

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

LIGHTRICKS LTD.,
Petitioner,

v.

PLOTAGRAPH, INC. and SASCHA CONNELLY,
Patent Owner.

Case IPR2023-00153
U.S. Patent No. 10,346,017
Filing Date: Aug. 13, 2018
Issue Date: July 9, 2019

**SECOND DECLARATION OF
PHILIP GREENSPUN, PH.D.**

TABLE OF CONTENTS

I.	INTRODUCTION	7
II.	“IN A LOOP” SHOULD BE CONSTRUED ACCORDING TO THE TERM’S PLAIN AND ORDINARY MEANING.....	7
A.	The plain and ordinary meaning of “in a loop” includes resetting and repeating the shifting of a set of pixels.....	7
B.	The specification does not limit “in a loop.”.....	10
C.	The POR’s construction for “in a loop” is inconsistent with the specification.....	12
III.	AEM TEACHES APPLYING “LOOPOUT()” TO A DEFORM PIN AND WHAT RESULTS.	14
IV.	IMU AND AEM TEACH “SHIFTING” A “SET OF PIXELS.”	15
A.	The Specification Discloses that IMV6’s Shepard’s Distortion “shift[s]” a “set of pixels.”	16
B.	The Specification Discloses that the Triangular Mesh Algorithm of AECS6’s Puppet Effect “shift[s]” a “set of pixels.”.....	22
1.	Affine Transformations.....	23
2.	Affine Transformations as Utilized in Triangular Mesh Algorithms	27
3.	Triangular Mesh Algorithms as Utilized in the ’017 Patent.....	31
4.	AECS6’s Puppet Effect Utilizes the Same Type of Triangular Mesh Algorithm as the ’017 Patent.	36
C.	Okabe’s Animation Technique was not Relied Upon for the Claimed “shifting.”	42
V.	THE IMU-OKABE COMBINATION SHIFTS A SET OF PIXELS “ALONG [A] NON-LINEAR PATHWAY.”	43
A.	IMU Discloses the Claim Language by Teaching Incrementally Moving Shepard’s Distortion Control Points Along a Curved Path.	44
B.	The IMU-Okabe Combination Repurposes Okabe’s Strokes to Allow the Curved Path to be Defined Visually.....	50

VI.	STROKES IN THE IMU-OKABE COMBINATION WOULD HAVE BEEN USER-VIEWABLE.	51
VII.	THE IMU-OKABE COMBINATION’S MATTE PERFORMS “MASK[ING]” OF PIXELS.....	52
VIII.	AECS6’S MASK/MATTE PERFORMS “MASK[ING]” OF PIXELS AND “SELECTING”/“IDENTIFYING” OF A “SET OF PIXELS.”	53
A.	AEM Teaches Applying AECS6’s Mask/Matte, Which Performs the Claimed “mask[ing].”	53
B.	AEM’s Mask/Matte Disclosure Likewise Teaches the Claimed “selecting”/“identifying” of a “set of pixels.”	54
IX.	CONCLUSION.....	56

LIST OF DOCUMENTS CONSIDERED

Exhibit	Shorthand	Description
1001	'017 Patent	U.S. Patent No. 10,346,017
1002	Greenspun	Declaration of Philip Greenspun, Ph.D.
1003	AEM	Adobe® After Effects® Help and tutorials, Adobe (2013)
1004	IMU-Home	Wayback Machine Capture dated Mar. 27, 2012 of Anthony Thyssen, <i>Examples of ImageMagick Usage (Version 6)</i> , ImageMagick (Mar. 15, 2011), http://www.imagemagick.org/Usage/ [https://web.archive.org/web/20120327064501/http://www.imagemagick.org/Usage/]
1005	IMU-Distorting	WayBack Machine Capture dated Mar. 29, 2012 of Anthony Thyssen, <i>ImageMagick v6 Examples - - Distorting Images</i> , ImageMagick (Mar. 21, 2012), http://www.imagemagick.org/Usage/distorts/ [https://web.archive.org/web/20120329131929/http://www.imagemagick.org/Usage/distorts/]
1006	IMU-Masking	Wayback Machine Capture dated Sept. 28, 2012 of Anthony Thyssen, <i>ImageMagick v6 Examples - - Masks</i> , ImageMagick (Mar. 10, 2011), http://www.imagemagick.org/Usage/masking/ [https://web.archive.org/web/20120928070642/http://www.imagemagick.org/Usage/masking/]
1007	IMU-Animating	Wayback Machine Capture dated Mar. 10, 2012 of Anthony Thyssen, <i>ImageMagick v6 Examples - - Animation Basics</i> , ImageMagick (Feb. 8, 2011), http://www.imagemagick.org/Usage/anim_basics/ [https://web.archive.org/web/20120310193613/http://www.imagemagick.org/Usage/anim_basics/]
1008	IMU-Windows ¹	Wayback Machine Capture dated Apr. 5, 2012 of Anthony Thyssen, <i>ImageMagick v6 Examples --</i>

¹ IMU-Home, IMU-Warping, IMU-Distorting, IMU-Masking, IMU-Animating, and IMU-Windows are collectively referred to hereinafter as “IMU.”

		<i>Usage under Windows</i> , ImageMagick (Mar. 21, 2012), http://www.imagemagick.org/Usage/windows/ [https://web.archive.org/web/20120405151502/http://www.imagemagick.org/Usage/windows/]
1009	Okabe	Makato Okabe, et al., <i>Creating Fluid Animation from a Single Image using Video Database</i> , 30 Computer Graphics Forum 1973 (2011)
1010	Li	Yin Li, et al., <i>Lazy Snapping</i> , 23 ACM Transactions on Graphics 303 (2004)
1011	'017 PH	Prosecution history of U.S. Patent No. 10,346,017
1012	Archive	Declaration of Nathaniel E Frank-White of the Internet Archive
1022		Plaintiffs' Unopposed Sworn Motion for Leave to Extend Time to Comply with P.R. 3-1 and P.R. 3-2 Dislcoure [sic] of Asserted Claims and Preliminary Infringement Contentions and Make Document Production, <i>Plotagraph, Inc. v. Lightricks Ltd.</i> , Civil Action No. 4:21-cv-03873, Dkt. No. 38 (S.D. Tex. May 5, 2022)
1041	Kernighan	Brian W. Kernighan & Dennis M. Ritchie, <i>The C Programming Language</i> (2d ed. 1988)
1042		Wayback Machine Capture dated Apr. 4, 2012 of <i>Inverse Distance Weighted Interpolation</i> , http://www.ems-i.com/smshelp/Data_Module/Interpolation/Inverse_Distance_Weighted.htm [https://web.archive.org/web/20120404084929/http://www.ems-i.com/smshelp/Data_Module/Interpolation/Inverse_Distance_Weighted.htm]
1043	Wampler	U.S. Patent App. Pub. No. 2017/0098326
1044	Igarashi	Takeo Igarashi et al., <i>As-Rigid-As-Possible Shape Manipulation</i> , 24 ACM Transactions on Graphics 1134 (July 1, 2005)
1045	Efros1	Wayback Machine Capture dated Sept. 9, 2015 of Alexei Efros, <i>Image Warping, 15-463: Computational Photography</i> , Carnegie Mellon University, Computer Science Department (Sept. 20, 2007), http://graphics.cs.cmu.edu/courses/15-463/2007_fall/Lectures/warping.pdf [https://web

		.archive.org/web/20070622221140/http://graphics.cs.cmu.edu/courses/15-463/2006_fall/www/Lectures/warping.pdf]
1046	Efros2	Wayback Machine Capture dated Sept. 10, 2015 of Alexei Efros, <i>Image Warping and Morphing, 15-463: Computational Photography</i> , Carnegie Mellon University, Computer Science Department (Sept. 25, 2007), http://graphics.cs.cmu.edu/courses/15-463/2007_fall/Lectures/morphing.pdf [https://web.archive.org/web/20150910054926/http://graphics.cs.cmu.edu/courses/15-463/2007_fall/Lectures/morphing.pdf]
1047		Wayback Machine Capture dated Sept. 10, 2015 of <i>15-463: Computational Photography</i> , Carnegie Mellon University, Computer Science Department (Fall 2007), http://graphics.cs.cmu.edu/courses/15-463/2007_fall/ [https://web.archive.org/web/20150910011432/http://graphics.cs.cmu.edu/courses/15-463/2007_fall/]
1048	IMU-Affine	Wayback Machine Capture dated May 25, 2012 of Anthony Thyssen, <i>ImageMagick v6 Examples - Affine Matrix Transforms</i> , ImageMagick (Oct. 10, 2009), http://www.imagemagick.org/Usage/distorts/affine/ [https://web.archive.org/web/20120525065408/http://www.imagemagick.org:80/Usage/distorts/affine/]
1049	Dong	European Patent No. 2,706,507 B1
1050		<i>Lightricks Ltd. v. Plotagraph, Inc.</i> , IPR2023-00569, Paper 8 (PTAB Oct. 2, 2023)

I. INTRODUCTION

1. My name is Philip Greenspun. This declaration expands on the conclusions that I have formed based on my analysis provided in my previous declaration. *See generally* Greenspun (EX1002).

II. “IN A LOOP” SHOULD BE CONSTRUED ACCORDING TO THE TERM’S PLAIN AND ORDINARY MEANING.

2. Paper 11 (“POR”) in this proceeding proposes “that the claim language shifting of a set of pixels by rendering and re-rendering ‘in a loop’ be construed as meaning that a leading pixel in the selected set of pixels be rendered at the starting point of the shift and behind one or more trailing pixels which it previously was leading.” POR, 21. In my opinion however, and consistent with my previous declaration (*see, e.g.*, Greenspun, ¶42), the plain and ordinary meaning of “in a loop” is broader, encompassing also the resetting and repeating of shifting a set of pixels that I discussed in my previous declaration. Although the POR points to the specification’s “examples” to support its narrow construction, these are indeed only “examples,” which do not indicate that “in a loop” means something different from the term’s plain and ordinary meaning.

A. The plain and ordinary meaning of “in a loop” includes resetting and repeating the shifting of a set of pixels.

3. In my opinion, the plain and ordinary meaning of “in a loop” includes shifting a set of pixels and, upon completion of this shift, resetting the set back to its original position and repeating the shift.

4. Under the POR’s construction, the claimed “in a loop” is satisfied only when a pixel “loops around any other pixels *during* shifting.” POR, 43.² However, a POSITA would have recognized that, within the relevant realm of “computer science, electrical engineering, [and] related field[s]” (*see* Greenspun, ¶24), the word “loop” is not confined to—and indeed includes more than—such looping. For instance, “while and for *loops*,” which were widely used in nearly all programming languages, are control statements that execute a set of code, and then—*not during but only afterwards*—loop by returning to the beginning of the set of code to repeat the execution until a condition is met. Kernighan, 74-77.

5. Further, within the specific sub-realm of “image processing and animation” including in “existing systems for image processing and animation” (*see* Greenspun, ¶24), a POSITA would have recognized that the word “loop” continues to hold a broader meaning than what the POR proposes. Indeed, AEM and IMU, which describe such then-existing systems, both teach—using the exact word “loop”—a type of looping in which a set of pixels is, *after shifting*, reset back to the set’s original position, from which the shift is then repeated.

