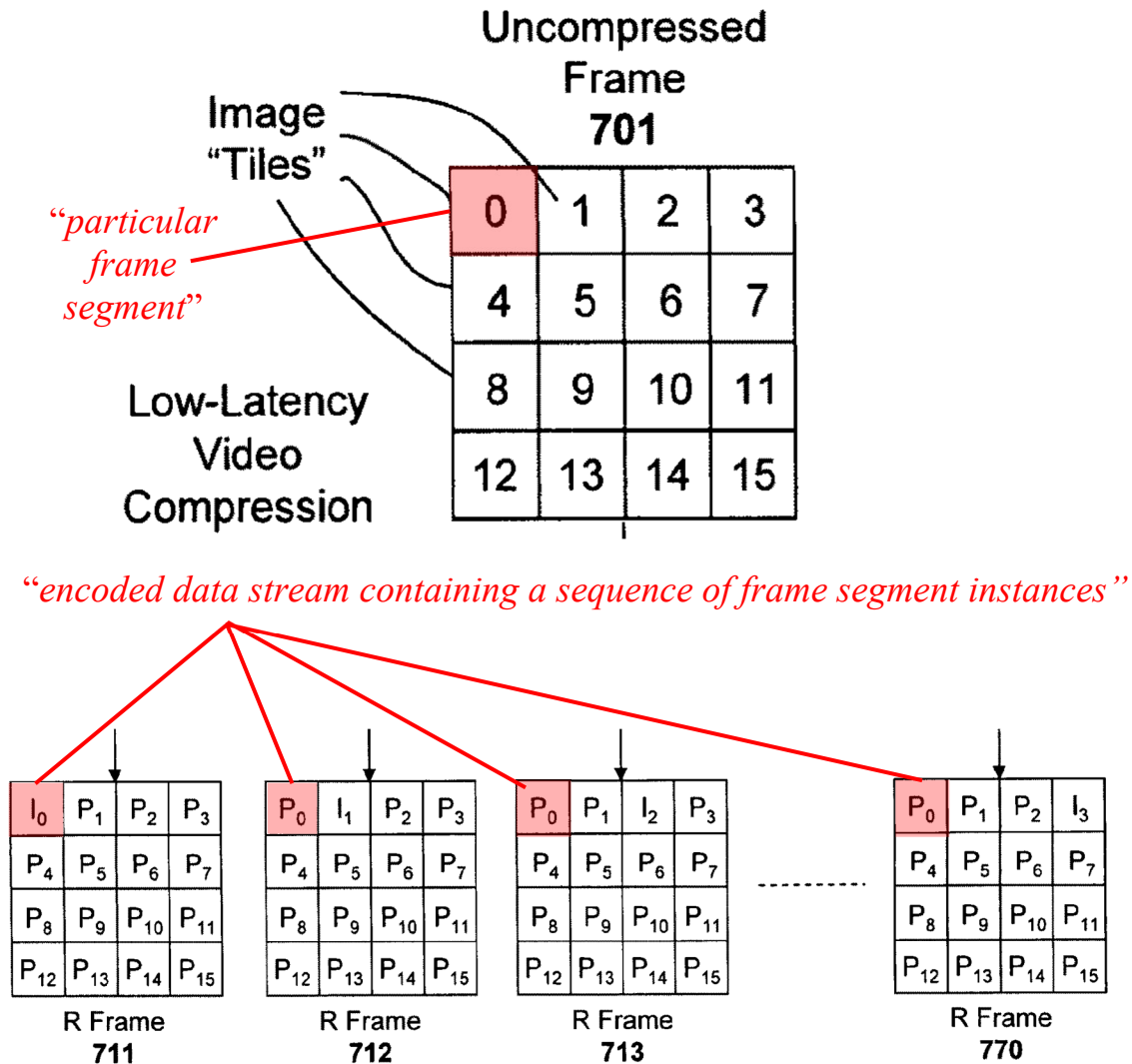
62. Thus, because Perlman describes individually encoding each tile in parallel as an independent process to produce parallel encoded streams, Perlman renders obvious “encoding each frame segment independently and in parallel to generate parallel encoded data streams, one for each of the plurality of frame segments” as claimed.

[1.4] each encoded data stream containing a sequence of frame segment instances each corresponding to a particular frame segment in the video frames, and

63. First, as explained above at [1.3], Perlman explains that each tile is encoded separately and in parallel to form “parallel encoded data streams.”

64. Second, Perlman discloses that each encoded data stream corresponds to a series of tile instances. This concept is further illustrated in Fig. 7a. As shown

in the annotated figure below, the sequence of tile instances for the top left tile (“a particular frame segment”) represent an “encoded data stream containing a sequence of frame segment instances.”



Ex.1006, Fig. 7a (partial, annotated).

65. Fig. 12 also illustrates the sequences of frame segment instances for the encoded data streams.

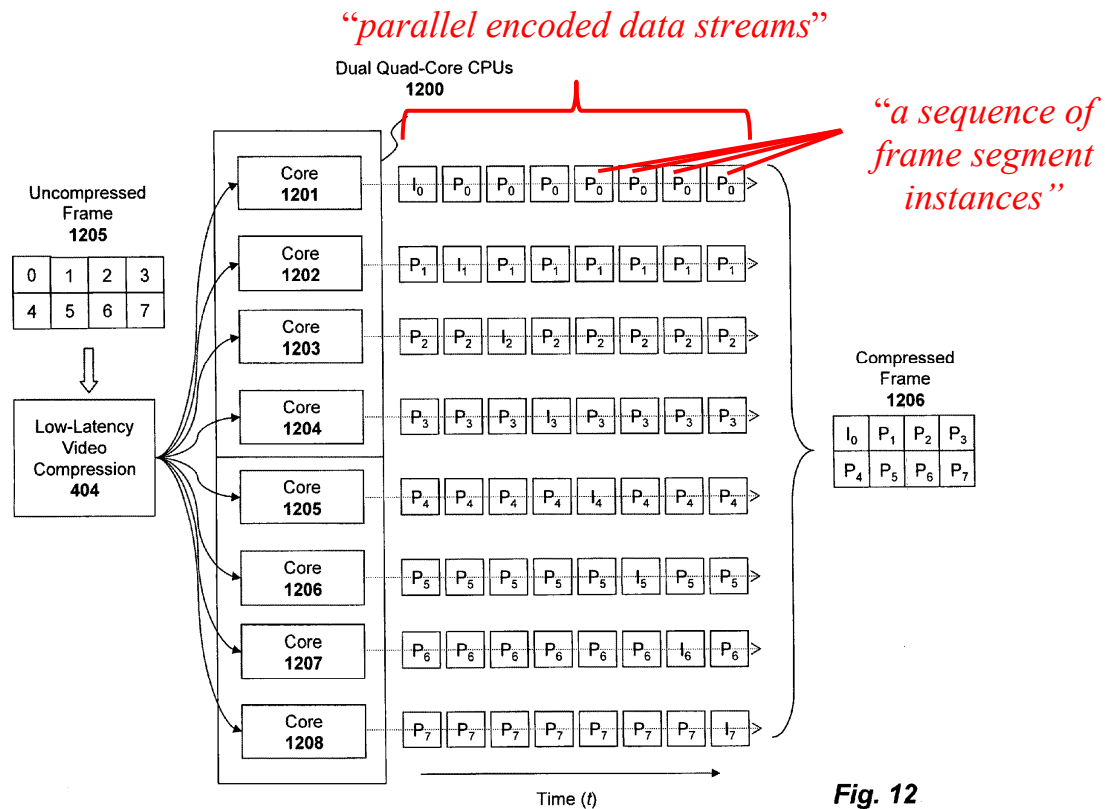


Fig. 12

Ex.1006, Fig. 12 (annotated).

In one embodiment, the low-latency compression logic 404 divides up the video frame into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame (i.e., the tile is compressed as if it is a separate video frame of 1/16th the size of the full image, and the compression used for this “mini” frame is I frame compression) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively. With each successive video frame, the tile to be encoded as an I tile is changed. Thus, in a given frame time, only one tile of the tiles in the video frame is an I tile, and the remainder of the tiles are P tiles. For example, in FIG.

7 a, tile 0 of uncompressed frame 701 is encoded as I tile I_0 and the remaining 1-15 tiles are encoded as P tiles P_1 through P_{15} to produce R frame 711. In the next uncompressed video frame 702, tile 1 of uncompressed frame 701 is encoded as I tile I_1 and the remaining tiles 0 and 2 through 15 are encoded as P tiles, P_0 and P_2 through P_{15} , to produce R frame 712. Thus, the I tiles and P tiles for tiles are progressively interleaved in time over successive frames. The process continues until an R tile 770 is generated with the last tile in the matrix encoded as an I tile (i.e., I_{15}). The process then starts over, generating another R frame such as frame 711 (i.e., encoding an I tile for tile 0) etc. Although not illustrated in FIG. 7 a, in one embodiment, the first R frame of the video sequence of R frames contains only I tiles (i.e., so that subsequent P frames have reference image data from which to calculate motion). Alternatively, in one embodiment, the startup sequence uses the same I tile pattern as normal, but does not include P tiles for those tiles that have not yet been encoded with an I tile. In other words, certain tiles are not encoded with any data until the first I tile arrives, thereby avoiding startup peaks in the video stream data rate 934 in FIG. 9 a, which is explained in further detail below. Moreover, as described below, various different sizes and shapes may be used for the tiles while still complying with these underlying principles.

Ex.1006, [0169].

66. Thus, because Perlman describes dividing a frame into tiles, each tile including a series of tile instances, Perlman renders obvious “*each encoded data stream containing a sequence of frame segment instances each corresponding to a*

particular frame segment in the video frames” as claimed.

[1.5] each encoded data stream containing I-frame segment instances that carry complete information about a particular frame segment and P-frame segment instances that carry partial information about the particular frame segment, the partial information representing information that has changed relative to a previous frame segment instance in that encoded data stream, and

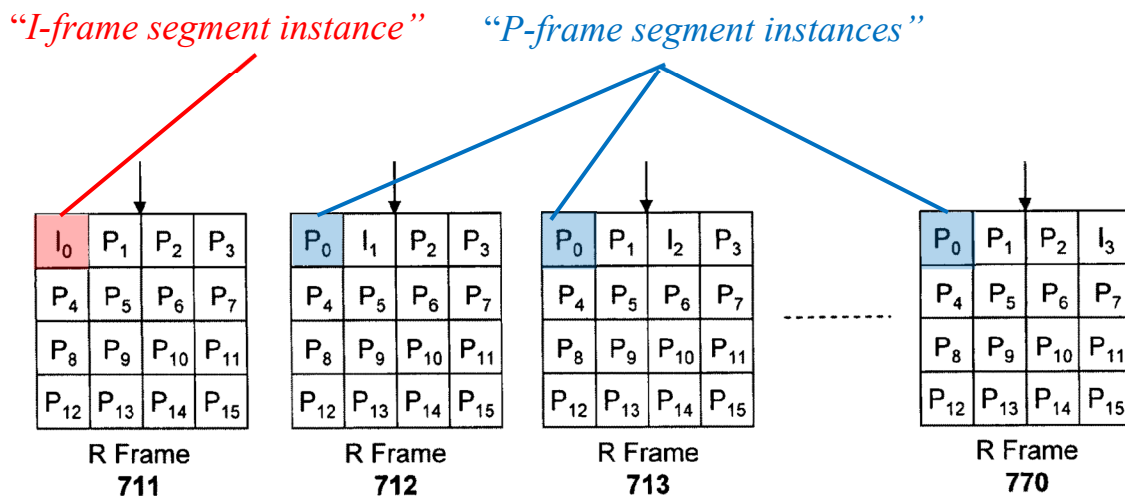
67. First, as explained above at [1.4], Perlman teaches separately encoding each tile in parallel to form “*parallel encoded data streams*” as claimed.

68. Second, Perlman explains that each of the encoded data streams includes both I-frames and P-frames.

The low-latency compression logic 404 employed in one embodiment uses several different techniques to address the range of problems with streaming low-latency compressed video, while maintaining high quality. First, the low-latency compression logic 404 generates only I frames and P frames, thereby alleviating the need to wait several frame times to decode each B frame. In addition, as illustrated in FIG. 7 a, in one embodiment, **the low-latency compression logic 404 subdivides each uncompressed frame 701-760 into a series of “tiles” and individually encodes each tile as either an I frame or a P frame.** The group of compressed I frames and P frames are referred to herein as “R frames” 711-770. In the specific example shown in FIG. 7 a, each uncompressed frame is subdivided into a 4×4 matrix of 16 tiles. However, these underlying principles are not limited to any particular subdivision scheme.

Ex.1006, [0168].

69. As shown in the figure below, the tiles at a particular position within the grid in each frame correspond to “*segment instances*” for a particular encoded data stream. For example, the tiles at the top left are all “*segment instances*” for the encoded data stream corresponding to the top left tile. Some of these “*segment instances*” are “*I-frame segment instances*” and some are “*P-frame segment instances*.”



Ex.1006, Fig. 7a (partial, annotated).

70. Perlman explains that the terms I-frame and P-frame have known meaning in the art. In video codec terminology, I-frames are reference frames that include all the information that is used to produce a frame. P-frames are much smaller in size as they represent only the changes from one frame to the next.

Briefly, an I frame 611 is a DCT-based compression of a complete uncompressed frame 501 (similar to a compressed JPEG image as described above). P frames 612-613 generally are significantly

smaller in size than I frames 611 because they take advantage of the data in the previous I frame or P frame; that is, they contain data indicating the changes between the previous I frame or P frame. B frames 670 are similar to that of P frames except that B frames use the frame in the following reference frame as well as potentially the frame in the preceding reference frame.