6. As I explained in my previous declaration, AEM teaches that AECS6’s “*loopOut()*” expression by default causes “all keyframes [to] *loop*.” AEM, 562;

² Emphasis added unless indicated otherwise.

Greenspun, ¶¶62, 106. Thus, when applied to a Puppet effect Deform pin, the “loopOut()” expression causes the Deform pin and surrounding pixels in the mesh to shift along the defined motion path and to *only thereafter* reset to repeat this shift in what AEM expressly states is a “*loop*.” AEM, 562, 219; Greenspun, ¶¶62, 106.

7. Similarly, as I also explained in my previous declaration, IMU teaches that IMV6’s “-*loop*” operator specifies the “[n]umber of times [a] GIF animation is to cycle *th[r]ough* the image sequence before stopping.” IMU-Animating, 2; Greenspun, ¶152. Specifically, IMU teaches that a GIF animation comprises a canvas, within which frames are “overlaid” or “cleared” on the canvas over time, and where, regardless, “the whole canvas is always cleared *at the end* of the animation sequence, *before it loops and repeats*.” IMU-Animating, 4-6. The “-loop” operator specifies how many such “loops” should occur—i.e., how many times the GIF repeats *after* playing its image sequence from beginning to end—where the default “zero” value causes an “infinite *loop*.” IMU-Animating, 2; *see also* Greenspun, ¶152. As such, when applied to a GIF animation created using, e.g., IMV6’s Shepard’s Distortion, the “-loop” operator by default causes the Shepard’s-Distortion-shifted pixels to reset *at the end* of the animation sequence, *not before*, and the shifting to repeat in what IMU expressly states is an “infinite *loop*.” Greenspun, ¶¶150-52, 167.

8. Thus, both AEM and IMU demonstrate that, when shifting a set of pixels—whether with AECS6’s Puppet effect or IMV6’s Shepard’s Distortion—the pixels “loop” when, after the set has been shifted, the set is merely reset and the shifting is repeated. While other ways to loop may exist, AEM and IMU nonetheless each illustrate a way that a POSITA would have recognized to be shifting “in a loop” under the term’s plain and ordinary meaning.

B. The specification does not limit “in a loop.”

9. As for the POR’s construction, the claims themselves contain no language reflecting what the POR proposes. While the POR states that “in a loop” requires specific positioning of “a leading pixel” and “one or more trailing pixels” (POR, 20-28), no claims recite any such “leading pixel,” “trailing pixels,” or any interaction between the two. Rather, the claims merely recite “rendering and re-rendering in a loop the first set of pixels being shifted.”

10. The POR nevertheless states that, “because all the described embodiments” perform what the POR defines as looping, the meaning of “in a loop” should be confined to such looping. POR, 21-24.

11. However, the specification does not provide any such definition for “in a loop,” nor excludes other known ways of looping. By contrast, all disclosed embodiments are expressly declared in the specification as being *exemplary*.

12. For instance, regarding the “primary illustrated embodiment” in FIGS. 5A-5D (*see* POR, 21-23), the specification prefaces the corresponding discussion with, “[b]y way of *example*....” ’017 Patent, 7:20-21. And when describing specifically FIG. 5D, which apparently depicts a “loop around” (*see* POR, 22), the specification again states, “[f]or *example*, pixel 500a is at the left of the shift, while corresponding pixel 500b is reappearing at the right of the shift to begin the shifting loop,” and, “[f]or *example*, once pixel 500a reaches the end of the link (i.e., ending point 530), pixel 500a disappears on the next step. Upon disappearing, pixel 500b then reappears at the beginning of the link (i.e., starting point 550) to start the loop.” ’017 Patent, 7:66-8:4.

13. Similarly, regarding the “alternative embodiment” (*see* POR, 23-24), the corresponding disclosure likewise states this is just one of “*various* alternative or additional embodiments,” under which, “[f]or *example*, in at least one embodiment, the pixels immediately begin cycling as soon as the shifting begins. *For example*, instead of what is shown in FIGS. 5A-5D, as soon as pixel 500 shifted to its new position depicted in FIG. 5B, pixel 500B would appear in the position depicted in FIG. 5D.” ’017 Patent, 8:19-26.

14. Because the specification provides only illustrative, nonlimiting, exemplary embodiments, a POSITA would not have understood the specification to limit the meaning of “in a loop” to anything other than the term’s plain and ordinary

meaning. Indeed, the specification explicitly confirms that “[t]he described embodiments are to be considered in all respects *only as illustrative and not restrictive*,” such that “[t]he present invention may be embodied in *other specific forms*...” ’017 Patent, 13:57-60.

C. The POR’s construction for “in a loop” is inconsistent with the specification.

15. The POR’s narrow construction for “in a loop” is inconsistent with not only the term’s plain and ordinary meaning as explained above but also the specification, especially when coupled with the POR’s other statements.

16. Specifically, *the specification explicitly describes embodiments that loop after shifting a set of pixels using a “mesh algorithm”* (e.g., that of AECS6’s Puppet effect) *or “Shepard’s distortion”* (e.g., IMV6’s “Shepard’s Distortion”) (’017 Patent, 8:28-37)—both of which, as I explain later (and as the POR also acknowledges) “move[] pixels by stretching or deforming the image” (POR, 46; *infra* at Section IV). Yet, according to the POR, these pixel “stretching” techniques are completely “incapable of shifting a set of pixels ‘in a loop,’” at least under the POR’s understanding of “in a loop.” *E.g.*, POR, 46, 48, 2-3.

17. As such, despite the specification explicitly stating that mesh algorithms and/or Shepard’s distortion *can* be used in the claimed invention, the POR’s statements assert the exact opposite: that such pixel “stretching” techniques *cannot* be used. As such, the POR’s understanding of “in a loop” directly contradicts

the specification and therefore does not reflect what a POSITA would have understood from reading the specification.

18. My previous declaration, by contrast, did not exclude the use of such pixel “stretching” techniques. Instead, I not only relied on these techniques (as disclosed in IMU and AEM) for teaching the claimed “shifting” (*infra* at Section IV), but also explained how such shifting can be executed “in a loop”: by simply resetting and repeating the movement of an AECS6 Deform pin using a “loopOut()” expression, and by similarly resetting and repeating an IMV6 Shepard’s Distortion GIF animation using a “-loop” operator, respectively (Section II.A). Greenspun, ¶¶104-107, 181-83.

19. Further, even the specification itself suggests that such resetting and repeating qualifies “loop[ing]” as claimed. As the specification discloses, “[i]n various embodiments, a user is provided with an **animation duration** and FPS rate option,” under which “a user may indicate that **each respective loop** in the pixel shifting only **take 2 seconds**.” ’017 Patent, 7:36-45. Such disclosure suggests that “**loop[ing]**” an entire shifting “**animation**” after a duration—e.g., 2 seconds—qualifies as shifting “in a loop.”

20. Thus, unlike the POR’s narrow construction, the resetting and repeating of a shift of a set of pixels is consistent with both what a POSITA and the specification consider to be shifting “in a loop.”

III. AEM TEACHES APPLYING “LOOPOUT()” TO A DEFORM PIN AND WHAT RESULTS.

21. The POR states that AEM’s disclosures regarding AECS6’s “loopOut()” expression have not been explained. POR, 45 (noting AEM’s discussion of “looping a segment of time” and “the In point and Out point of [a] layer” (citing AEM, 562)). I, however, provided that explanation in my previous declaration, including in particular by quoting AEM’s high-level summary of how “loopOut()” works overall (and how a POSITA would have understood “loopOut()” works): that “loopOut()” by “default” causes “all keyframes [to] loop.” AEM, 562, *quoted in* Greenspun, ¶¶62, 106. As I also explained in my previous declaration, such a teaching therefore also teaches that applying “loopOut()” specifically to a Deform pin would by “default” would cause “all keyframes [*of the Deform pin* to] loop”—i.e., “the Deform pin’s position will be reset to the starting keyframe upon reaching the ending keyframe during keyframe interpolation, and the interpolation will be repeatedly looped then and thereafter until otherwise specified by the user....” Greenspun, ¶106. This exact result is also shown in the screenshots of the AECS6 animation that I generated and discussed in my previous declaration, which thus show visually what AEM teaches in words alone. Greenspun, ¶107, Attachment B. Indeed, the POR acknowledges that such is what is shown in my screenshots. *E.g.*, POR, 13 (“At Animation Time Point (3:01) the Deform pin has been returned to its starting point as the keyframes are reset to the beginning.”).

22. The POR also states that “no disclosure in AEM... suggest[s] applying a loopOut expression to a Deform pin.” POR, 45. In my previous declaration, however, I showed how AEM teaches applying “loopOut()” to a Deform pin by quoting AEM, 562—which teaches that “loopOut()” is an “*expression*”—and AEM, 219—which teaches that “[y]ou can use *expressions* to link the positions of *Deform pins* to motion tracking data, audio amplitude keyframes, or any other properties.” AEM, 562, 219, *quoted in* Greenspun, ¶106. Such teachings together sufficiently teach applying a “loopOut()” *expression* to a Deform pin.

IV. IMU AND AEM TEACH “SHIFTING” A “SET OF PIXELS.”

23. Regardless of the looping discussed above, the POR asserts that “IMU... does not teach *shifting of a set of pixels* but rather a *single* control point” (*e.g.*, POR, 38-39) and that AEM’s teachings are similarly deficient (*see, e.g.*, POR, 14-16 & n.1). The POR appears to acknowledge at least that a “*single*” pixel is “*shift[ed]*” by IMV6’s Shepard’s Distortion and AECS6’s Puppet effect, but the POR states that “[a]ny *other* pixels in an image are *translated* only as a result of the image being stretched,” not “shift[ed]” with the “single” pixel—i.e., the claimed “shifting” a “set of pixels.” POR, 38. In essence, the POR proposes that only a *pure-translation* meets the claimed “shifting”; if a pixel is also “stretched” or otherwise deformed when translated, then it is not “shift[ed].”

24. The specification, however, expressly states that “shifting” a “set of pixels” occurs even when pixels also undergo “stretch[ing].”

A. The Specification Discloses that IMV6’s Shepard’s Distortion “shift[s]” a “set of pixels.”

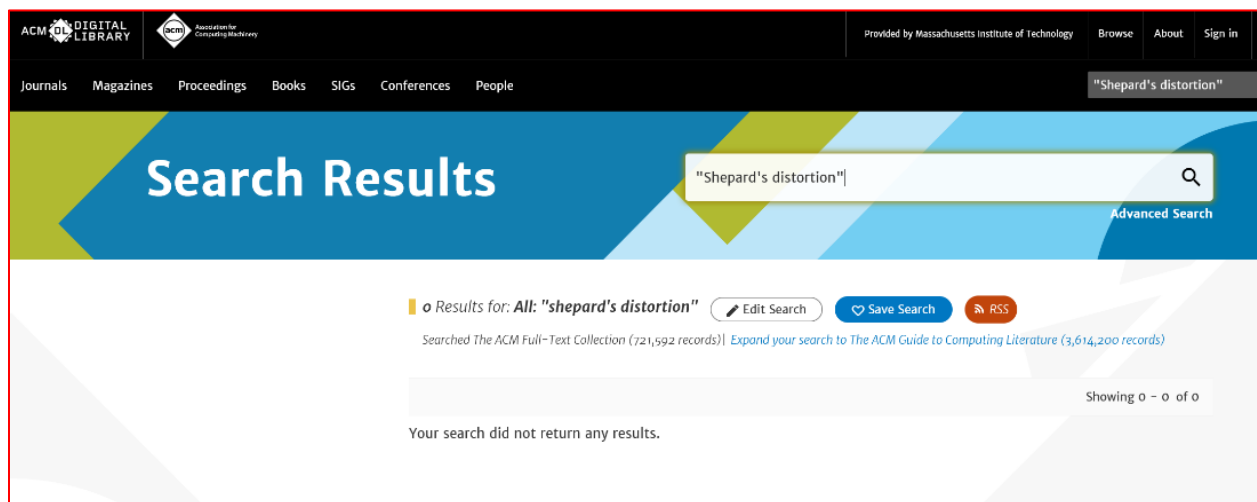
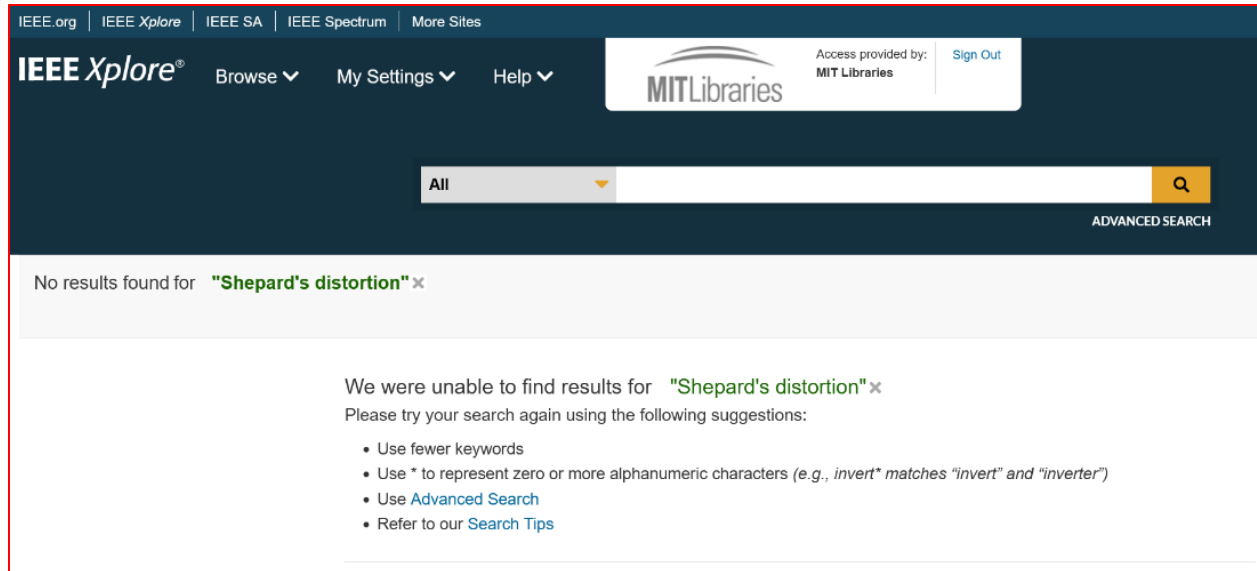
25. As I explained in my previous declaration, the specification discloses different exemplary ways to—as also recited in the claims—“*shift*” *not just a single pixel* but, explicitly, a “*set of pixels*,” including “alpha blending,” such as “utiliz[ing] a sin, cosine, or linear alpha blend”; “using a mesh algorithm”; “animation blending”; and, as most relevant here, “us[ing] a warping function, such as *Shepard’s distortion*.” ’017 Patent, 8:28-41; Greenspun, ¶181.

26. The specification never again discusses Shepard’s distortion. This, in my opinion, may have been an implicit acknowledgement that a POSITA already knew what Shepard’s distortion was and how it works. And indeed, “Shepard’s distortion” was a recognized term of art by the earliest claimed priority date (“ECPD”). The term’s namesake comes from a 1968 paper by Donald Shepard titled “A two-dimensional interpolation function for irregularly-spaced data,” in which Shepard describes a method for interpolating unknown data values between and around “irregularly-spaced” apart “data *points*” using a distance-based weighted

average. Shepard, 1-2; *see also* EX1042.³ Shepard's method can be used in "diverse" types of data contexts, such as in "mete[o]rology,... geography,... [and] biology" (Shepard, 1), and, in its general form, is commonly known as "Shepard's interpolation" or "Shepard's method" (*e.g.*, EX1042).

27. IMV6 utilizes a version of Shepard's interpolation, which IMU refers to as "Shepard's distortion." *E.g.*, IMU-Distorting, 59-62. Notably, aside from the specification, this "Shepard's distortion" terminology appears to be used **only** by IMU and IMV6—in fact, no search results are yielded when searching IEEE and ACM's academic literature databases using the search term "Shepard's distortion":

³ EX1042 is a Wayback Machine capture of a webpage linked to in IMU-Distorting. IMU-Distorting, 59 ("Inverse Squared Distance Interpolation"). Since IMU-Distorting was publicly accessible before the ECPD (*see* Greenspun, ¶44), so too was the linked webpage shown in EX1042. EX1042's capture date of April 4, 2012, as indicated by its Wayback Machine URL of https://web.archive.org/web/20120404084929/http://www.ems-i.com/smshelp/Data_Module/Interpolation/Inverse_Distance_Weighted.htm (*see* Archive, ¶5 (explaining URL)), confirms EX1042's pre-ECPD public availability. I printed the webpage at this URL and provided counsel with the printout that I understand is used as EX1042 in this proceeding.



By contrast, a search for “Shepard’s interpolation” in these databases yields numerous results:

IEEE Xplore® Browse ▾ My Settings ▾ Help ▾ MITLibraries Access provided by: MIT Libraries Sign Out

All [Search] ADVANCED SEARCH

Search within results [Search]

Download PDFs Items Per Page ▾ Export

Showing 1-19 of 19 results for "Shepard interpolation" ×

☐ Conferences (17) ☐ Journals (2)

Show

- ☒ All Results
- ☐ Subscribed Content ?
- ☐ Open Access Only

Year ▴

☒ Range ☐ Single Year

1993 2023

Clear Apply

Author ▾

Affiliation ▾

☐ Select All on Page

Sort By Relevance ▾

☐ **Fast inpainting-based compression: Combining Shepard interpolation with joint inpainting and prediction**

Pascal Peter

2019 IEEE International Conference on Image Processing (ICIP)

Year: 2019 | Conference Paper | Publisher: IEEE

Cited by: Papers (4)

Abstract HTML PDF CC

☐ **Sliding block shepard interpolation method for removing high-density impulse noise**

Chaipichit Cumpim; Rachu Punchalard

2017 International Electrical Engineering Congress (iEECON)

Year: 2017 | Conference Paper | Publisher: IEEE

Cited by: Papers (1)

ACM DIGITAL LIBRARY Association for Computing Machinery Provided by Massachusetts Institute of Technology Browse About Sign In

Journals Magazines Proceedings Books SIGs Conferences People

"Shepard's interpolation"

Search Results "Shepard's interpolation" [Search] Advanced Search

People

- Names ▾
- Institutions ▾
- Authors ▾
- Reviewers ▾

Publications

- Journal/Magazine Names ▾
- Proceedings/Book Names ▾
- All Publications ▾
- Content Type ▾

45 Results for All: "shepard's interpolation" Edit Search Save Search RSS

Searched The ACM Full-Text Collection (721,592 records) | Expand your search to The ACM Guide to Computing Literature (3,614,200 records)

RESULTS VIDEOS Showing 1 - 20 of 45 Results

☐ Select All per page: 10 20 50 Relevance ▾

☐ **RESEARCH-ARTICLE** Consistent shepard interpolation for SPH-based fluid animation

November 2019

Stefan Reinhardt, Tim Krake, Bernhard Eberhardt, Daniel Weiskopf

ACM Transactions on Graphics (TOG), Volume 38, Issue 6 • Article No.: 189, pp 1–11 • <https://doi-org.libproxy.mit.edu/10.1145/3355089.3356503>

We present a novel technique to correct errors introduced by the discretization of a fluid body when animating it with smoothed particle hydrodynamics (SPH). Our approach is based on the Shepard correction, which reduces the interpolation errors from ...

7 321 Highlights ▾

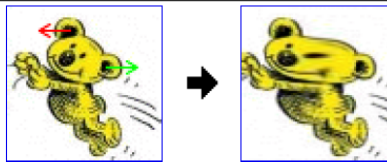
Thus, the fact that the specification explicitly recites “Shepard’s distortion” perhaps indicates that the specification is referring specifically to *IMV6’s Shepard’s Distortion* and stating that such may be used to shift pixels. *See* ’017 Patent, 8:28-41. Nevertheless, the underlying method of “Shepard’s distortion” remains the same as what is described in Shepard: “Shepard’s distortion”—as it appears in both the specification and IMU—remains a “*point driven*” technique that, at a high-level, is “equivalent to a thick block of ‘taffy’ representing [a] source image[] having pins driven into it and then the pins moved around” (i.e., pins, or “control points,” are irregularly-spaced apart). *Compare* IMU-Distorting, 62, 59, *with* Shepard, 1.

28. Hence, Shepard’s distortion “does not move areas, but *points*,” and “will swirl and stretch and compress the image between the control points, but it tries hard not to rotate or scale the image near the control points” (i.e., interpolates pixel values based on distance to a control point). *Compare* IMU-Distorting, 59-60, 62, *with* Shepard, 1-2. Stated differently, when applying Shepard’s distortion to an image, the movement of only some pixels is explicitly specified with a vector while, by contrast, the movement of other pixels is inferred via Shepard’s interpolation from the explicitly specified vectors, causing these other pixels to be stretched, compressed, and reoriented.

29. Such Shepard’s distortion, as IMU teaches, is utilized in IMV6’s “Shepard’s Distortion” effect. As I explained in my previous declaration, IMU

provides an example that depicts an image of a koala being “pulled and stretched apart” by two IMV6 Shepard’s Distortion “control *points*” that, after being placed, are moved apart from one another by a specified amount (i.e., 10 pixels):

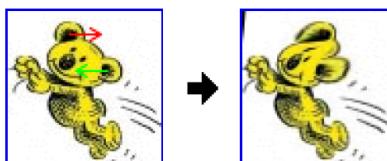
```
convert koala.gif -virtual-pixel Black \
-distort Shepards '30,11 20,11 48,29 58,29' \
koala_ear_pull.png
```



IMU-Distorting, 59; Greenspun, ¶¶150, 172. Indeed, as shown, only the control points move the specified 10-pixel amount in IMV6; all other surrounding pixels are “stretched” or “compressed” based on distance to the control points. IMU-Distorting, 59.