Ex.1006, [0157]; *see also* Ex.1006, [0062].

71. Accordingly, a POSITA would have understood that Perlman uses the terms I-frame and P-frame consistent with their generally accepted meaning within the field of video encoding. Thus, when Perlman uses the term I-frame, it refers to a frame that “*that carr[ies] complete information about a particular frame segment.*” When Perlman uses the term P-frame, it refers to a frame that “*carr[ies] partial information about the particular frame segment, the partial information representing information that has changed relative to a previous frame segment instance in that encoded data stream.*”

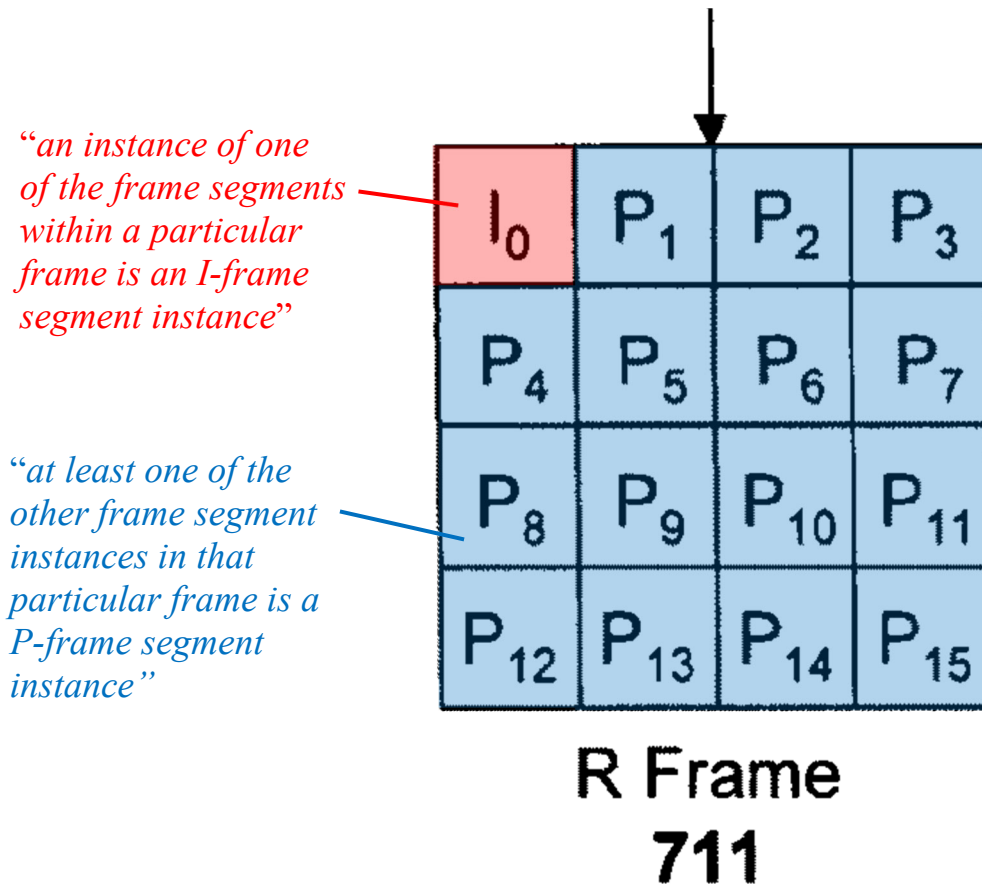
72. Thus, because Perlman’s technique uses I-frames and P-frames for each of the streams associated with each of the tiles, and a POSITA would have understood the meaning of I-frames and P-frames, Perlman renders obvious “*each encoded data stream containing I-frame segment instances that carry complete information about a particular frame segment and P-frame segment instances that carry partial information about the particular frame segment, the partial*

information representing information that has changed relative to a previous frame segment instance in that encoded data stream” as claimed.

[1.6] wherein the encoding of the frame segments for each frame is performed such that when an instance of one of the frame segments within a particular frame is an I-frame segment instance, at least one of the other frame segment instances in that particular frame is a P-frame segment instance,

73. Perlman provides an example in which, for a particular frame (711), one of the tiles (top left) is an I-frame instance (“*when an instance of one of the frame segments within a particular frame is an I-frame segment instance*”), and all other tiles are P-frame instances (“*at least one of the other frame segment instances in that particular frame is a P-frame segment instance*”). This is shown in Fig. 7a below.²

² While I provide R Frame 711 as an example, the other illustrated frames (712-713) similarly show only one I-frame tile while the rest of the tiles are P-frames.



Ex.1006, Fig. 7a (partial, annotated).

In one embodiment, the low-latency compression logic 404 divides up the video frame into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame (i.e., the tile is compressed as if it is a separate video frame of 1/16th the size of the full image, and the compression used for this “mini” frame is I frame compression) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively. With each successive video frame, the tile to be encoded as an I tile is changed. Thus, in a given frame time, only one tile of the tiles in the video frame is an I

tile, and the remainder of the tiles are P tiles. For example, in FIG. 7 a, tile 0 of uncompressed frame 701 is encoded as I tile I_0 and the remaining 1-15 tiles are encoded as P tiles P_1 through P_{15} to produce R frame 711. In the next uncompressed video frame 702, tile 1 of uncompressed frame 701 is encoded as I tile I_1 and the remaining tiles 0 and 2 through 15 are encoded as P tiles, P_0 and P_2 through P_{15} , to produce R frame 712. **Thus, the I tiles and P tiles for tiles are progressively interleaved in time over successive frames. The process continues until an R tile 770 is generated with the last tile in the matrix encoded as an I tile (i.e., I_{15}).** The process then starts over, generating another R frame such as frame 711 (i.e., encoding an I tile for tile 0) etc. Although not illustrated in FIG. 7 a, in one embodiment, the first R frame of the video sequence of R frames contains only I tiles (i.e., so that subsequent P frames have reference image data from which to calculate motion). Alternatively, in one embodiment, the startup sequence uses the same I tile pattern as normal, but does not include P tiles for those tiles that have not yet been encoded with an I tile. In other words, certain tiles are not encoded with any data until the first I tile arrives, thereby avoiding startup peaks in the video stream data rate 934 in FIG. 9 a, which is explained in further detail below. Moreover, as described below, various different sizes and shapes may be used for the tiles while still complying with these underlying principles.

Ex.1006, [0169].

74. Thus, because Perlman provides an example where one tile is an I-frame and all other tiles are P-frames, Perlman renders obvious “*wherein the*

encoding of the frame segments for each frame is performed such that when an instance of one of the frame segments within a particular frame is an I-frame segment instance, at least one of the other frame segment instances in that particular frame is a P-frame segment instance” as claimed.

[1.7] whereby in the respective parallel encoded data streams in any period of time when an I-frame segment instance is present in one of the parallel encoded data streams, a P-frame segment instance is present in at least one of the other parallel encoded data streams;

75. Perlman explains that the I-frame for each of the encoded data streams is transmitted in a different time frame.

In one embodiment, the low-latency compression logic 404 divides up the video frame into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame (i.e., the tile is compressed as if it is a separate video frame of 1/16th the size of the full image, and the compression used for this “mini” frame is I frame compression) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively. With each successive video frame, the tile to be encoded as an I tile is changed. **Thus, in a given frame time, only one tile of the tiles in the video frame is an I tile, and the remainder of the tiles are P tiles.** For example, in FIG. 7 a, tile 0 of uncompressed frame 701 is encoded as I tile I₀ and the remaining 1-15 tiles are encoded as P tiles P₁ through P₁₅ to produce R frame 711. In the next uncompressed video frame 702, tile 1 of uncompressed frame 701 is encoded as I tile I₁ and the

remaining tiles 0 and 2 through 15 are encoded as P tiles, P_0 and P_2 through P_{15} , to produce R frame 712. **Thus, the I tiles and P tiles for tiles are progressively interleaved in time over successive frames.**

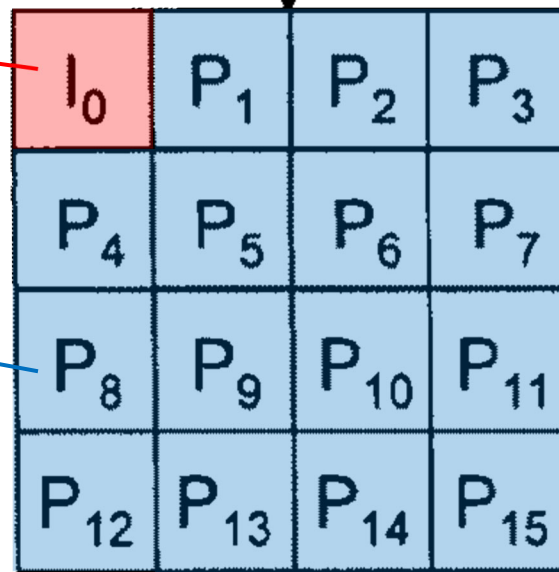
The process continues until an R tile 770 is generated with the last tile in the matrix encoded as an I tile (i.e., I_{15}). The process then starts over, generating another R frame such as frame 711 (i.e., encoding an I tile for tile 0) etc. Although not illustrated in FIG. 7 a, in one embodiment, the first R frame of the video sequence of R frames contains only I tiles (i.e., so that subsequent P frames have reference image data from which to calculate motion). Alternatively, in one embodiment, the startup sequence uses the same I tile pattern as normal, but does not include P tiles for those tiles that have not yet been encoded with an I tile. In other words, certain tiles are not encoded with any data until the first I tile arrives, thereby avoiding startup peaks in the video stream data rate 934 in FIG. 9 a, which is explained in further detail below. Moreover, as described below, various different sizes and shapes may be used for the tiles while still complying with these underlying principles

Ex.1006, [0169].

76. The annotated figure below illustrates a given frame time (*“period of time”*). Each of the different tiles corresponds to a different parallel encoded data stream. For a given frame time, there is one I-frame tile (*“when an I-frame segment instance is present in one of the parallel encoded data streams”*) and all other tiles are P-frames (*“a P-frame segment instance is present in at least one of the other parallel encoded data streams”*).

“in the respective parallel encoded data streams in any period of time when an I-frame segment instance is present in one of the parallel encoded data streams”

“a P-frame segment instance is present in at least one of the other parallel encoded data streams”



R Frame

711

Ex.1006, Fig. 7a (partial, annotated).

77. Fig. 12 also illustrates that in a particular “*period of time*” there is only one I-frame and all other tiles are P-frames.

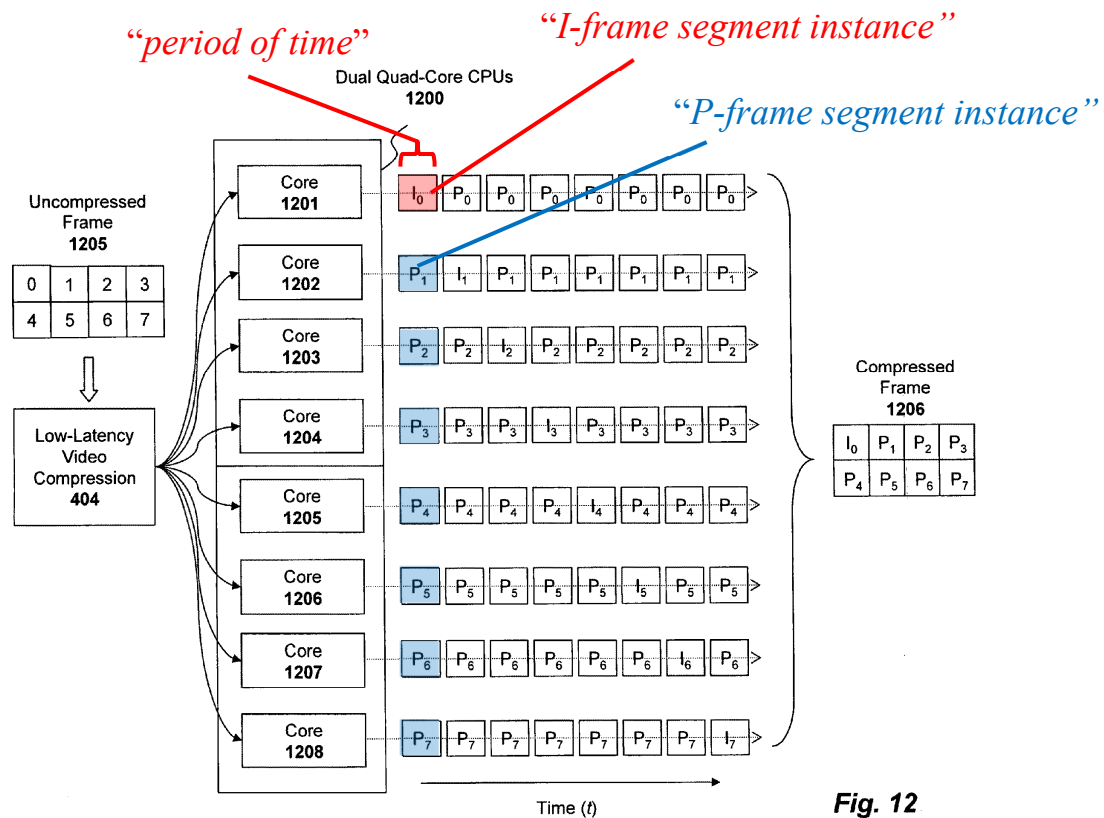


Fig. 12

Ex.1006, Fig. 12 (annotated).

78. Thus, because Perlman describes distributing the I-frames in time such that for a given frame time, one tile is an I-frame and all other tiles are P-frames, Perlman renders obvious *“whereby in the respective parallel encoded data streams in any period of time when an I-frame segment instance is present in one of the parallel encoded data streams, a P-frame segment instance is present in at least one of the other parallel encoded data streams”* as claimed.