30. IMU also provides a reverse example, which instead “push[es]” the control points “toward each other” by the same 10-pixel amount:

```
convert koala.gif -virtual-pixel Black \
-distort Shepards '30,11 40,11 48,29 38,29' \
koala_ear_push.png
```



IMU-Distorting, 61-62. Again, only the control points move the specified 10-pixel amount, while surrounding pixels this time stretch and compress based on distance to the control points, creating here an overall “swirl” effect. IMU-Distorting, 61.

31. And although IMV6's Shepard's Distortion as shown causes *only control points* to translate while instead causing surrounding pixels to “swirl and stretch and compress” (IMU-Distorting, 59-60, 62), such Shepard's distortion nonetheless meets the claimed “*shifting*,” *not just of one pixel* but, again, per the specification and as explained in my previous declaration, a “*set of pixels*.” '017 Patent, 8:28-41; Greenspun, ¶181. Indeed, the POR agrees with the above, not only by agreeing that Shepard's distortion is a “*pixel-based shifting* mechanism,” but also using this as a basis for distinguishing from the animation technique taught in Okabe. POR, 37; *see also infra* at Section IV.C

B. The Specification Discloses that the Triangular Mesh Algorithm of AECS6's Puppet Effect “shift[s]” a “set of pixels.”

32. As discussed, the specification makes clear that, in addition to Shepard's distortion, “using a *mesh algorithm*”—particularly one that “triangulates the mesh using defined points and then calculates an affine transformation for every angle”—is also “shifting” a “set of pixels” as claimed. '017 Patent, 8:28-41; Greenspun, ¶104.

33. Such “triangulat[ion]”-and-“affine[-]transformation”-based mesh algorithms were well-known by the ECPD. *E.g.*, Wampler, ¶¶[0002]-[0003] (discussing how “[e]xisting software applications allow[ed] users to animate electronic characters and other artwork” “using a mesh of triangles”); Igarashi (describing one such mesh algorithm). Indeed, the specification appears to

acknowledge that a POSITA already knew how the disclosed mesh triangulation and affine transforms work because the specification never provides such details. Nevertheless, as a POSITA would have understood, such mesh algorithms operate by not only translating pixels but also stretching, compressing, and reorienting pixels at the same time, particularly when calculating the disclosed “affine transformation[s].”

1. Affine Transformations

34. In affine transformations, an input image’s pixels are “map[ped]” to new positions in an output image by applying a “linear” function to each pixel that combines “rotation, shear, scale, and translation,” “perform[ing] all four of the given distortions all at the same time.” IMU-Affine,⁴ 11, 1 (noting also that such is how

⁴ IMU-Affine is a Wayback Machine capture of a subpage of the ImageMagick.com website’s “Examples of ImageMagick Usage (Version 6)” section (*see* Greenspun, ¶44) and was accessible by the interested public before the ECPD because it, like EX1042, was linked to in IMU-Distorting (*see* IMU-Distorting, 1 (“Affine Matrix Transforms (separate sub-directory)”), 23 (“[S]ee the Affine Matrix transforms sub-page.”)). IMU-Affine’s capture date by the Wayback Machine on March 29, 2012, as indicated by its Wayback Machine URL of <https://web.archive.org/web/20120329131929/http://www.imagemagick.org/Usage>

affine transforms work “both in [IMV6] *and all other Image Processors*”); accord Efron¹,⁵ 24, 16, 21; Wampler, ¶[0003]; Igarashi, 7. Thus, although the input

/distorts/, confirms IMU-Affine’s pre-ECPD public availability. I printed the webpage at this URL and provided counsel with the printout that I understand is used as the “IMU-Affine” exhibit in this proceeding.

⁵ Efron¹ as well as Efron² consist of lecture slides presented in 2007 for Carnegie Mellon University Computer Science Department’s “15-463 (15-862): Computational Photography” course, as indicated by the course’s website (EX1047, 2), which provides links to both Efron¹ and Efron² (EX1047, 5-6 (“Th Sept 20 Image Warping Slides (ppt, pdf)[;] Tu Sept 25 Image Morphing Slides (ppt, pdf),” respectively). The course’s website was accessible by the interested public before the ECPD, as shown by the Wayback Machine’s archive of the website on September 6, 2015, at https://web.archive.org/web/20150906000017/http://graphics.cs.cmu.edu/courses/15-463/2007_fall/. And thus, so too were the linked lecture slides of Efron¹ and Efron²; indeed, these lecture slides were also Wayback-Machine archived in September 2015 on the 9th (https://web.archive.org/web/20150909230549/http://graphics.cs.cmu.edu/courses/15-463/2007_fall/Lectures/warping.pdf) and 10th (https://web.archive.org/web/20150910054926/http://graphics.cs.cmu.edu/courses/15-463/2007_fall/Lectures/morphing.pdf), respectively. These

image's pixels are "preserve[d]" in the output image, an affine transform nevertheless causes *more* than only a "translation" of these pixels. *See* IMU-Affine, 12, 1.

35. Beginning with the affine transform's "rotation" distortion, rotation, as self-explanatory, causes the alignment of each pixel to reorient by a given angle. IMU-Affine, 4. Efros1 provides an example in which an image of a house is rotated:



rotation

Efros, 3-4. As shown, the image's pixels have been reoriented clockwise with respect to one another.

lecture slides' pre-ECPD public availability is also confirmed by subsequent, pre-ECPD-dated citing references. *E.g.*, Dong, code (56). I downloaded Efros1 and Efros2 from their respective Wayback Machine URLs and provided counsel with copies that I understand are used as the "Efros1" and "Efros2" exhibits in this proceeding.

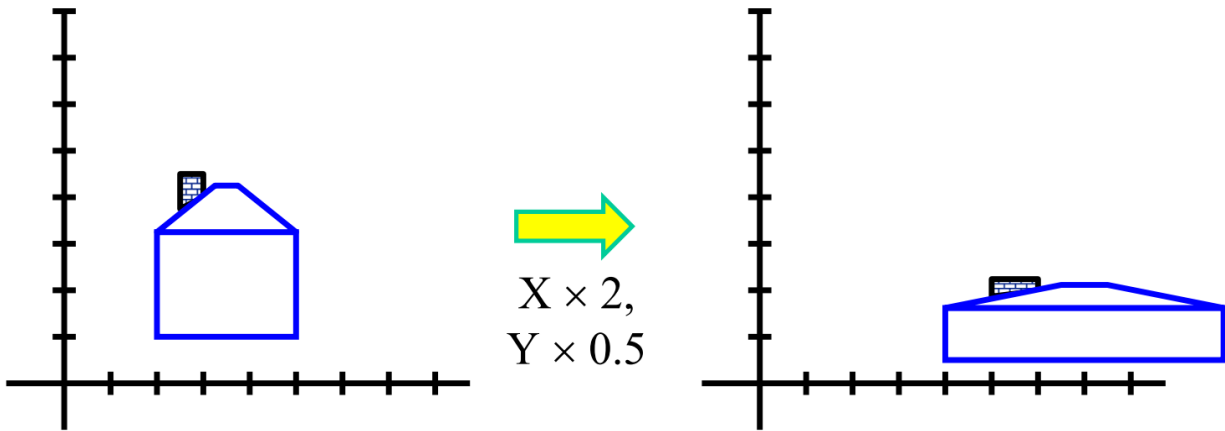
36. Similar to rotation, the “shear” distortion also causes pixels to be reoriented, but does so by sliding opposing halves of an input image in perpendicular but opposite directions (e.g., pixels in the upper and lower half of the image are slid respectively left and right, or right and left), resulting in the output image taking on a parallelogram shape. *See* IMU-Affine, 4; Efros1, 13. Efros1 provides an example of shearing the same house image above:



affine

Efros, 3-4. As shown, the house image has been horizontally sheared—the pixels in the upper half of the image have been shifted rightward while the pixels in the lower half have been shifted leftward—causing the image taking on a parallelogram shape as a result of the mapping.

37. Finally, “scale,” as implied by its name, scales the input image’s dimensions such that, when mapped to the output image, the pixels stretch and/or compress, as illustrated in Efros1’s scaling example below:



Efros1, 7; IMU-Affine, 3. As can be seen, the depicted house’s width has been scaled up by two (“ $X \times 2$ ”), whereas its height has been scaled down by half (“ $Y \times 0.5$ ”), causing the house’s pixels (most noticeably in the chimney) to be stretched horizontally but compressed vertically.

38. Affine transformations combine “pure translation”—which “only moves an image to another location without distortion”—with each of the three above distortions—which, as explained, reorient, stretch, and compress pixels. See IMU-Affine, 6, 1; IMU-Warping, 24, 16. The above thus illustrates that affine transformations do *more* than simply and purely translate an image’s pixels; they also at the same time reorient (via rotation and shear) and stretch and compress (via scale) these same pixels.

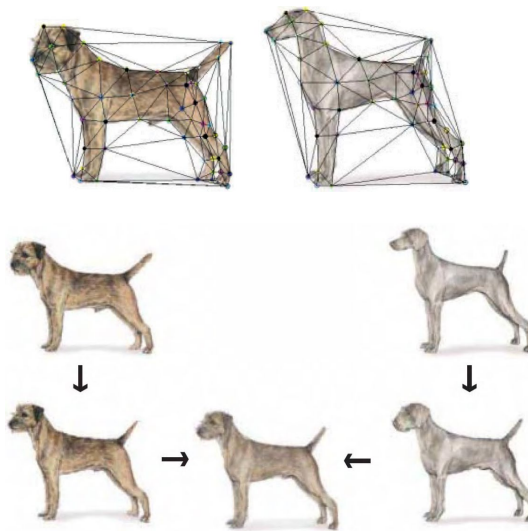
2. Affine Transformations as Utilized in Triangular Mesh Algorithms

39. When used in typical triangular mesh algorithms, affine transforms still continue to reorient, stretch, compress, and translate pixels as discussed above, but

do so to triangular subsets of pixels in an image on a subset-by-subset basis, rather than all of the image's pixels at once. *See, e.g.*, Igarashi, 7 (describing how, in a triangular mesh algorithm, “a linear mapping from an original triangle [in a starting mesh] to a deformed triangle [in an ending mesh] ignoring the translation factor can be represented by a 2×2 affine transformation matrix... represented as a combination of a rotation part R_γ , a shearing part s_h , and a scaling part s_x, s_y ”). Specifically, in such triangular mesh algorithms: (1) a suitable “mesh of vertices forming triangles” is defined for an image (determined using, e.g., “Delaunay triangulation”); (2) user inputs are received that select and translate some of these vertices, known thereafter as “handles”; (3) the unselected (and therefore non-handle) vertices are dragged along with the handles but in varying amounts (e.g., according to “bounded biharmonic weights”), thereby causing the mesh's triangles to deform; and (4) an affine transformation is calculated for each triangle separately, thereby mapping

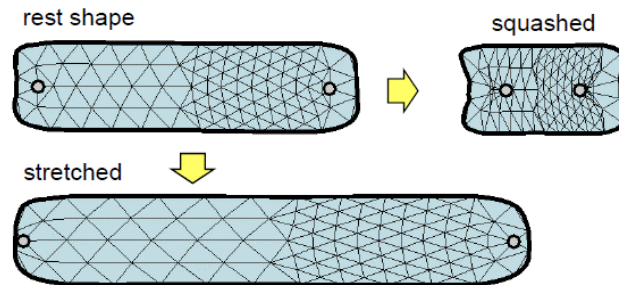
each triangle's pixels to their new locations accordingly. *E.g.*, Igarashi, 2-3, 7; Wampler, ¶¶[0037], [0002]-[0004]; Efros2, ⁶ 23-26, 29.