[1.8] transmitting the parallel encoded data streams independently over a network to a receiver to permit the receiver to decode each frame segment and recombine the decoded frame segments from the different encoded data streams to reproduce an image;

79. First, Perlman explains that the compressed tiles (*“parallel encoded data streams”*) are transmitted to a destination client over the Internet (*“over a network”*).

Each frame time, the resulting compressed tiles are combined into a packet stream, using the techniques previously described, **and then the compressed tiles are transmitted to a destination client 415.**

Ex.1006, [0214].

The graphical and audio output of the executed game or application software—i.e., new sequences of video images—are provided to the low-latency video compression logic 404 which compresses the sequences of video images according to low-latency video compression techniques, such as those described herein and transmits a compressed video stream, typically with compressed or uncompressed audio, back to the client 415 over the Internet 410

(or, as described below, over an optimized high speed network service that bypasses the general Internet). Low-latency video decompression logic 412 on the client 415 then decompresses the video and audio streams and renders the decompressed video stream, and typically plays the decompressed audio stream, on a display device 422. Alternatively, the audio can be played on speakers separate from the display device 422 or not at all. Note that, despite the fact that input device 421 and display device 422 are shown as free-standing devices in FIGS. 2 a and 2 b, they may be integrated within client devices such as portable computers or mobile devices

Ex.1006, [0117].

80. Second, while Perlman describes combining the compressed tiles into a packet stream, the compressed tiles maintain their independence through transmission. Perlman's streams are "*independently*" transmitted because they have to be recombined on the receiving end. Perlman decompresses each tile separately and then combines the tiles for rendering the full image. Indeed, similar to the parallel encoding process, Perlman describes a parallel **decoding** process as well.

The video decompression logic 412 running on the client 415 decompresses each tile as if it is a separate video sequence of small I and P frames, and then renders each tile to the frame buffer driving display device 422. For example, I₀ and P₀ from R frames 711 to 770 are used to decompress and render tile 0 of the video image. Similarly, I₁ and P₁ from R frames 711 to 770 are used to reconstruct tile 1, and so on. **As mentioned above, decompression of I frames and P frames is well known in the art, and decompression of I tiles and P tiles can be accomplished by having a multiple instances of a video decompressor running in the client 415.**

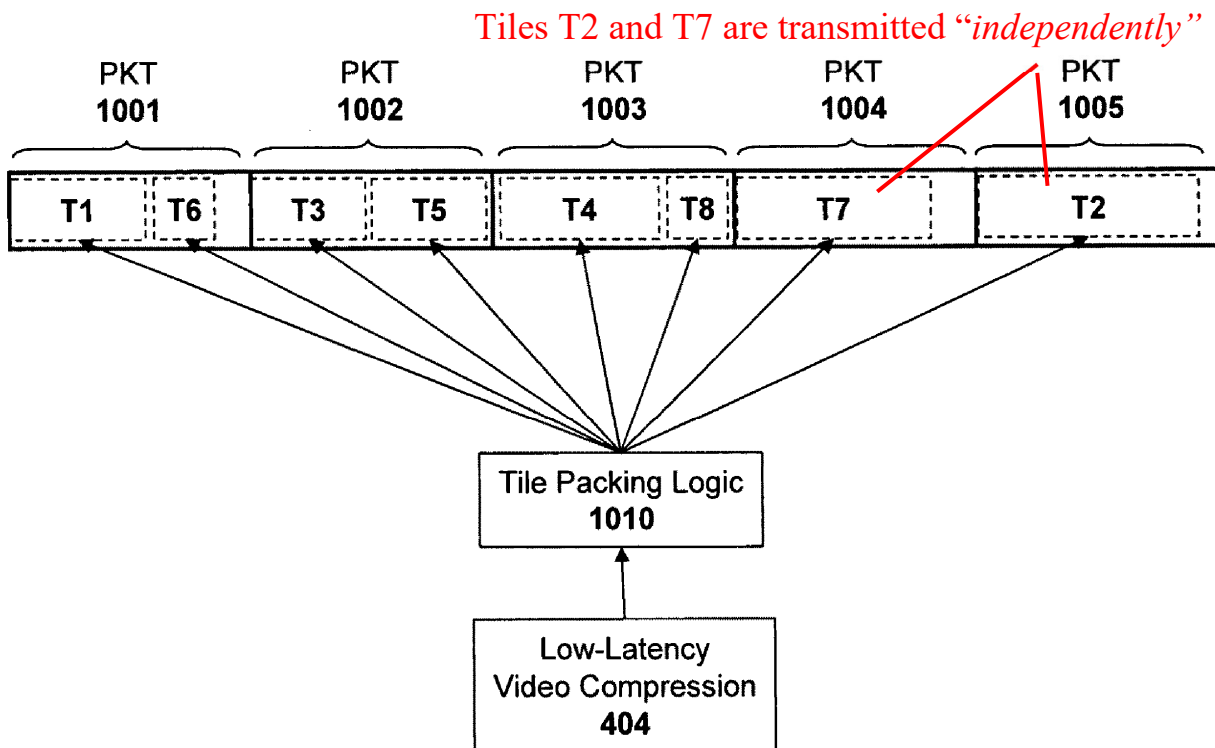
Although multiplying processes would seem to increase the computational burden on client 415, it actually does not because the tile themselves are proportionally smaller relative to the number of additional processes, so the number of pixels displayed is the same as if there were one process and using conventional full sized I and P frames.

Ex.1006, [0170].

81. Accordingly, because Perlman's technique decompresses each tile

separately and reconstructs the video on the receiving end, Perlman's compressed tiles maintain their independence through transmission and are thus transmitted "*independently over a network to a receiver.*"

82. Alternatively, Perlman's tiles may be transmitted in separate packets and are thus transmitted "*independently.*" Perlman describes tile packing logic in which tile T7 is in packet 1004 and tile T2 is added to packet 1005. *See* Ex.1006, [0198]. This is shown below in Fig. 10b.



Ex.1006, Fig. 10b (annotated).

83. Accordingly, because each tile is part of a different data stream, and at least some of the tiles are transmitted in separate packets, at least some of the

encoded streams are transmitted “*independently*” from each other. Moreover, it would have been obvious that in some instances, each of the different tiles from the different streams would be transmitted in different packets. Thus, because a POSITA would have found it obvious for each tile to be transmitted in a separate packet, Perlman renders obvious “*transmitting the parallel encoded data streams independently over a network ...*” as claimed.

84. **Third**, Perlman’s technique transmits the encoded data streams independently for decompression and recombination on the receiving end.

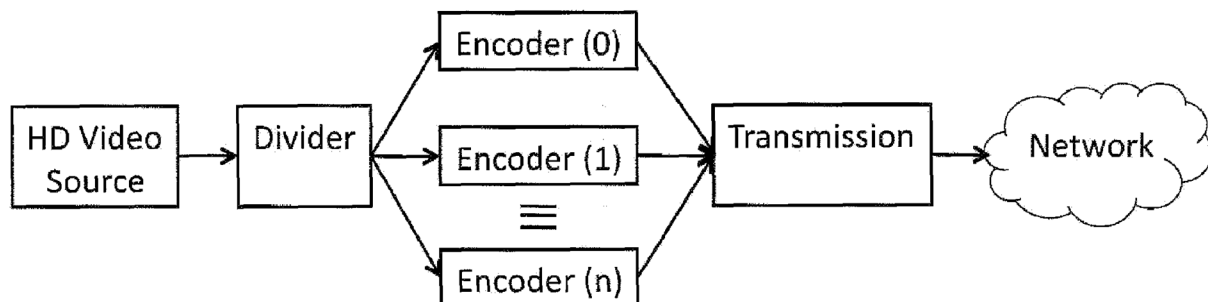
The video decompression logic 412 running on the client 415 decompresses each tile as if it is a separate video sequence of small I and P frames, and then renders each tile to the frame buffer driving display device 422. For example, I_0 and P_0 from R frames 711 to 770 are used to decompress and render tile 0 of the video image. Similarly, I_1 and P_1 from R frames 711 to 770 are used to reconstruct tile 1, and so on. **As mentioned above, decompression of I frames and P frames is well known in the art, and decompression of I tiles and P tiles can be accomplished by having a multiple instances of a video decompressor running in the client 415.**

Although multiplying processes would seem to increase the computational burden on client 415, it actually does not because the tile themselves are proportionally smaller relative to the number of additional processes, so the number of pixels displayed is the same as if there were one process and using conventional full sized I and P frames.

Ex.1006, [0170].

85. Accordingly, Perlman's technique *"permit[s] the receiver to decode each frame segment and recombine the decoded frame segments from the different encoded data streams to reproduce an image."*

86. Perlman describes combining the independently encoded streams for transmission similar to how transmission is described in the '426 patent. In the '426 patent, the independently encoded streams are illustrated as being combined for transmission as shown in Fig. 3 below.



Ex.1001, Fig. 3.

87. Thus, because Perlman describes transmitting separately encoded data streams such that the receiver can reassemble the streams to form the video image, Perlman renders obvious *"transmitting the parallel encoded data streams independently over a network to a receiver to permit the receiver to decode each frame segment and recombine the decoded frame segments from the different encoded data streams to reproduce an image"* as claimed.

[1.9] wherein during transmission of I-frame segment instances of one of the

frame segments, P-frame segment instances of all other frame segments are transmitted.

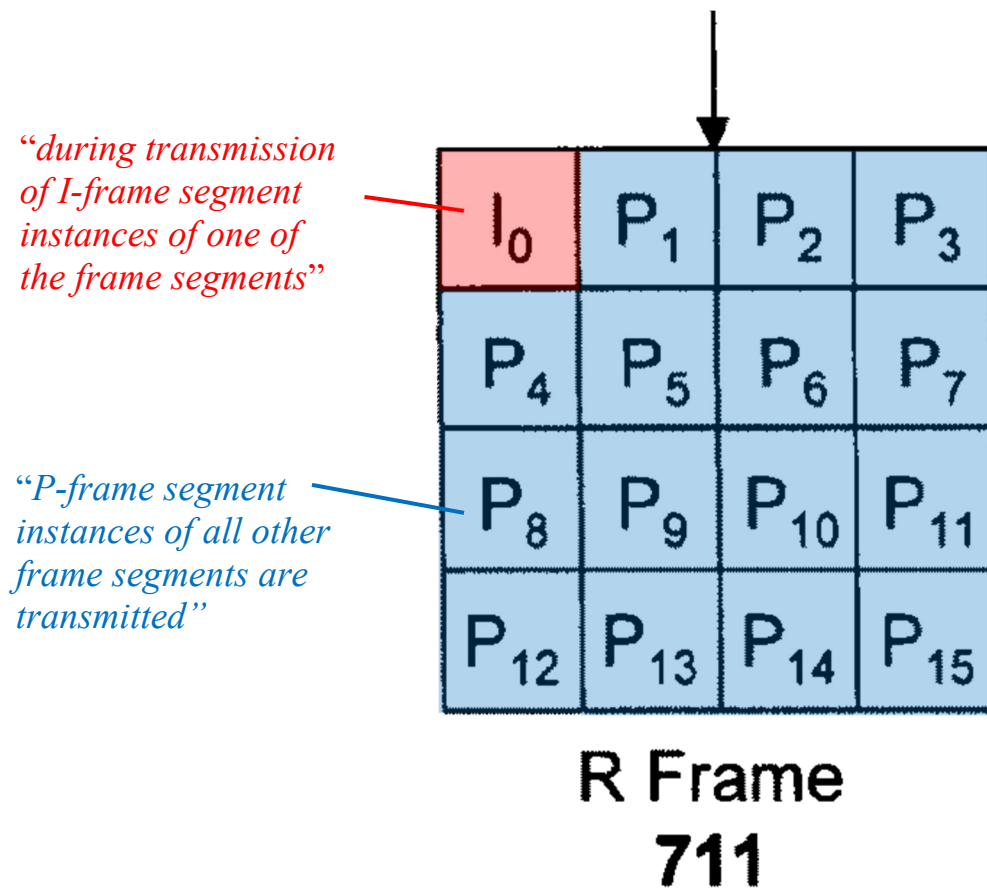
88. Perlman provides an example in which, when one of the frames is an I-frame, all other frames are P-frames.

In one embodiment, the low-latency compression logic 404 divides up the video frame into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame (i.e., the tile is compressed as if it is a separate video frame of 1/16th the size of the full image, and the compression used for this “mini” frame is I frame compression) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively. With each successive video frame, the tile to be encoded as an I tile is changed. **Thus, in a given frame time, only one tile of the tiles in the video frame is an I tile, and the remainder of the tiles are P tiles.**

For example, in FIG. 7 a, tile 0 of uncompressed frame 701 is encoded as I tile I_0 and the remaining 1-15 tiles are encoded as P tiles P_1 through P_{15} to produce R frame 711. In the next uncompressed video frame 702, tile 1 of uncompressed frame 701 is encoded as I tile I_1 and the remaining tiles 0 and 2 through 15 are encoded as P tiles, P_0 and P_2 through P_{15} , to produce R frame 712. Thus, the I tiles and P tiles for tiles are progressively interleaved in time over successive frames. The process continues until an R tile 770 is generated with the last tile in the matrix encoded as an I tile (i.e., I_{15}). The process then starts over, generating another R frame such as frame 711 (i.e., encoding an I tile for tile 0) etc. Although not illustrated in FIG. 7 a, in one embodiment,

the first R frame of the video sequence of R frames contains only I tiles (i.e., so that subsequent P frames have reference image data from which to calculate motion). Alternatively, in one embodiment, the startup sequence uses the same I tile pattern as normal, but does not include P tiles for those tiles that have not yet been encoded with an I tile. In other words, certain tiles are not encoded with any data until the first I tile arrives, thereby avoiding startup peaks in the video stream data rate 934 in FIG. 9 a, which is explained in further detail below. Moreover, as described below, various different sizes and shapes may be used for the tiles while still complying with these underlying principles.

Ex.1006, [0169].



Ex.1006, Fig. 7a (partial, annotated).

89. Fig. 12 also illustrates that during transmission of an I-frame tile, all other tiles are P-frames.

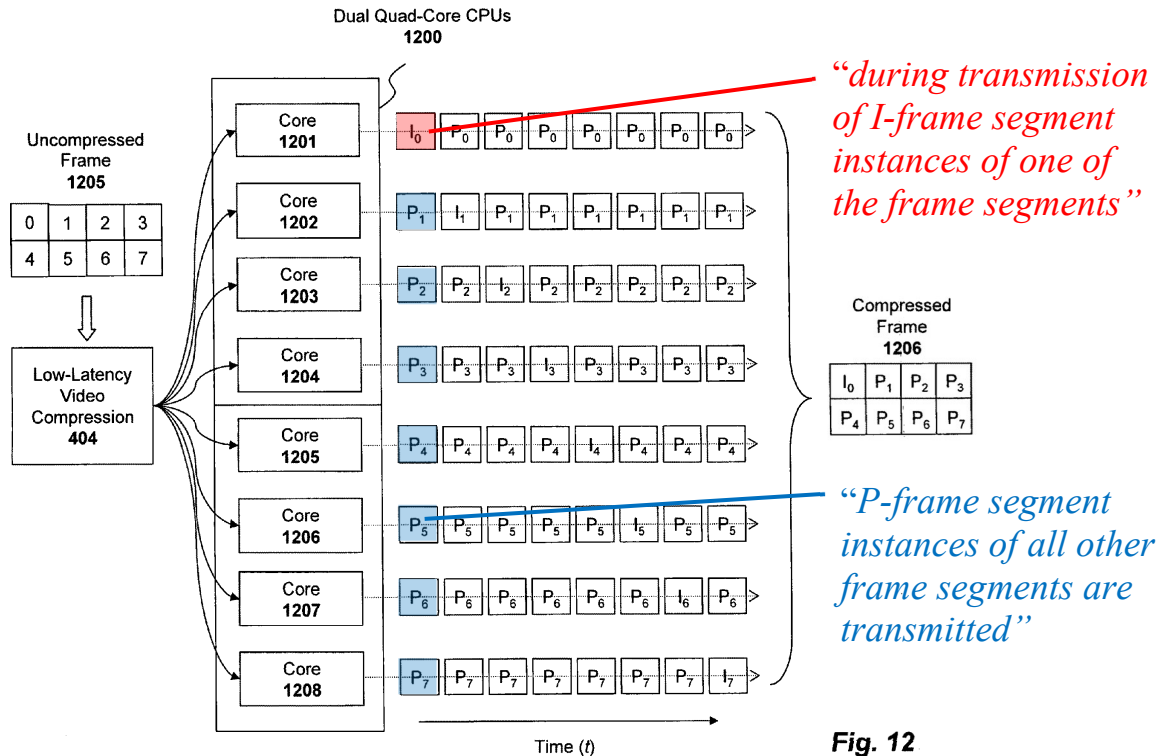


Fig. 12

Ex.1006, Fig. 12 (annotated).

90. Thus, because Perlman provides an example where one tile is an I-frame and all other tiles are P-frames, Perlman renders obvious “wherein during transmission of I-frame segment instances of one of the frame segments, P-frame segment instances of all other frame segments are transmitted” as claimed.

3. Claim 2

[2.1] A method according to claim 1, wherein the frame segments correspond to predetermined parts of the video frames.

91. Perlman illustrates an example in which the compression logic divides the frame into a four-by-four grid of 16 tiles as shown in Fig. 7a.

The low-latency compression logic 404 employed in one embodiment uses several different techniques to address the range of problems with streaming low-latency compressed video, while maintaining high quality. First, the low-latency compression logic 404 generates only I frames and P frames, thereby alleviating the need to wait several frame times to decode each B frame. In addition, as illustrated in FIG. 7 a, in one embodiment, the low-latency compression logic 404 subdivides each uncompressed frame 701-760 into a series of “tiles” and individually encodes each tile as either an I frame or a P frame. The group of compressed I frames and P frames are referred to herein as “R frames” 711-770. **In the specific example shown in FIG. 7 a, each uncompressed frame is subdivided into a 4×4 matrix of 16 tiles.** However, these underlying principles are not limited to any particular subdivision scheme.

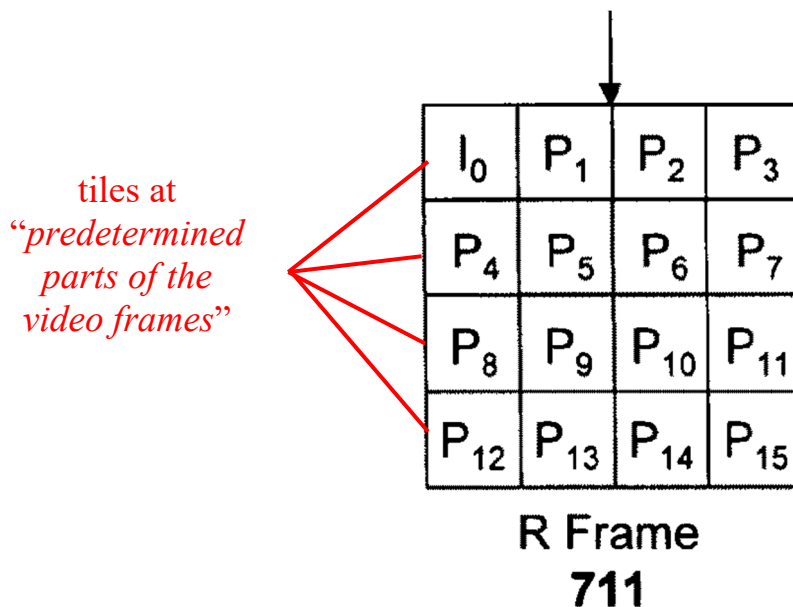
Ex.1006, [0168].

92. The four-by-four grid is a predetermined pattern of equally-sized tiles, each sized at 1/16th of the total frame.

In one embodiment, the low-latency compression logic 404 divides up the video frame into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame (i.e., the tile is compressed as if it is a separate video frame **of 1/16th the size of the full image**, and the compression used for this “mini” frame is I frame compression) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively.

With each successive video frame, the tile to be encoded as an I tile is changed. Thus, in a given frame time, only one tile of the tiles in the video frame is an I tile, and the remainder of the tiles are P tiles. For example, in FIG. 7 a, tile 0 of uncompressed frame 701 is encoded as I tile I_0 and the remaining 1-15 tiles are encoded as P tiles P_1 through P_{15} to produce R frame 711.

Ex.1006, [0169].



Ex.1006, Fig. 7a (partial, annotated).

93. Accordingly, by using the compression logic to divide the frame into a four-by-four tile grid, the tiles “correspond to predetermined parts of the video frames” as claimed.

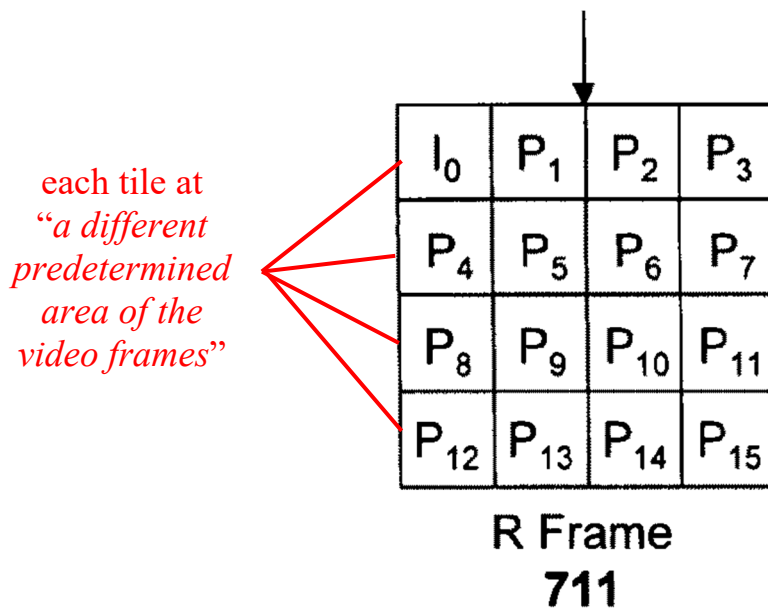
94. Thus, because Perlman describes dividing the tiles in a predetermined grid pattern, Perlman renders obvious “wherein the frame segments correspond to predetermined parts of the video frames” as claimed.

4. Claim 3

[3.1] *A method according to claim 1, wherein each frame segment corresponds to a different predetermined area of the video frames.*

95. First, as discussed above at [2.1], Perlman describes a method to divide the frame into 16 equally-sized tiles.

96. Second, in the example shown in Fig. 1 there are no overlapping tiles and thus each tile “corresponds to a different predetermined area of the video frames.”



Ex.1006, Fig. 7a (partial, annotated).

97. Thus, because Perlman describes dividing the frame so as to produce non-overlapping tiles, Perlman renders obvious “wherein each frame segment corresponds to a different predetermined area of the video frames” as claimed.

5. Claim 4

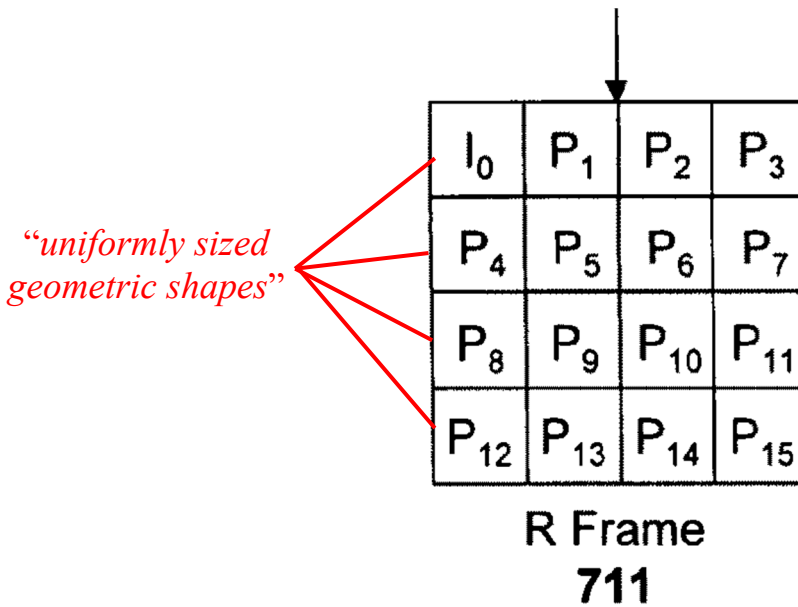
[4.1] A method according to claim 1, wherein the frame segments are uniformly sized geometric shapes.

98. Perlman both illustrates and describes an example in which the tiles are “*uniformly sized*” at 1/16th of the total image.

In one embodiment, the low-latency compression logic 404 divides up the video frame into a number of tiles, and encodes (i.e., compresses) one tile from each frame as an I frame (**i.e., the tile is compressed as if it is a separate video frame of 1/16th the size of the full image, and the compression used for this “mini” frame is I frame compression**) and the remaining tiles as P frames (i.e., the compression used for each “mini” 1/16th frame is P frame compression). Tiles compressed as I frames and as P frames shall be referred to as “I tiles” and “P tiles”, respectively. With each successive video frame, the tile to be encoded as an I tile is changed. Thus, in a given frame time, only one tile of the tiles in the video frame is an I tile, and the remainder of the tiles are P tiles. For example, in FIG. 7 a, tile 0 of uncompressed frame 701 is encoded as I tile I_0 and the remaining 1-15 tiles are encoded as P tiles P_1 through P_{15} to produce R frame 711.

Ex.1006, [0169].

99. As illustrated in Fig. 7a below, the tiles are divided into equal-sized squares (“*uniformly sized geometric shapes*”).



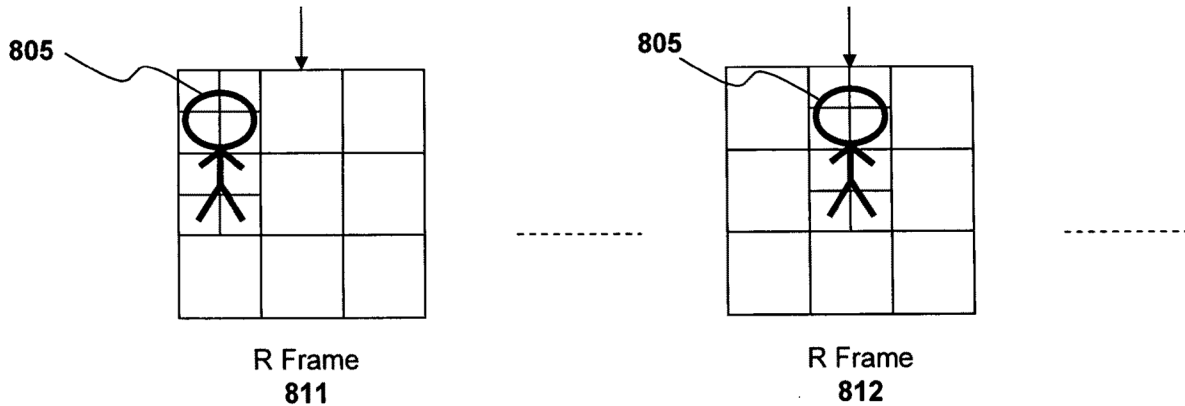
Ex.1006, Fig. 7a (partial, annotated).