⁶ Efros2 describes how mesh algorithms are used in “morphing”—an animated effect that smoothly converts one image into another. *See* Efros2, 26. Efros2 provides an example where a Border Terrier is morphed into a Weimaraner:



Efros2, 26, 21. When morphing, the discussed steps (1)-(4) of mesh algorithms are still performed, except that (i) a mesh is defined for **both** images, each mesh having an equal number of vertices placed on matching “key feature points” in each image (e.g., mesh vertices are defined on the eye, nose, tail, and other key parts of each dog above); (ii) one image’s vertices are moved to where the corresponding vertices are located in the other (e.g., the Border Terrier’s nose and tail vertices are moved to where the Weimaraner’s nose and tail are located, and vice versa); and (iii) the

40. Igarashi provides an example of applying a mesh algorithm to an elongate shape:

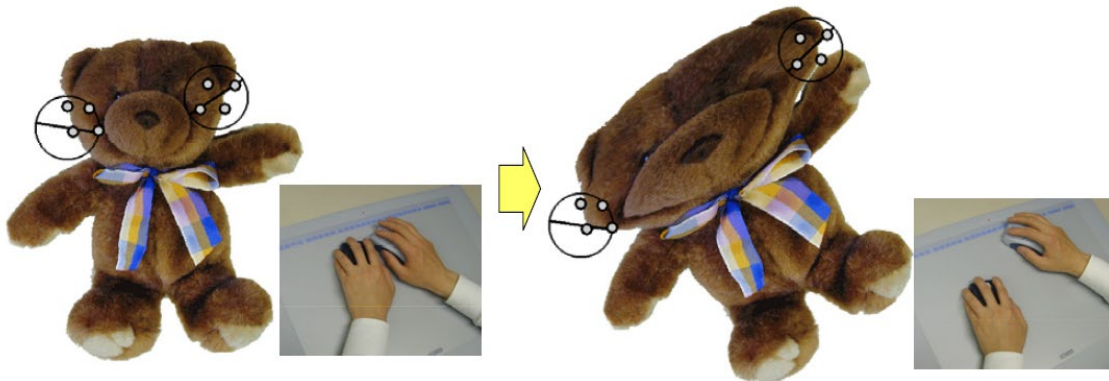


Igarashi, 7. As shown in the shape’s “rest shape,” two mesh vertices have been selected to be handles (grey dots), which are located on horizontally-opposing ends of the shape. And when “stretched,” the handles as shown have been translated to move horizontally apart, causing each of the mesh’s triangles to also translate horizontally *but also increase in width* (as particularly shown by the horizontal edge of each of the large triangles growing in length), thereby causing each triangle’s pixels to also translate horizontally *but also stretch horizontally* due to the

images are “cross-dissolve[d]”—i.e., one image’s alpha (nontransparency) is decreased while the other’s is increased. Efros2, 24-26, 35-36, 21. Still, because a mesh algorithm is used, an affine transform continues to be calculated for each triangle in each mesh. Efros2, 26. Indeed, the Border Terrier’s neck stretches and torso compresses when morphing into the Weimaraner, and vice versa, particularly as a result of scaling.

associated affine transformations (particularly, a horizontal scale up). The reverse—“squashed”—is also depicted, which shows the same handles translated horizontally but towards each other, causing the shape’s pixels to not only translate horizontally as well **but also compress horizontally** (particularly, due to a horizontal scale down).

41. Igarashi also provides a further example of applying the same mesh algorithm but to a teddy bear image—a “real 3D object[.]”—to better illustrate how an image’s pixels are **more than** translated by the mesh algorithm:



See Igarashi, 6-7. As illustrated, although the mesh generated for the teddy bear is not visible, the teddy bear’s pixels have nonetheless been translated, **as well as stretched and rotated**, by the placement and movement of eight handles (grey dots) and the calculation of associated underlying affine transformations in the mesh algorithm. See Igarashi, 2.

3. Triangular Mesh Algorithms as Utilized in the ’017 Patent

42. As illustrated and discussed above, in triangular mesh algorithms, while handles are translated, the other pixels of the mesh are translated as well **but also**

reoriented, stretched, and compressed in accordance with the handle movement.

See Wampler, ¶[0061] (“[T]he system recognizes the movement of one or more *handles as translations* and then figures out *the rest of the parameters of an affine transformation* for each of the handles.”). Yet, per the ’017 Patent’s specification, using such mesh algorithms—i.e., the translation of handles and the corresponding translation *and* reorienting, stretching, and compressing of the mesh’s pixels—nevertheless constitutes “shift[ing]” a “set of pixels” as claimed. ’017 Patent, 8:28-41.

43. Such is shown by comparing the prior discussion on triangular mesh algorithms with the ’017 Patent specification’s FIG. 4, the latter of which likewise depicts an image’s pixels—including, as relevant here, sets of pixels associated with various links—being *more than* translated by a triangular mesh algorithm:

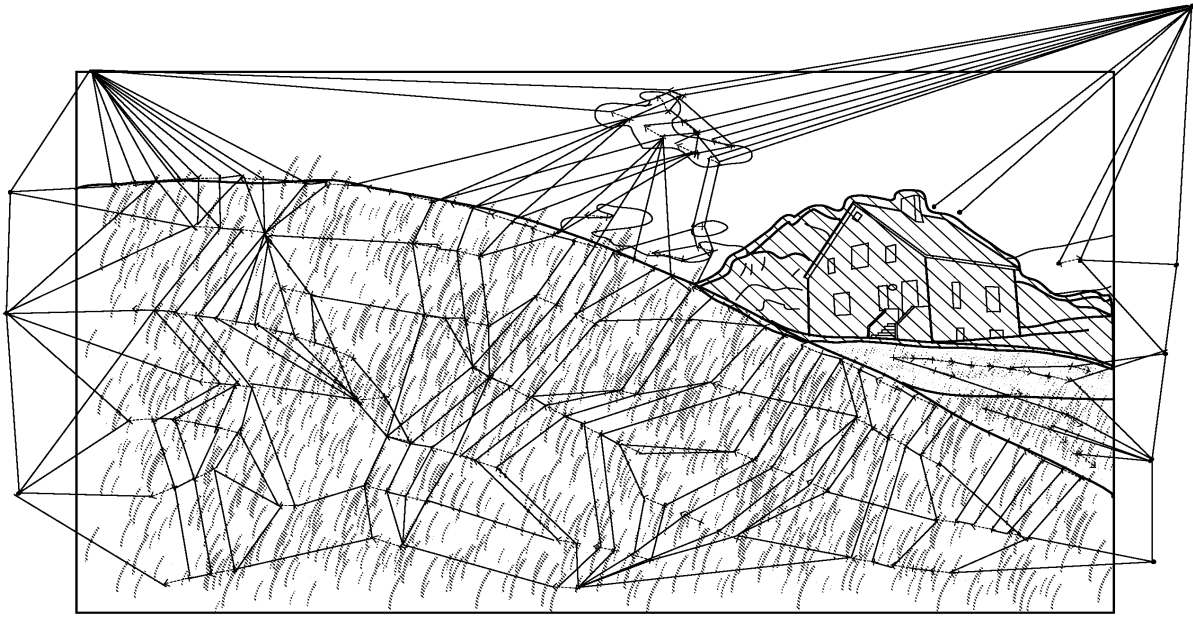


Fig. 4

See '017 Patent, FIG. 4, 6:39-48. “FIG. 4 depicts a starting mesh and ending mesh overlaid on [a] digital image of [a] house and landscape” ('017 Patent, 6:39-44), as well as “various links connecting beginning points and ending points” defined by the user (*compare* '017 Patent, 10:55-11:28, FIG. 3C, *with* '017 Patent, FIG. 4).

44. Below, one such **link** has been annotated in **red** alongside a **green-**annotated **edge** that forms part of one of the two depicted meshes:

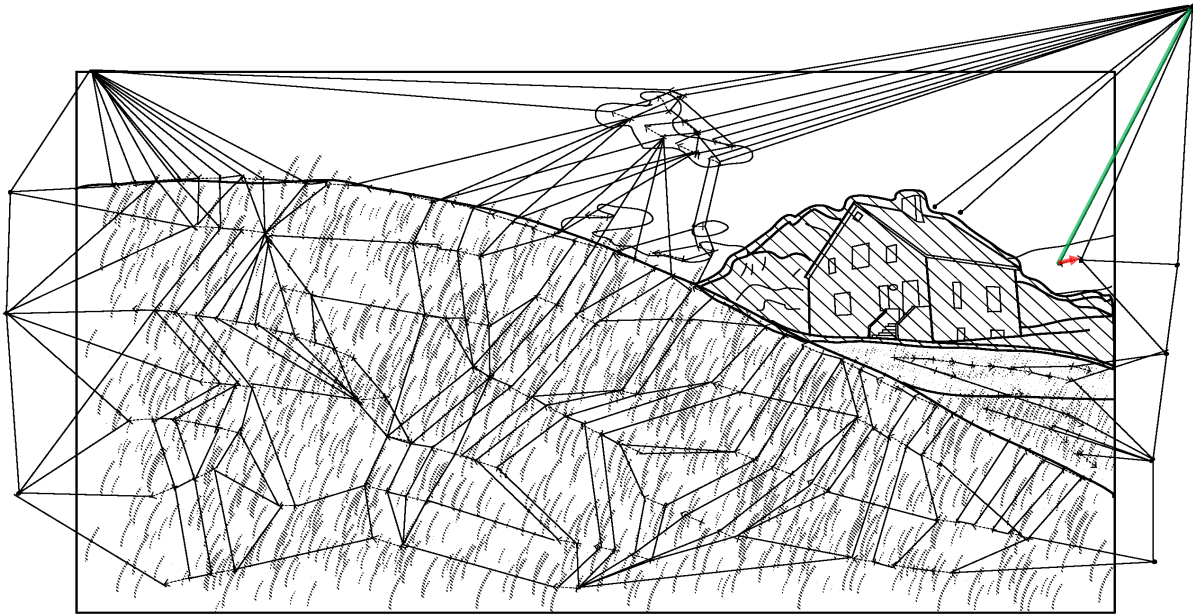


Fig. 4

'017 Patent, FIG. 4 (annotations added). As shown, the **link's** user-defined **starting** point is the lower vertex of the **edge**. The **edge** thus, by association, may be presumed to be the edge of a triangle in the "**starting** mesh"—i.e., is a **starting edge** (see '017 Patent, 6:39-44)—while the **link's** starting point, being both a vertex of the starting mesh and user-defined, is synonymous to a mesh algorithm handle (see Igarashi, 2-3; Wampler, ¶¶[0036]-[0037]).