100. Thus, because Perlman describes an example in which the tiles are equal-sized squares, Perlman renders obvious “*wherein the frame segments are uniformly sized geometric shapes*” as claimed.

6. Claim 5

[5.1] *A method according to claim 1, wherein at least one of frame segment size and shape is different for one of the frame segments compared to another of the frame segments.*

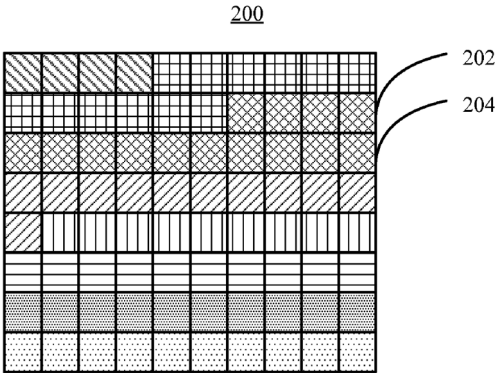
101. While Perlman provides an example in which the tiles are equal-sized, Perlman discloses that the tiles may have various sizes and shapes.

**Ex.1006, Fig. 8 (partial).**

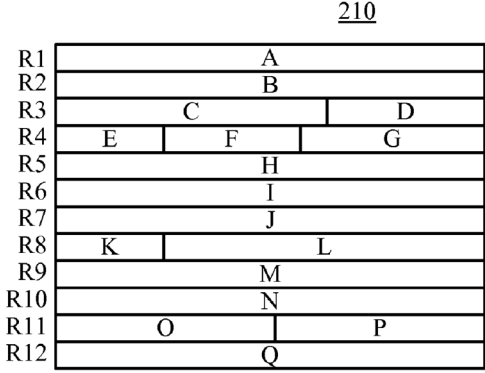
R frames can be configured in a variety of different ways, depending upon the nature of the video sequence, the reliability of the channel, and the available data rate. In an alternative embodiment, a different number of tiles is used than 16 in a 4×4 configuration. For example 2 tiles may be used in a 2×1 or 1×2 configuration, 4 tiles may be used in a 2×2, 4×1 or 1×4 configuration, 6 tiles may be used in a 3×2, 2×3, 6×1 or 1×6 configurations or 8 tiles may be used in a 4×2 (as shown in FIG. 7 b), 2×4, 8×1 or 1×8 configuration. **Note that the tiles need not be square, nor must the video frame be square, or even rectangular. The tiles can be broken up into whatever shape best suits the video stream and the application used.**

Ex.1006, [0174].

102. Indeed, it was known in the art to use variously sized and shaped slices. *See* Ex.1008, Fig. 3A, 3B.



Ex.1008, Fig. 3A.



Ex.1008, Fig. 3B.

FIG. 3A illustrates a first exemplary sample slice partitioning of a frame 200 according to certain configurations using a slice structure in accordance with the MPEG-1 standard. The different slice partitions are denoted by different cross hatching. In this example, some of the macroblocks of the slices occupy two horizontal rows that are adjacent. In this frame 200, a slice structure includes macroblocks 202 on a first horizontal row and macroblocks 204 on a second horizontal row. In these structures not all of the macroblocks in a slice group needs to be immediately adjacent.

FIG. 3B illustrates a second exemplary sample slice partitioning of a frame 210 according to certain configurations using a slice structure in accordance with the MPEG-2 standard. In FIG. 3B, the slice structures are individually denoted by A-Q. These slice structures are horizontally arranged from row to row without order. For example, the slice structure A extends the entire first horizontal row R1. Likewise, the slice structure B extends the entire second horizontal row R2. However, in this example, in the third horizontal row R3, the row R3 is shared by the slice structures C and D. The arrangement of slice structure on the horizontal rows R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, and

R12 are intended to be exemplary. Nonetheless, each slice can occupy at most one horizontal row only and the right boundary of the frame always marks the end of a slice.

Ex.1008, [0062]-[0063]

103. Thus, because Perlman provides examples of using various shapes and sizes as known in the art, Perlman renders obvious “*wherein at least one of frame segment size and shape is different for one of the frame segments compared to another of the frame segments*” as claimed.

7. Claim 7

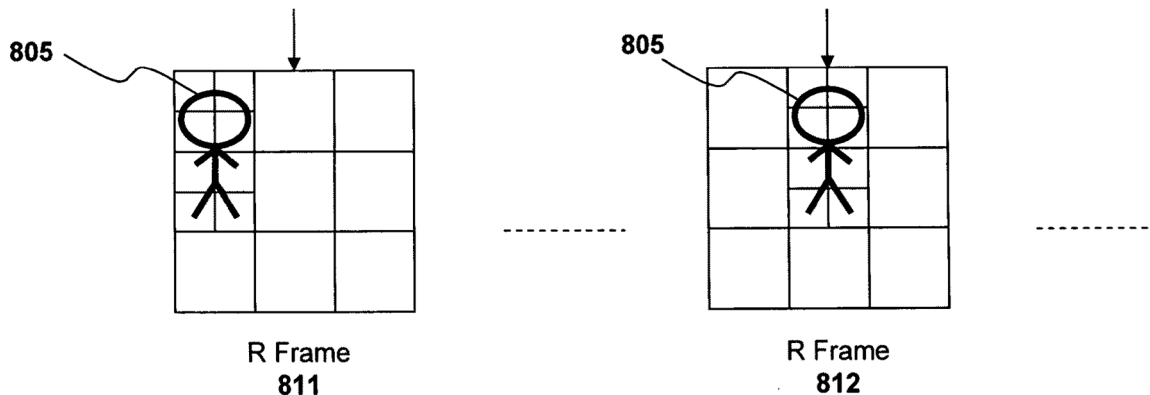
[7.1] *A method according to claim 1, wherein the frame segments are determined in dependence upon external application data and indicative of the amount of motion that is expected in different areas of the video frames.*

104. First, Perlman discloses that an uncompressed frame may be divided into tiles based on where there is expected to be more scene complexity (“*amount of motion that is expected in different areas of the video frames*”).

In one embodiment, to improve resolution of certain regions of the video stream, the video compression logic 404 uses smaller tiles to encode areas of the video stream with relatively more scene complexity and/or motion than areas of the video stream with relatively less scene complexity and/or motion. For example, as illustrated in FIG. 8, smaller tiles are employed around a moving character 805 in one area of one R frame 811 (potentially followed by a series of R frames with the same tile sizes (not shown)). Then, when the character 805 moves to a new area of the image, smaller tiles are

used around this new area within another R frame 812, as illustrated. As mentioned above, various different sizes and shapes may be employed as “tiles” while still complying with these underlying principles.

Perlman, [0178].



Ex.1006, Fig. 8 (partial).

105. Perlman also gives an example in which the frames are encoded based on where there is expected to be more or less motion using pre-configured settings.

In another embodiment, the low-latency video compression logic 404 in FIGS. 7 a and 7 b controls the allocation of bits to the various tiles in the R frames either by **being pre-configured by settings, based on known characteristics of the video sequence to be compressed**, or automatically, based upon an ongoing analysis of the image quality in each tile. **For example, in some racing video games, the front of the player's car (which is relatively motionless in the scene) takes up a large part of the lower half of the screen, whereas the upper half of the screen is entirely filled with the oncoming roadway, buildings and scenery, which is almost always in motion.** If the compression

logic 404 allocates an equal number of bits to each tile, then the tiles on the bottom half of the screen (tiles 4-7) in uncompressed frame 721 in FIG. 7 b, will generally be compressed with higher quality than tiles than the tiles in the upper half of the screen (tiles 0-3) in uncompressed frame 721 in FIG. 7 b. If this particular game, or this particular scene of the game is known to have such characteristics, then the operators of the hosting service 210 can configure the compression logic 404 to allocate more bits to the tiles in the top of the screen than to tiles at the bottom of the screen. Or, the compression logic 404 can evaluate the quality of the compression of the tiles after frames are compressed (using one or more of many compression quality metrics, such as Peak Signal-To-Noise Ratio (PSNR)) and if it determines that over a certain window of time, certain tiles are consistently producing better quality results, then it gradually allocates more bits to tiles that are producing lower quality results, until the various tiles reach a similar level of quality. In an alternative embodiment, the compressor logic 404 allocates bits to achieve higher quality in a particular tile or group of tiles. For example, it may provide a better overall perceptual appearance to have higher quality in the center of the screen than at the edges.

Perlman, [0177].

106. A POSITA would have found it obvious that in the context of video conferencing (which Perlman describes as discussed at [1.0]), there would be more motion around a user's face and less motion in the user's background.

107. Second, Perlman describes dividing the frame into tiles and encoding

the tiles (*“the frame segments are determined”*) based on pre-configured settings (*“external application data”*). Data from a game, teleconference, or movie application that indicates where a moving character is currently at within the frame is *“application data”* that is *“external”* to Perlman’s compression logic. This data may include for example, “pre-configured by settings, based on known characteristics of the video sequence to be compressed.” Ex.1006, [0177].

108. Perlman describes dividing frames based on expected motion similar to how the ’426 patent describes dividing frames based on external application data:

In a simple application the segments are defined as geometric shapes such as rectangles or squares. Optionally, **the segments are defined based upon knowledge that is provided for**, or determined on the basis of **usage of a particular videoconferencing or telepresence system**. For instance, if it is expected that a single participant will be seated centrally within the video frame then an acceptable strategy for dividing the frame may be the one that is illustrated in FIG. 5d. wherein the central rectangle is located over the portion of the frame that contains the participants face. Thus, the horizontal and vertical segments define portions of the frame that are expected to contain less motion. The central rectangle can further be assigned to an encoder with a high bitrate, whilst the remaining segments are allotted lower bitrates.

Ex.1001, 7:30-44.

109. Thus, because Perlman describes dividing and encoding frame tiles

based on pre-configured settings of known characteristics (or character movement) that indicates where expected motion will take place, Perlman renders obvious “*wherein the frame segments are determined in dependence upon external application data and indicative of the amount of motion that is expected in different areas of the video frames*” as claimed.

8. Claim 8

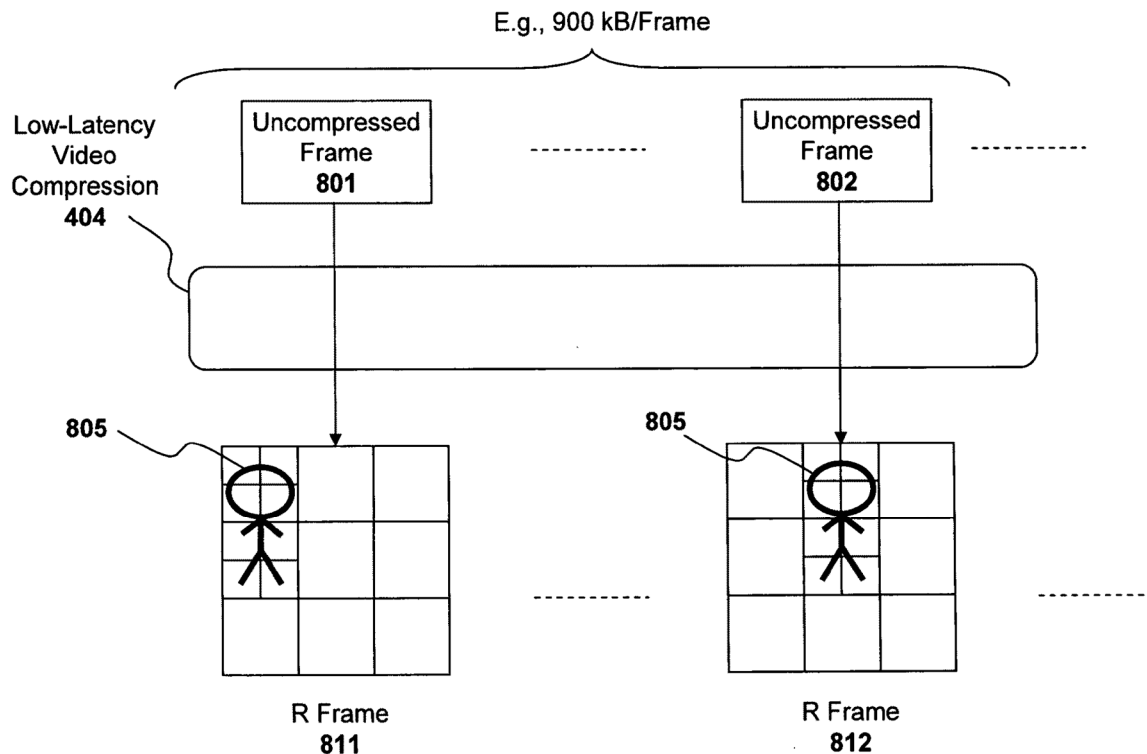
[8.1] A method according to claim 1, wherein the frame segments are determined in dependence upon external prior use data indicative of the amount of motion previously determined in different areas of the video frames.

110. First, as described above at [7.1], a POSITA would have found it obvious to divide a frame into multiple tiles based on where there is expected to be more scene complexity and motion (“*data indicative of the amount of motion*”). Perlman describes a technique in which a frame is divided up based on the motion within the video sequence.

In one embodiment, to improve resolution of certain regions of the video stream, the video compression logic 404 uses smaller tiles to encode areas of the video stream with relatively more scene complexity and/or motion than areas of the video stream with relatively less scene complexity and/or motion. **For example, as illustrated in FIG. 8, smaller tiles are employed around a moving character 805 in one area of one R frame 811 (potentially followed by a series of R frames with the same tile sizes (not shown)). Then, when the character 805 moves to a new area of the image, smaller tiles are**

used around this new area within another R frame 812, as illustrated. As mentioned above, various different sizes and shapes may be employed as “tiles” while still complying with these underlying principles.

Ex.1006, [0178].



Ex.1006, Fig. 8.

111. Accordingly, by tracking motion within the frame and using that tracking to divide the frame into tiles, Perlman teaches dividing frames based on prior use data. A POSITA would have found it obvious that to sense motion within the frame, an application would determine where the object has been (prior use) to compare with where the object is now. In other words, changes in motion involve both past and present frames and thus include “*external prior use data indicative of*

the amount of motion previously determined in different areas of the video frames.”

Again, data from a game, teleconference, or movie application that indicates where a moving character is currently at within the frame is “*application data*” that is “*external*” to Perlman’s compression logic.

112. Additionally, as described above at [7.1], Perlman describes dividing and encoding the tiles (“*frame segments are determined*”) based on pre-configured settings (“*prior use data indicative of the amount of motion previously determined in different areas of the video frames*”). For example, Perlman explains that the encoding of tiles may be based on “pre-configured by settings, based on known characteristics of the video sequence to be compressed.” Ex.1006, [0177]. As an example of known characteristics, Perlman describes a racing game in which “the front of the player’s car (which is relatively motionless in the scene) takes up a large part of the lower half of the screen, whereas the upper half of the screen is entirely filled with the oncoming roadway, buildings and scenery, which is almost always in motion.” Ex.1006, [0177]. The data from the game that indicates where motion is expected (based on known characteristics) is “*external*” to the compression logic. Accordingly, data associated with a game, teleconference, or movie application that indicate where there is more scene complexity or motion is “*application data*” that is “*external*” to Perlman’s compression logic.

113. Thus, because Perlman describes using various techniques to divide a

frame into tiles based on where motion is detected (comparing past and current position of an object), and encoding the tiles based on pre-configured settings, Perlman renders obvious “*wherein the frame segments are determined in dependence upon external prior use data indicative of the amount of motion previously determined in different areas of the video frames*” as claimed.

9. Claim 9

[9.1] A method according to claim 1, wherein a period between regularly spaced I-frame segment instances for one of the frame segments is different than a period between regularly spaced I-frame segment instances for another of the frame segments.

114. Perlman explains that an I-frame occurs for each tile at regular intervals (“*a period between regularly spaced I-frame segment instances*”), for example, once every sixteen frames: “the 16 tiles that make up an I frame cycle through in 16 frames times,” so for each tile “an I frame is introduced every 16 frame times.” Ex.1006, [0172]. Perlman also contemplates other regular intervals, such as “an I tile generated only every other frame.” Ex.1006, [0175].

115. Perlman explains that I-frames may be more frequent for cases where there are more scene changes.

In yet another embodiment, **certain tiles may be sequenced with I tiles more often than other tiles if, for example, certain areas of the screen are known to have more motion requiring from frequent I tiles**, while others are more static (e.g., showing a score for a game) requiring less frequent I tiles. Moreover, although each frame is

illustrated in FIGS. 7 a-b with a single I tile, multiple I tiles may be encoded in a single frame (depending on the bandwidth of the transmission channel). Conversely, certain frames or frame sequences may be transmitted with no I tiles (i.e., only P tiles).

Ex.1006, [0175].

116. Accordingly, if some tiles produce I-frames more frequently due to more motion, then some tiles will have different periods between the I-frames than others.

117. Thus, because Perlman describes a dividing technique in which some tiles have more frequent I-frames than others, Perlman renders obvious “*wherein a period between regularly spaced I-frame segment instances for one of the frame segments is different than a period between regularly spaced I-frame segment instances for another of the frame segments*” as claimed.

10. Claim 10

[10.1] *A method according to claim 1, wherein the frame segments are encoded in encoders having different encoding characteristics.*

118. Perlman teaches that different tiles may be compressed at different bit rates (“*different encoding characteristics*”). Perlman also indicates that the compression logic may use different processors or other hardware components (“*encoders*”). *See supra*, [1.3].

In another embodiment, the low-latency video compression logic 404 in FIGS. 7 a and 7 b controls the allocation of bits to the various tiles in

the R frames either by being pre-configured by settings, based on known characteristics of the video sequence to be compressed, or automatically, based upon an ongoing analysis of the image quality in each tile. For example, in some racing video games, the front of the player's car (which is relatively motionless in the scene) takes up a large part of the lower half of the screen, whereas the upper half of the screen is entirely filled with the oncoming roadway, buildings and scenery, which is almost always in motion. If the compression logic 404 allocates an equal number of bits to each tile, then the tiles on the bottom half of the screen (tiles 4-7) in uncompressed frame 721 in FIG. 7 b, will generally be compressed with higher quality than tiles than the tiles in the upper half of the screen (tiles 0-3) in uncompressed frame 721 in FIG. 7 b. **If this particular game, or this particular scene of the game is known to have such characteristics, then the operators of the hosting service 210 can configure the compression logic 404 to allocate more bits to the tiles in the top of the screen than to tiles at the bottom of the screen.** Or, the compression logic 404 can evaluate the quality of the compression of the tiles after frames are compressed (using one or more of many compression quality metrics, such as Peak Signal-To-Noise Ratio (PSNR)) and if it determines that over a certain window of time, certain tiles are consistently producing better quality results, then it gradually allocates more bits to tiles that are producing lower quality results, until the various tiles reach a similar level of quality. In an alternative embodiment, the compressor logic 404 allocates bits to achieve higher quality in a particular tile or group of tiles. For example, it may provide a better overall perceptual

appearance to have higher quality in the center of the screen than at the edges.

Ex.1006, [0177].

119. In such a scenario, Perlman suggests configuring the video compression logic 404 “to allocate more bits to the tiles in the top of the screen than to tiles at the bottom of the screen.” Ex.1006, [0177]. Accordingly, because some tiles have more allocated bits, they are encoded at a higher bit rate (“*different encoding characteristics*”). Perlman also indicates that the compression logic may use different processors or other hardware components (“*encoders*”). *See supra*, [1.3].

120. Thus, because Perlman describes allocating more bits to the tiles at the top of the screen than those at the bottom, Perlman renders obvious “*wherein the frame segments are encoded in encoders having different encoding characteristics*” as claimed.

11. Claim 11

[11.1] *A method according to claim 10, wherein at least one of the encoders has a bitrate that is greater than a bitrate of the other encoders.*

121. As described above at [10.1], Perlman describes a technique in which some tiles are allocated more bits, and are thus encoded at a higher bitrate. Thus, Perlman renders obvious “*wherein at least one of the encoders has a bitrate that is greater than a bitrate of the other encoders*” as claimed.

12. Claim 12

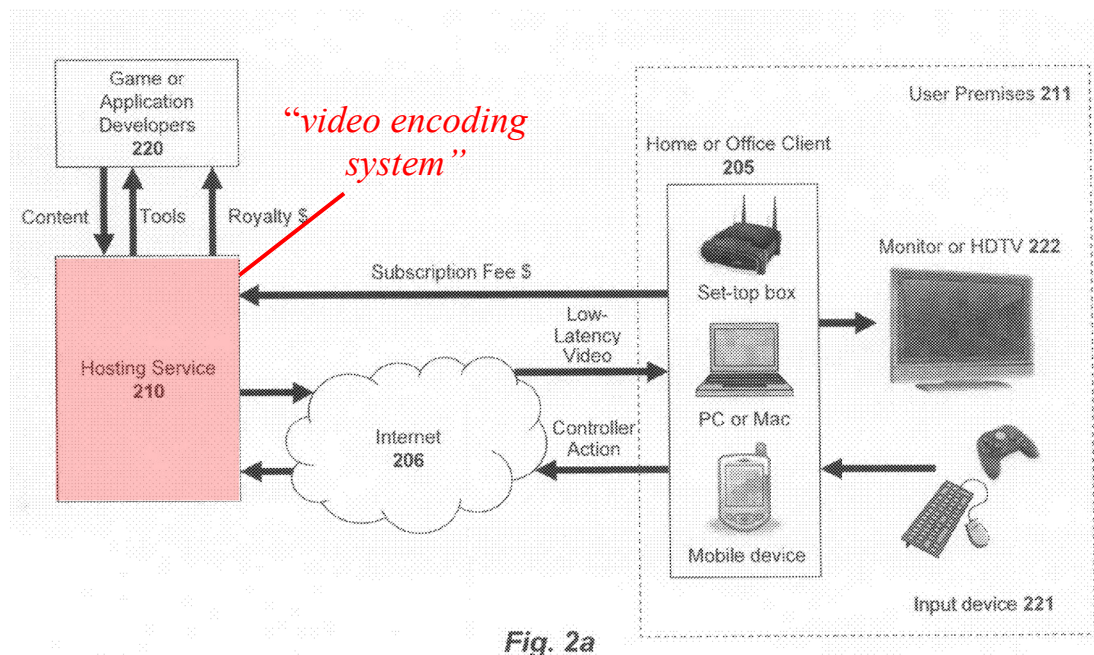
[12.0] *A video encoding system for encoding in real time a video signal that is associated with one of a videoconference and a telepresence session, the system comprising:*

122. First, Perlman describes a hosting service 210 for compressing real-time video signals (“*video encoding system for encoding in real time a video signal*”).

During playing of a game or use of an application on the hosting service 210, a home or office client device 205 receives keyboard and/or controller input from the user, and then it transmits the controller input through the Internet 206 to the **hosting service 210 that executes the gaming program code in response and generates successive frames of video output (a sequence of video images) for the game or application software** (e.g., if the user presses a button which would direct a character on the screen to move to the right, the game program would then create a sequence of video images showing the character moving to the right). **This sequence of video images is then compressed using a low-latency video compressor, and the hosting service 210 then transmits the low-latency video stream through the Internet 206.** The home or office client device then decodes the compressed video stream and renders the decompressed video images on a monitor or TV. Consequently, the computing and graphical hardware requirements of the client device 205 are significantly reduced. The client 205 only needs to have the processing power to forward the keyboard/controller input to the Internet 206 and decode and decompress a compressed video stream received from the Internet

206, which virtually any personal computer is capable of doing today in software on its CPU (e.g., a Intel Corporation Core Duo CPU running at approximately 2 GHz is capable of decompressing 720p HDTV encoded using compressors such as H.264 and Windows Media VC9). And, in the case of any client devices, dedicated chips can also perform video decompression for such standards in real-time at far lower cost and with far less power consumption than a general-purpose CPU such as would be required for a modern PC. Notably, to perform the function of forwarding controller input and decompressing video, home client devices 205 do not require any specialized graphics processing units (GPUs), optical drive or hard drives, such as the prior art video game system shown in FIG. 1.

Ex.1006, [0103].



Ex.1006, Fig. 2a (annotated).

123. As explained above at [1.0], compressing video data is “encoding”

video data as claimed. As also explained above at [1.0], Perlman's video streams are "*real time*" streams as claimed.

124. Second, for the reasons explained above at [1.0], Perlman explains that its techniques performed by the encoding system can be used for video conferencing or as part of a video game system that includes video conferencing ("*a video signal that is associated with a videoconference or telepresence session*"). See, e.g., Ex.1006, [0259].

125. Thus, because Perlman describes a system for compressing video data using a codec, and that video stream can be used for video conferencing, Perlman renders obvious a "*video encoding system for encoding in real time a video signal that is associated with one of a videoconference and a telepresence session*" as recited in the claim.

[12.1] a port for receiving a stream of video data comprising a series of sequential video frames;

126. First, Perlman's hosting server ("*system*") includes compression logic 404 that receives the graphical and audio output of the executed game or application software ("*a stream of video data*"). See *supra*, [1.1].

In addition, in one embodiment, **the hosting service 210 employs low-latency video compression logic 404 described in detail below.** The video compression logic 404 may be implemented in software, hardware, or any combination thereof (certain embodiments of which

are described below). Video compression logic 404 includes logic for compressing audio as well as visual material.

Ex.1006, [0116].

The graphical and audio output of the executed game or application software—i.e., new sequences of video images—are provided to the low-latency video compression logic 404 which compresses the sequences of video images according to low-latency video compression techniques, such as those described herein and transmits a compressed video stream, typically with compressed or uncompressed audio, back to the client 415 over the Internet 410 (or, as described below, over an optimized high speed network service that bypasses the general Internet). Low-latency video decompression logic 412 on the client 415 then decompresses the video and audio streams and renders the decompressed video stream, and typically plays the decompressed audio stream, on a display device 422. Alternatively, the audio can be played on speakers separate from the display device 422 or not at all. Note that, despite the fact that input device 421 and display device 422 are shown as free-standing devices in FIGS. 2 a and 2 b, they may be integrated within client devices such as portable computers or mobile devices

Ex.1006, [0117].

127. For the reasons explained above at [1.1], Perlman's video stream "*compris[es] a series of sequential video frames.*"

128. Second, a POSITA would have understood that the physical component that receives the input video is a "port" within the commonly

understood meaning of the term. *See e.g.*, Ex.1009, 844 (“Port ... An interface point connecting a communications channel and a device.”).