45. Below, a **blue** annotation of an additional **edge** has been further added:

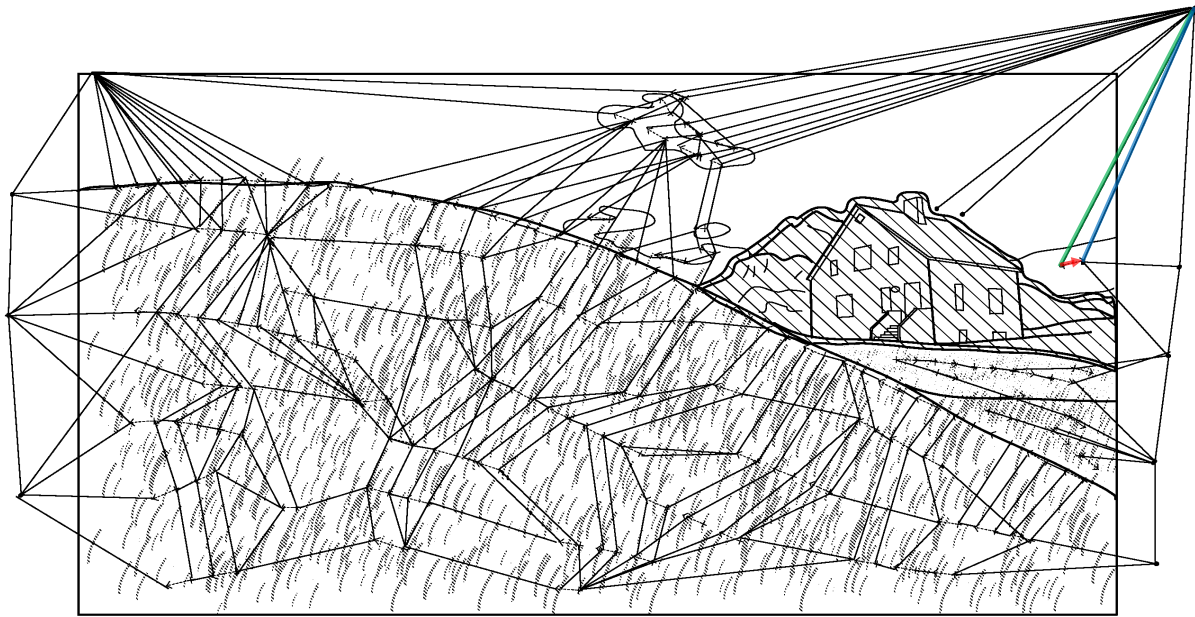


Fig. 4

'017 Patent, FIG. 4 (annotations added). As shown, the **link's ending** point forms the lower vertex of this additional **edge**, which, again by association, is thus an **ending edge** within the “**ending mesh**.” See '017 Patent, 6:39-44. Further, because the **starting edge** and **ending edge** are both coupled to the **link** and share the same upper vertex, the **ending edge** is thus the same edge of the same triangle as the **starting edge**, but located where the **starting edge** ultimately shifts to with the starting point (i.e., handle) along the **link**. See '017 Patent, 6:39-44; Igarashi, 2-3; Wampler, ¶¶[0036]-[0037].

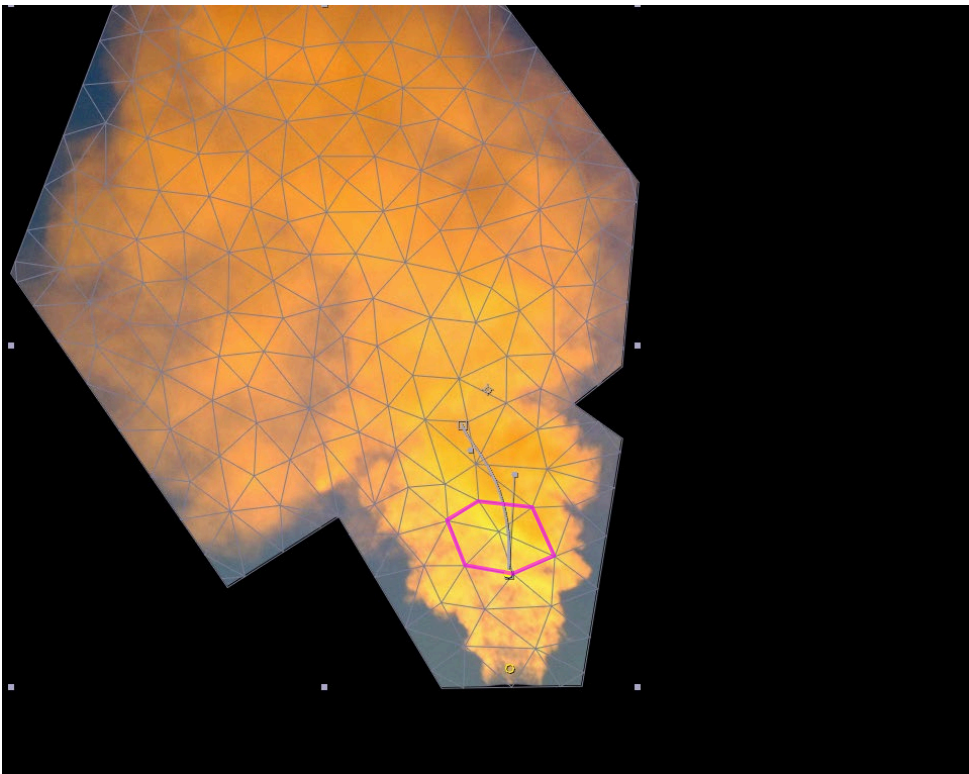
46. As illustrated, the **ending edge** is shorter and at a steeper angle than the **starting edge**. As such, pixels along the **starting edge** (and thereby, as a function of the underlying affine transform, all pixels of the **starting edge's** triangle (see IMU-

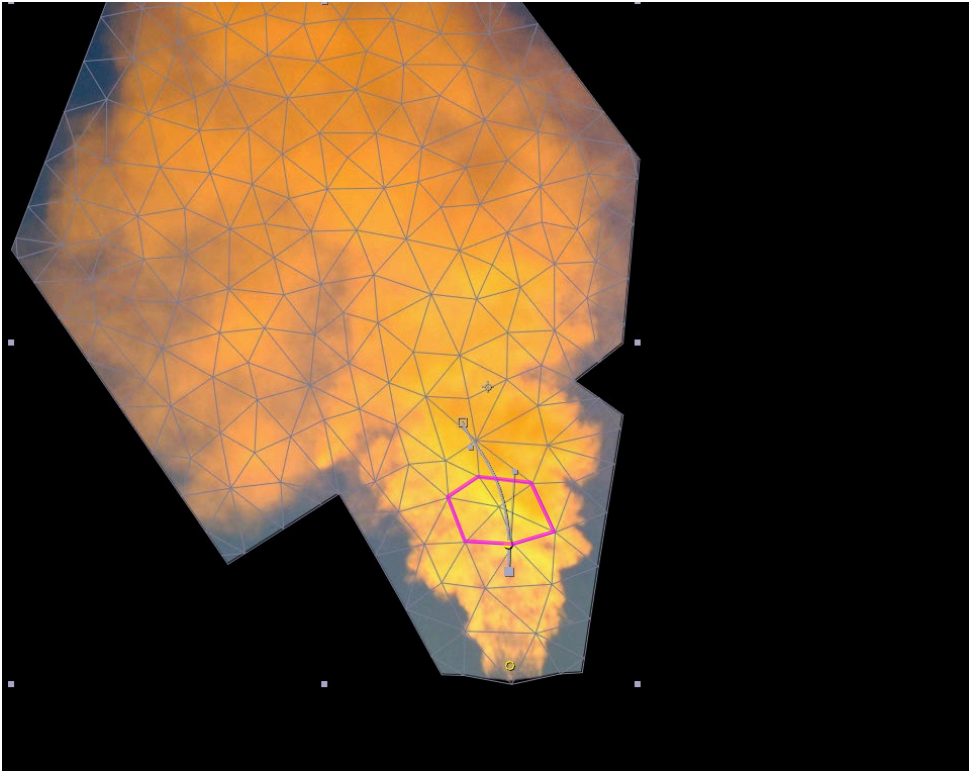
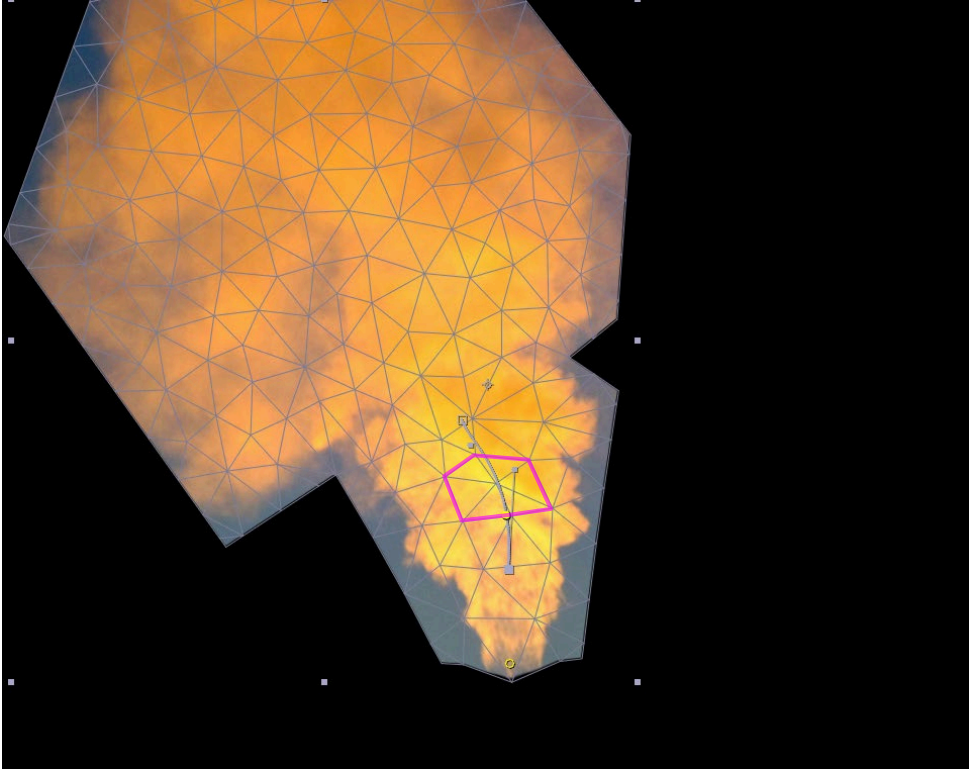
Affine, 11)—which necessarily include, as relevant here, pixels in the **link's** associated set of pixels (*see* '017 Patent, 11:18-28)—are compressed and reoriented when shifted along the **link** to the **ending edge's** location as a result of applying an affine transform to the **starting edge's** associated triangle. *Compare* '017 Patent, FIG. 4, 8:34-37, *with* Igarashi, 2-3, 7, *and* Wampler, FIGS. 5-7, ¶¶[0003]-[0004], [0036]. As such, FIG. 4 illustrates that, even if pixels are compressed, reoriented, or otherwise stretched when shifted by a mesh algorithm along a link, these pixels are still “shifted” as claimed according to the specification and are thus not necessarily excluded from claimed “set of pixels”—i.e., pure translation of the set of pixels is unnecessary.