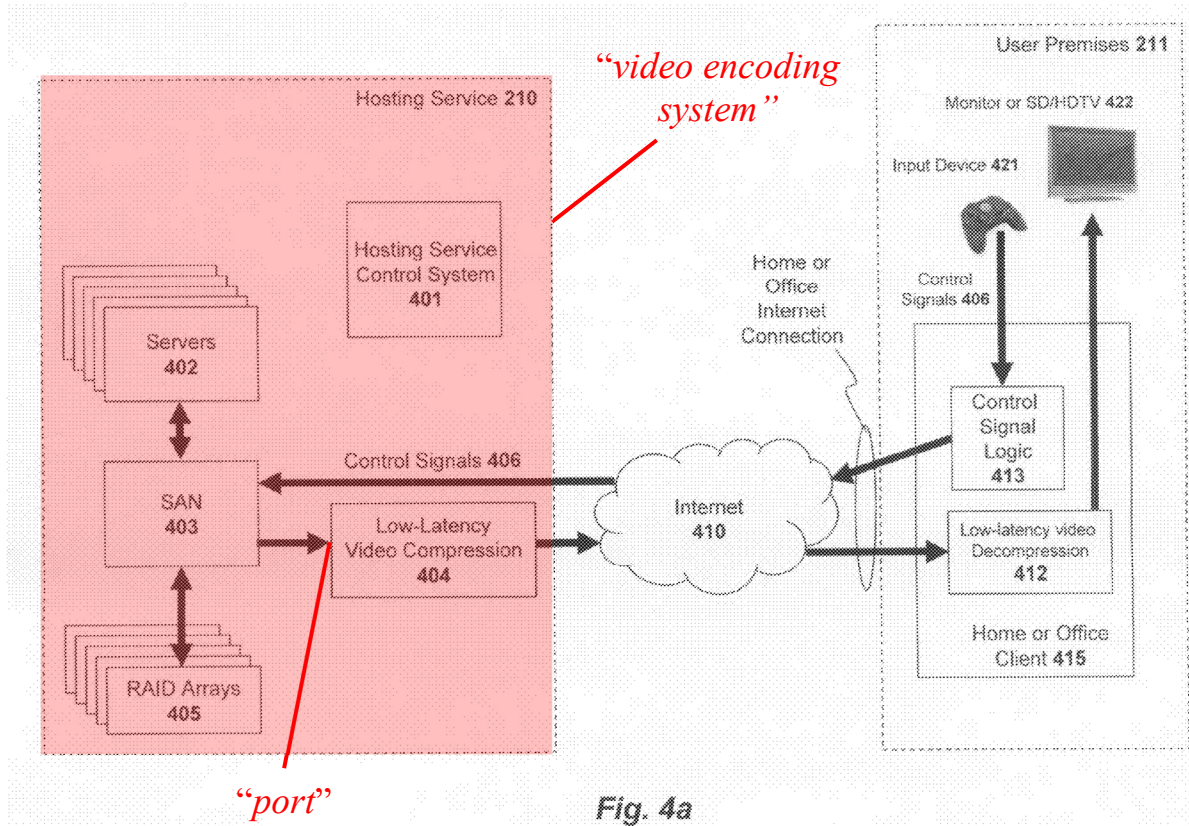


Fig. 4a

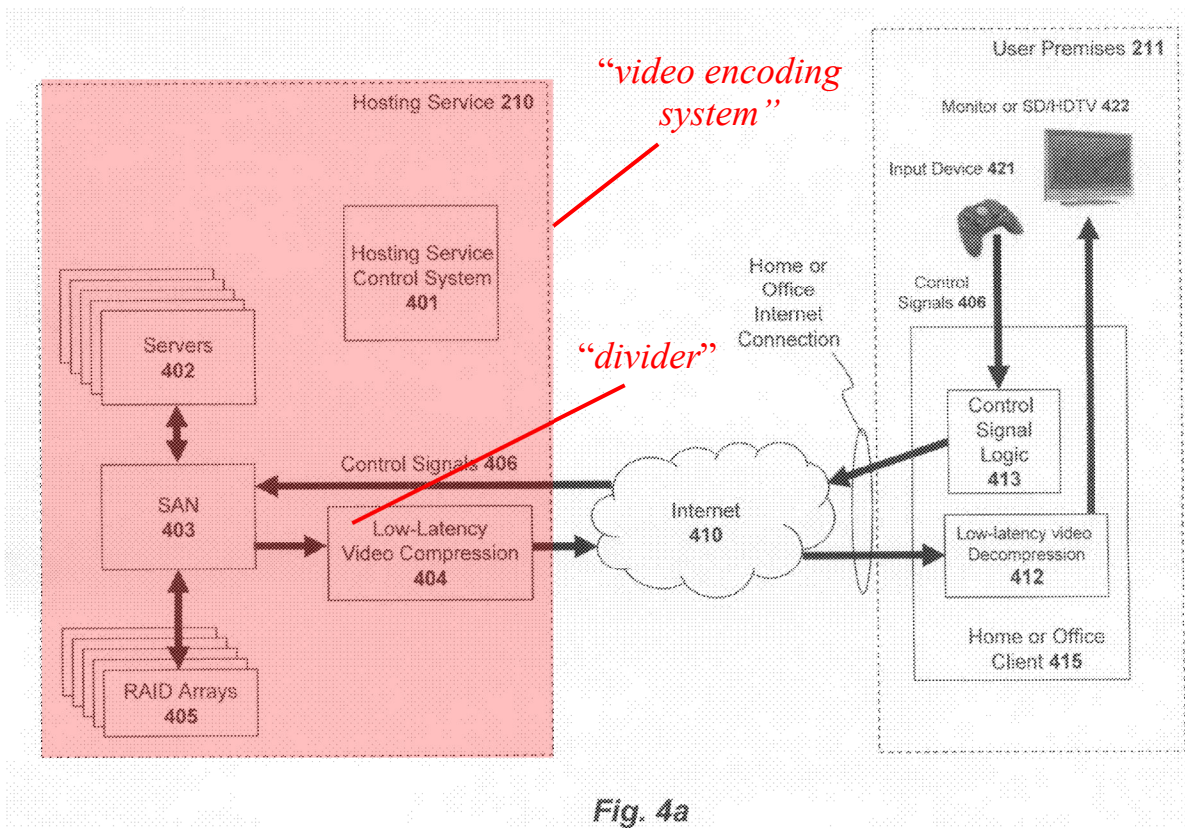
Ex.1006, Fig. 4a (annotated).

129. Thus, because Perlman describes a system that includes compression logic that receives audio and video output from an application, Perlman renders obvious “a port for receiving a stream of video data comprising a series of sequential video frames” as claimed.

[12.2] a divider for dividing the stream of video data into a series of substreams of data corresponding to frame segments, wherein each frame segment represents a part of a complete video frame;

130. First, for the reasons explained above at [1.2]-[1.4], Perlman’s compression logic 404 divides the frame into a sequence of separate tiles (“*series of substreams of data corresponding to frame segments*”) and encodes each substream to form parallel encoded streams.

131. Second, a POSITA would have understood that the hardware/software functionality of compression logic 404 that divides the frames into tiles teaches a “divider” as claimed. The output of the divider—and input to the encoder—are the “*series of substreams of data corresponding to frame segments.*”



Ex.1006, Fig. 4a (annotated).

132. Thus, because Perlman's system includes compression hardware and software that divides a video stream into separately encoded streams, Perlman renders obvious "*a divider for dividing the stream of video data into a series of substreams of data corresponding to frame segments, wherein each frame segment represents a part of a complete video frame*" as claimed.

[12.3] *a plurality of video encoders arranged in parallel for independently encoding the respective frame segments to generate parallel encoded data streams, one for each of the plurality of frame segments,*

133. For the reasons described above at [1.3] Perlman describes encoding the tiles in parallel, which may be done using several processors, FPGAs, or other hardware or software components ("*plurality of video encoders*").

134. Thus, because Perlman describes encoding each tile in parallel to create parallel encoded streams, one for each tile, Perlman renders obvious "*a plurality of video encoders arranged in parallel for independently encoding the respective frame segments to generate parallel encoded data streams, one for each of the plurality of frame segments*" as claimed.

[12.4] *each encoded data stream containing I-frame segment instances that carry complete information about a particular frame segment and P-frame segment instances that carry partial information about a particular frame segment, the partial information representing information that has changed relative to a previous frame segment instance in that encoded data stream, and*

135. This limitation is substantially similar to [1.5]. Thus, for the reasons explained above at [1.5], Perlman renders this limitation obvious.

[12.5] *the video encoders being arranged to encode each frame segment such that when an instance of one of the frame segments within a particular frame is an I-frame segment instance, at least one of the other frame segment instances in that particular frame is a P-frame segment instance,*

136. This limitation is substantially similar to that of [1.6]. As discussed above at [12.4], Perlman's system includes video encoders. Thus, for the reasons explained above at [12.4] and [1.6], Perlman renders obvious "*the video encoders being arranged to encode each frame segment such that when an instance of one of the frame segments within a particular frame is an I-frame segment instance, at least one of the other frame segment instances in that particular frame is a P-frame segment instance*" as claimed.

[12.6] *whereby in the respective parallel encoded data streams in any period of time when an I-frame segment instance is present in one of the parallel encoded data streams, a P-frame segment instance is present in at least one of the other parallel encoded data streams; and*

137. This limitation is substantially similar to [1.7]. Thus, for the reasons explained above at [1.7], the prior art renders this limitation obvious.

[12.7] *a transceiver for transmitting the parallel encoded data streams independently over a network to a receiver to permit the receiver to decode each frame segment and recombine the decoded frame segments to reproduce an image;*

138. First, Perlman describes transmitting the encoded video data over a network.

In operation, while playing a video game or using an application at the user premises 211 via a keyboard, mouse, game controller or other input

device 421, control signal logic 413 on the client 415 transmits control signals 406 a-b (typically in the form of UDP packets) representing the button presses (and other types of user inputs) actuated by the user to the hosting service 210. The control signals from a given user are routed to the appropriate server (or servers, if multiple servers are responsive to the user's input device) 402. As illustrated in FIG. 4 a, control signals 406 a may be routed to the servers 402 via the SAN. Alternatively or in addition, control signals 406 b may be routed directly to the servers 402 over the hosting service network (e.g., an Ethernet-based local area network). Regardless of how they are transmitted, the server or servers execute the game or application software in response to the control signals 406 a-b. Although not illustrated in FIG. 4 a, various networking components such as a firewall(s) and/or gateway(s) may process incoming and outgoing traffic at the edge of the hosting service 210 (e.g., between the hosting service 210 and the Internet 410) and/or at the edge of the user premises 211 between the Internet 410 and the home or office client 415. The graphical and audio output of the executed game or application software—i.e., new sequences of video images—are provided to the low-latency video compression logic 404 which compresses the sequences of video images according to low-latency video compression techniques, such as those described herein and **transmits a compressed video stream, typically with compressed or uncompressed audio, back to the client 415 over the Internet 410** (or, as described below, over an optimized high speed network service that bypasses the general Internet). Low-latency video decompression logic 412 on the client 415 then decompresses the video and audio streams and renders the decompressed video stream, and

typically plays the decompressed audio stream, on a display device 422. Alternatively, the audio can be played on speakers separate from the display device 422 or not at all. Note that, despite the fact that input device 421 and display device 422 are shown as free-standing devices in FIGS. 2 a and 2 b, they may be integrated within client devices such as portable computers or mobile devices.

Ex.1006, [0117].

139. A POSITA would have found it obvious to transmit the data using a “*transceiver*.” Perlman provides an explicit example in which a network interface is a “transceiver.”

The WAN interface 442 may be a Cable or DSL modem, a **WiMax transceiver, a Fiber transceiver**, a Cellular data interface, a Internet Protocol-over-powerline interface, or any other of many interfaces to the Internet. Further, Firewall/Router/NAT device 443 (and potentially WAN interface 442) may be integrated into the client 415. An example of this would be a mobile phone, which includes software to implement the functionality of home or office client 415, as well as the means to route and connect to the Internet wirelessly through some standard (e.g., 802.11g).

Ex.1006, [0143]. Moreover, a POSITA would have found it obvious to use network interface devices that both receive and transmit data because video-conferencing involves two-way communication.

140. Second, for the reasons explained above at [1.8], Perlman’s

transmitter is “for transmitting the parallel encoded data streams independently over a network to a receiver to permit the receiver to decode each frame segment and recombine the decoded frame segments to reproduce an image.”

[12.8] wherein during transmission of I-frame segment instances of one of the frame segments, P-frame segment instances of all other frame segments are transmitted.

141. This limitation is substantially similar to [1.9]. Thus, for the reasons explained above at [1.9], the prior art renders this limitation obvious.

13. Claim 13

[13.1] A video encoding system according to claim 12 comprising: a memory for storing predetermined data defining the frame segments, which correspond to predetermined parts of the video frames.

142. First, for the reasons explained above at [2.1] and [3.1], Perlman describes dividing a frame into a pre-determined arrangement (i.e., a four-by-four grid).

143. Second, Perlman’s hosting service 110 (“system”) includes various types of memory.

In one embodiment, the servers 402 are diskless, i.e., rather than having its own local mass storage (be it optical or magnetic storage, or semiconductor-based storage such as **Flash memory** or other mass storage means serving a similar function), each server accesses shared mass storage through fast backplane or network connection. In one embodiment, this fast connection is a **Storage Area Network (SAN) 403 connected to a series of Redundant Arrays of Independent**

Disks (RAID) 405 with connections between devices implemented using Gigabit Ethernet. As is known by those of skill in the art, a SAN 403 may be used to combine many RAID arrays 405 together, resulting in extremely high bandwidth-approaching or potentially exceeding the bandwidth available from the **RAM** used in current gaming consoles and PCs. And, while RAID arrays based on rotating media, such as magnetic media, frequently have significant seek-time access latency, RAID arrays based on semiconductor storage can be implemented with much lower access latency. In another configuration, some or all of the servers 402 provide some or all of their own mass storage locally. For example, a server 402 may store frequently-accessed information such as its operating system and a copy of a video game or application on low-latency local Flash-based storage, but it may utilize the SAN to access RAID Arrays 405 based on rotating media with higher seek latency to access large databases of geometry or game state information on a less frequent bases.

Ex.1006, [0115].

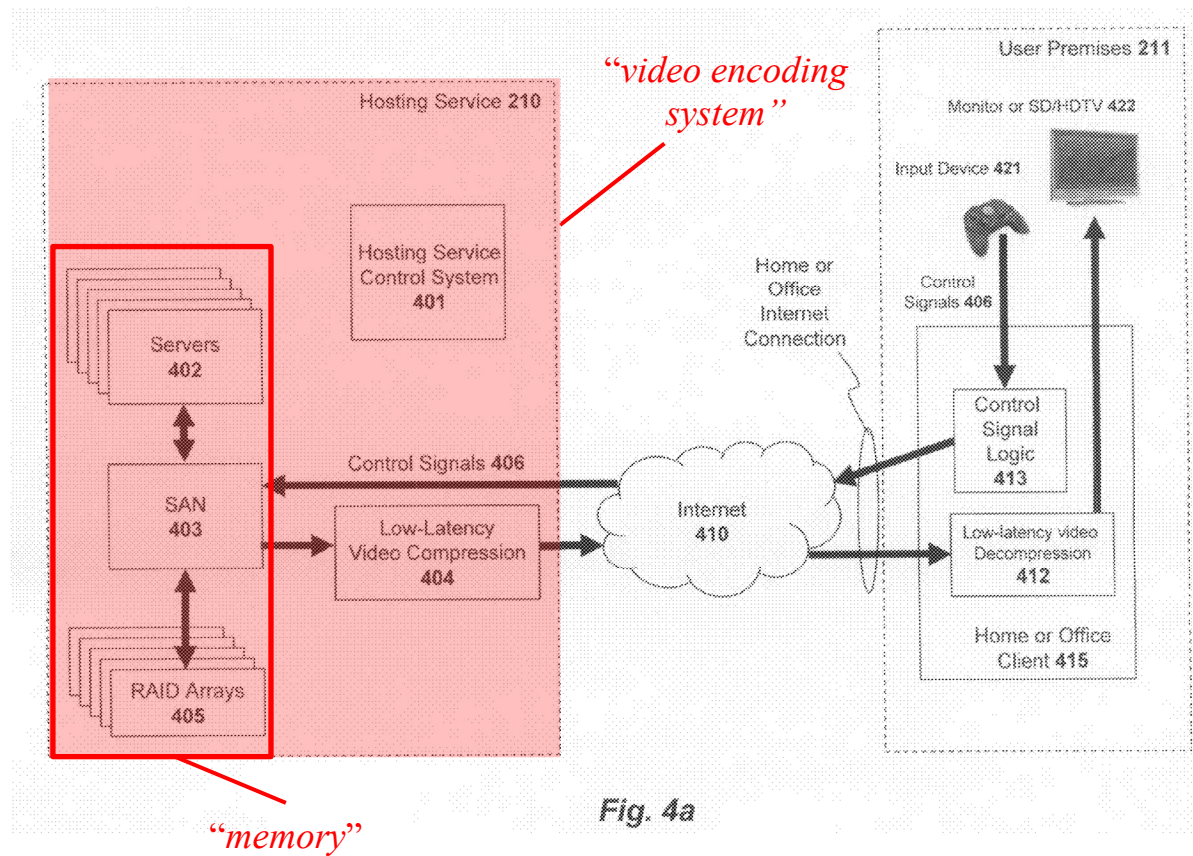


Fig. 4a

Ex.1006, Fig. 4a (annotated).

144. **Third**, a POSITA would have found it obvious to store any data used by the compression logic in such memory. Indeed, Perlman explains that the techniques it discloses may be embodied on machine readable media.

Elements of the disclosed subject matter may also be provided as a machine-readable medium for storing the machine-executable instructions. The machine-readable medium may include, but is not limited to, **flash memory, optical disks, CD-ROMs, DVD ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, propagation media or other type of machine-readable media suitable for storing electronic instructions.** For example, the present

invention may be downloaded as a computer program which may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection)

Ex.1006, [0358].

145. Thus, because Perlman's system includes a memory which may be used to stored data and machine-readable instructions related to dividing a frame into a predetermined four-by-four tile grid, Perlman renders obvious "*a memory for storing predetermined data defining the frame segments, which correspond to predetermined parts of the video frames.*"

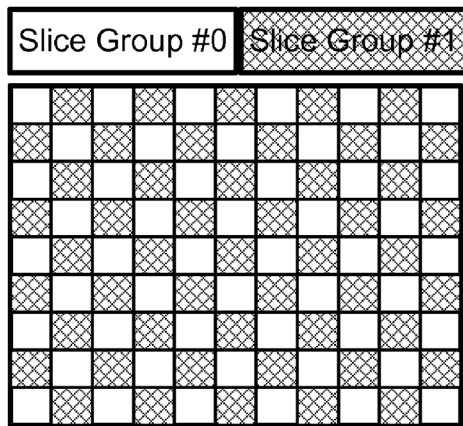
B. Ground 2: Claim 6 is obvious in view of Perlman and Oguz.

1. Summary of Oguz

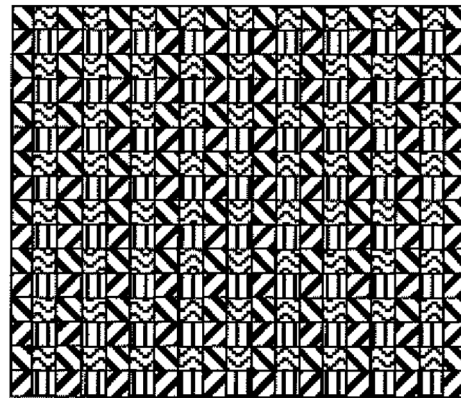
146. Oguz "relates generally to video encoding and, more specifically, to techniques for content adaptive video frame slicing and non-uniform access unit coding for improved coding efficiency." Ex.1008, [0003]. Oguz explains that "multimedia processing systems, such as video encoders, may encode multimedia data using encoding methods based on international standards such as Moving Picture Experts Group (MPEG)-1, -2 and -4 standards, the International Telecommunication Union (ITU)-T H.263 standard, and the ITU-T H.264 standard and its counterpart, ISO/IEC MPEG-4, Part 10, i.e., Advanced Video Coding

(AVC).” Ex.1008, [0042].

147. Like both Perlman and the ’426 patent, Oguz explains that “each frame or field may further include one or more slices, or sub-portions of the frame or field.” Ex.1008, [0043]. Oguz illustrates various “slice partitioning examples.” For example, Oguz illustrates an example where the frame slices are in a checkerboard pattern, as shown below in comparison with a similar checkerboard example from the ’426 patent.



Ex.1008, Fig. 4B.



Ex.1001, Fig. 5 (partial)

FIG. 4B illustrates a sample frame 300 with Type 1 FMO based on partitioning according to the H.264/AVC standard. **In FIG. 4B, the frame 300 includes a checker-board pattern for improved error resilience and concealment.** For illustrative purposes, those macroblocks that are depicted as white are associated with slice group # 0. Those macroblocks that are depicted as black are for slice group # 1. Thus, there is an alternating pattern which can be affected through the use of FMO. This is permitted because the FMO no longer requires

the slices to consist of neighboring macroblocks. **Thus, the checker-board pattern essentially provides scattered slices.**

Ex.1008, [0065].

The FMO of the H.264/AVC standard comprises seven different types, labeled Type 0-Type 6. However, for illustrative purposes only Types 1 and 2 are described here to provide examples of the slice structure. The FMO for the purposes of error resiliency allows macroblocks to be ordered in a manner that no macroblock is surrounded by any other macroblock from the same slice group. Hence, in the event an error occurs (e.g. a slice gets lost during transmission), reconstruction of the missing blocks can rely on the information of the available surrounding macroblocks. Type 6 FMO is the most random type. Type 6 FMO allows full flexibility to the user. The other FMO types such as Type 0-Type 5 are restricting in that a certain pattern must be followed.

Ex.1008, [0066].

148. Thus, Oguz provides evidence that it was known to divide a frame into a checkerboard pattern in which a particular slice group (“*segment*”) represents a non-contiguous area.

2. Reasons to Combine Oguz with Perlman

149. A POSITA implementing Perlman’s frame dividing techniques would have found it obvious to do so in a variety of manners such as those described by either Perlman or Oguz. For example, a POSITA would have found it obvious to implement Perlman’s frame slicing method so that particular slices are non-

contiguous as taught by Oguz.

150. While Perlman provides an example in which a video frame is divided into a four-by-four grid of 16 tiles, Perlman explains that other slice geometries may be used. For example, Perlman states that “various different sizes and shapes may be used for the tiles while still complying with these underlying principles.” Ex.1006, [0169]. “Note that the tiles need not be square, nor must the video frame be square, or even rectangular. The tiles can be broken up into whatever shape best suits the video stream and the application used.” Ex.1006, [0174].

151. Accordingly, a POSITA would have found it obvious to divide a frame according to any known, suitable alternatives to the four-by-four grid described by Perlman. Oguz provides one example of such an alternative by placing the tiles in a checkerboard pattern: “the frame 300 includes a checkerboard pattern for improved error resilience and concealment.” Ex.1008, [0065].

152. A POSITA would have had a reasonable expectation of success in combining the teachings of Oguz with those of Perlman. Oguz explains that the checkerboard pattern is useful for the H.264 codec. Perlman lists the H.264 codec as one example of a codec that may be used with its teachings. Ex.1006, [0156] (listing “types of compression, **such as H.264**, or Windows Media VC9, MPEG2 and MPEG4.”). Accordingly, when using the H.264 codec, a POSITA would have had expected success in dividing a frame into two segments positioned in a

checkerboard pattern.

153. Thus, the combination of Oguz with Perlman is an example of using a known technique (checkerboard pattern) to improve similar methods (Perlman's frame dividing technique) in the same way (using the H.264 codec).

3. Claim 6

[6.1] A method according to claim 1, wherein at least one of the frame segments comprises a plurality of predetermined non-contiguous areas of the video frames.

154. A POSITA would have found it obvious to divide the video frame into two slices put into a checkerboard pattern as evidenced by Oguz.

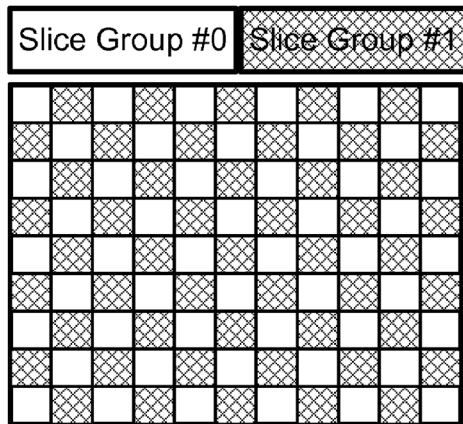
FIG. 4B illustrates a sample frame 300 with Type 1 FMO based on partitioning according to the H.264/AVC standard. **In FIG. 4B, the frame 300 includes a checker-board pattern for improved error resilience and concealment.** For illustrative purposes, those macroblocks that are depicted as white are associated with slice group # 0. Those macroblocks that are depicted as black are for slice group # 1. Thus, there is an alternating pattern which can be affected through the use of FMO. This is permitted because the FMO no longer requires the slices to consist of neighboring macroblocks. **Thus, the checker-board pattern essentially provides scattered slices.**

Ex.1008, [0065].

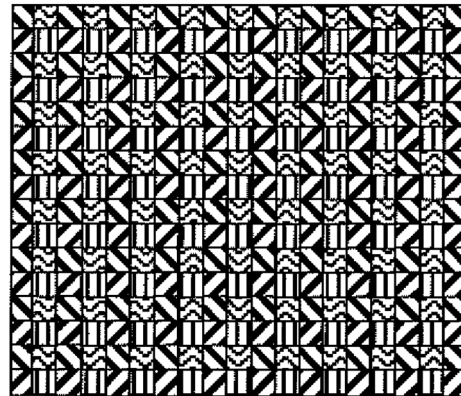
The FMO of the H.264/AVC standard comprises seven different types, labeled Type 0-Type 6. However, for illustrative purposes only Types 1 and 2 are described here to provide examples of the slice structure.

The FMO for the purposes of error resiliency allows macroblocks to be ordered in a manner that no macroblock is surrounded by any other macroblock from the same slice group. Hence, in the event an error occurs (e.g. a slice gets lost during transmission), reconstruction of the missing blocks can rely on the information of the available surrounding macroblocks. Type 6 FMO is the most random type. Type 6 FMO allows full flexibility to the user. The other FMO types such as Type 0-Type 5 are restricting in that a certain pattern must be followed.

Ex.1008, [0066].



Ex.1008, Fig. 4B.



Ex.1001, Fig. 5 (partial)

155. When the frame is divided into such a pattern, the two different slices comprise “a plurality of predetermined non-contiguous areas of the video frames.” For the reasons explained above at VIII.B.3, a POSITA would have found it obvious to implement Perlman’s techniques in a variety of ways, including a checkboard pattern as described by Oguz.

156. Thus, because Perlman describes dividing a frame into a variety of

sizes and shapes, and Oguz provides an example of dividing a frame into a checkerboard pattern, Perlman and Oguz render obvious “*wherein at least one of the frame segments comprises a plurality of predetermined non-contiguous areas of the video frames*” as claimed.

IX. CONCLUSION

157. This declaration and my opinions herein are made to the best of my knowledge and understanding, and based on the material available to me, at the time of signing this declaration. I declare that all statements made herein on 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 or Title 18 of the United States Code.

Date: March 14, 2024I. Freedman