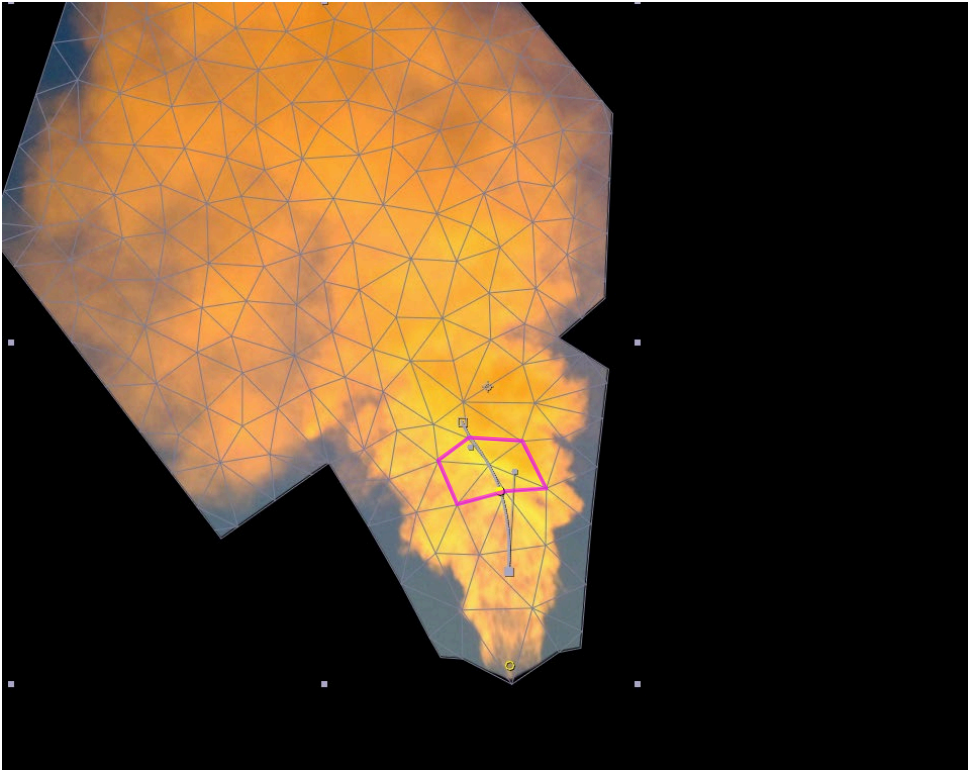
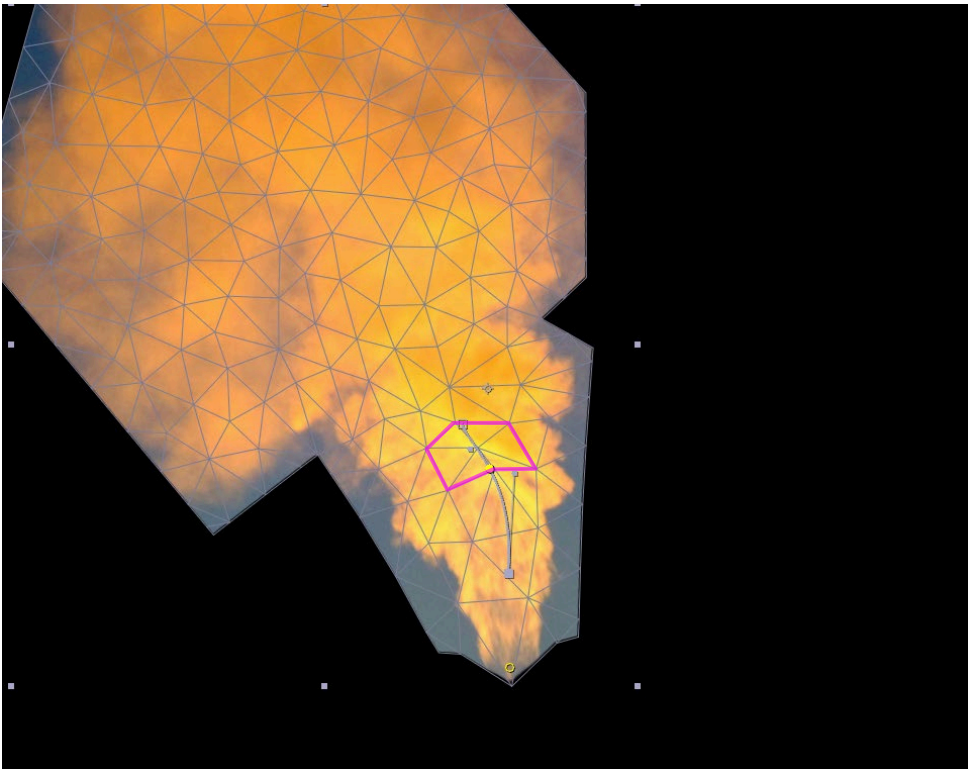
4. AECS6's Puppet Effect Utilizes the Same Type of Triangular Mesh Algorithm as the '017 Patent.

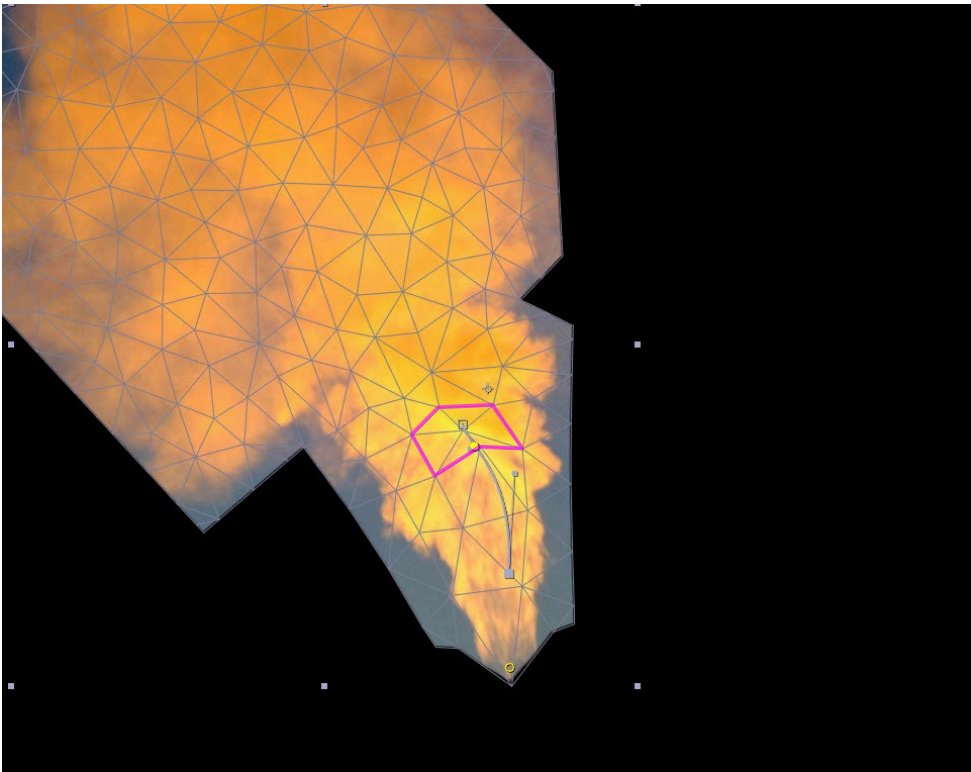
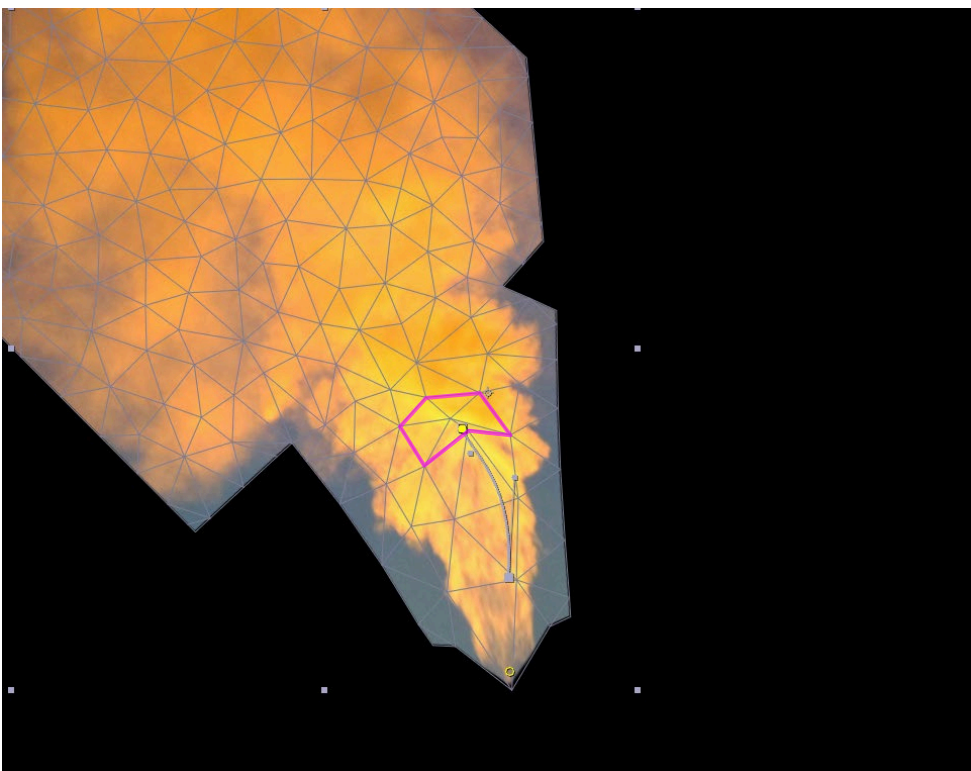
47. Turning next to AECS6's Puppet effect, AEM teaches, as I explained in my previous declaration, that AECS6's Puppet effect similarly uses a mesh algorithm that “divides” a layer's nontransparent pixels into a “mesh of triangles” and causes “the pixels [to] move with the mesh” to “accommodate” a Deform pin's movement along a motion path. AEM, 218-221; Greenspun, ¶¶55, 96-97. I also included in my previous declaration screenshots from AECS6—specifically of an animation that I created therein using the Puppet effect on a smokestack image—that served to aid in visualizing these teachings from AEM. Greenspun, ¶¶81, 107, Attachment B.

48. For the instant declaration, I have recreated the aforementioned Puppet effect animation of the smokestack image using the same steps described in my previous declaration (*see* Greenspun, ¶¶63-81), but using a higher computer screen resolution to further aid in visualizing how pixels move as a result of the Puppet effect's mesh algorithm. I have also included cropped screenshots of this recreated, higher resolution animation as it appears in AECS6's Composition Panel, but now at half-second time intervals across the animation to further aid in such visualization:

Animation Time Point	Composition Frame
(0:00:00:00)	

Animation Time Point	Composition Frame
(0:00:00:12)	 A composition frame showing a large, irregularly shaped orange and yellow mesh structure against a black background. The mesh is composed of many small triangles. A smaller, pink-outlined hexagonal mesh is visible within the larger structure, positioned towards the bottom right. Several small, light blue square markers are scattered around the pink hexagon. A small yellow circle is located at the bottom right of the pink hexagon.
(0:00:01:00)	 A composition frame showing a large, irregularly shaped orange and yellow mesh structure against a black background. The mesh is composed of many small triangles. A smaller, pink-outlined hexagonal mesh is visible within the larger structure, positioned towards the bottom right. Several small, light blue square markers are scattered around the pink hexagon. A small yellow circle is located at the bottom right of the pink hexagon.

Animation Time Point	Composition Frame
(0:00:01:12)	 A composition frame showing a wireframe mesh of a face, likely a mouth, rendered in orange and yellow. The mesh is overlaid on a dark background. A pink polygon is drawn on the lower part of the mesh, and a blue line connects two points on the mesh. The frame is labeled (0:00:01:12) on the left.
(0:00:02:00)	 A composition frame showing a wireframe mesh of a face, likely a mouth, rendered in orange and yellow. The mesh is overlaid on a dark background. A pink polygon is drawn on the lower part of the mesh, and a blue line connects two points on the mesh. The frame is labeled (0:00:02:00) on the left.

Animation Time Point	Composition Frame
(0:00:02:12)	 A composition frame showing a wireframe map of a geographical area. The map is overlaid on a black background. The map area is filled with a dense grid of orange and yellow triangles, suggesting a terrain or elevation map. A pink polygon is drawn on the map, and a yellow line with arrows indicates a path or trajectory. The path starts at a yellow circle at the bottom right and moves towards the pink polygon. There are also small white squares on the map.
(0:00:02:23)	 A composition frame showing a wireframe map of a geographical area, similar to the one above. The map is overlaid on a black background. The map area is filled with a dense grid of orange and yellow triangles. A pink polygon is drawn on the map, and a yellow line with arrows indicates a path or trajectory. The path starts at a yellow circle at the bottom right and moves towards the pink polygon. There are also small white squares on the map.

49. As shown in the screenshots above, pink annotations have been added showing where and how a set of six triangles—the pixels of which, as relevant here, are parallel to the motion path—moves across the animation with the Deform pin. As can be seen, the outline of the triangles deforms as the Deform pin moves along the motion path, with the triangles’ pixels deforming correspondingly—indeed, at (0:00:00:00) the outline of the set of triangles resembles a hexagonal shape, but at (0:00:02:23) the bottom of this hexagonal outline has inverted, indicating that the set of pixels within the outline have been compressed together. And indeed, the POR also appears to recognize that such similarly occurs in the animation provided in my previous declaration and the corresponding screenshots therein. *See* POR, 48 (“Portions of the image are merely stretched or deformed in order to move the [Puppet effect] Deform pin.”); 14-16 & n.1 (“Dr. Greenspun’s shifting of the Deform pin [in AEM] is indistinguishable from the shifting of a single pixel.”).

50. Additionally, as an even further aid in visualizing AEM’s teachings, I exported the above-depicted animated GIF from AECS6 in accordance with AEM’s instructions for doing so (*see* AEM, 574) and uploaded the animated GIF to archive.org for permanent hosting, which can be viewed at the following URL: https://archive.org/details/smoke_202402.

51. Yet, the stretching and compressing of these pixels by a mesh algorithm in the animation depicted in the screenshots and GIF animation above—even if *more* than a pure translation of a set of pixels—is no different than how sets of pixels move in both the typical and the ‘017 Patent specification’s triangular mesh algorithms discussed above, which, again, expressly meets the claimed “shifting” a “set of pixels” according to the specification. *See* ‘017 Patent, 8:28-41, 6:49-63. As such, by teaching how to use such a mesh algorithm to create the above animation (*see* AEM, 218-221; Greenspun, ¶¶55, 96-97), AEM therefore teaches the claimed “shifting” of a “set of pixels.”

C. Okabe’s Animation Technique was not Relied Upon for the Claimed “shifting.”

52. The POR states that Okabe uses a “video patch[ing]” animation technique that does not “shift” pixels. POR, 36-40. The POR also states that, because Okabe already uses this “patch-based” technique, “[o]ne reading Okabe would not look to a pixel-based shifting mechanism such as Shepard’s distortion.” POR, 37.

53. In my previous declaration, however, I did not rely on Okabe’s animation technique, much less did I explain how a POSITA would have modified *Okabe* after viewing IMU—in fact, I explained the opposite: that a POSITA would have modified *IMU* in view of Okabe. *E.g.*, Greenspun, ¶165.

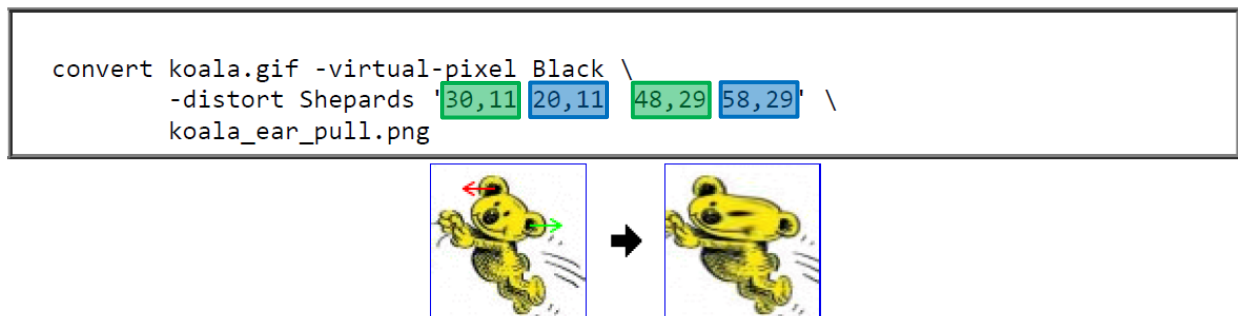
54. Further, my previous declaration never relied on or even mentioned Okabe’s alleged “video patch[]”-based animation because, in my opinion, such was unnecessary for explaining a POSITA’s motivation for combining IMU and Okabe. In fact, in my previous declaration, I never argued that Okabe teaches the claimed “shifting” of pixels at all. Rather, I relied solely on IMU’s teachings regarding IMV6’s “Shepard’s Distortion” for the claimed “shifting.” *E.g.*, Greenspun, ¶182 (“[T]he IMU-Okabe Combination... shift[s] the selected pixels in the stroke’s direction ***using IMV6’s Shepard’s Distortion.***”). At most, my discussions of Okabe with respect to the claimed “shifting” were solely that, when conducting such ***Shepard’s-distortion-driven*** “shifting” in the IMU-Okabe Combination—i.e., the movement of ***IMV6 Shepard’s Distortion control points*** to specific positions over time (*see* Section IV.A)—the specifying of positions of IMV6 Shepard’s Distortion control points would have been performed visually “by drawing a corresponding ‘stroke’ from Okabe,” rather than manually using the written commands from IMU. Greenspun, ¶182.

V. THE IMU-OKABE COMBINATION SHIFTS A SET OF PIXELS “ALONG [A] NON-LINEAR PATHWAY.”

55. The Board “question[ed] whether the [IMU-Okabe] combination teaches the claimed ‘automatically shifting the first set of pixels along [a] non-linear pathway.’” Paper 9, 26-27; *see also* EX1050, 23-27. To alleviate any such questions, below I clarify my statements from my previous declaration.

A. IMU Discloses the Claim Language by Teaching Incrementally Moving Shepard's Distortion Control Points Along a Curved Path.

56. As I explained in my previous declaration, IMU teaches using IMV6's Shepard's Distortion on a koala image to generate a new, correspondingly distorted image:



IMU-Distorting, 59 (annotations added); Greenspun, ¶¶150, 172. Specifically, as explained, IMU teaches how to place Shepard's Distortion *control points* on specific *starting coordinates* in an image—e.g., one at (30,11) and another at (48,29) above—and how to then move the control points to *new coordinates*—e.g., (20,11) and (58,29) above, respectively—to generate a correspondingly distorted image. IMU-Distorting, 59; Greenspun, ¶¶150, 172.

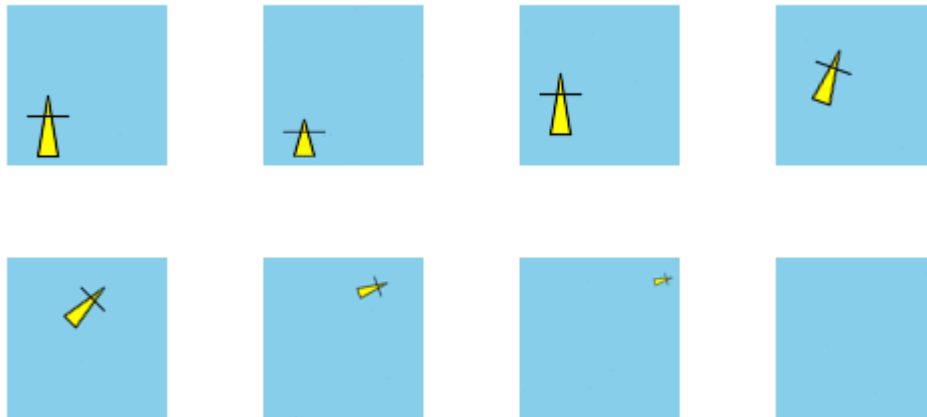
57. I further explained that, in view of IMU's other teachings, IMU's koala example would have been understood by a POSITA as being a *single* increment within a potentially *multi*-increment Shepard's Distortion animation. See Greenspun, ¶172 (explaining that, "[t]o animate with distortion effects," Shepard's Distortion control point movement "is done incrementally for each successive

frame,” where IMU’s koala example “provides *an example of one such increment...*”).

58. Specifically, regarding such a *multi-increment, control-point-based* animation, I explained that IMU indeed expressly teaches the general concept of performing multi-increment, control-point-based animation in IMV6, albeit through an example command using a different *but nevertheless control-point-based* distortion, “Scale-Rotate-Translate (SRT) Distortion”:

```
convert -size 80x80 xc:skyblue -fill yellow -stroke black \
  -draw 'path "M 15,75 20,45 25,75 Z M 10,55 30,55" ' \
  spaceship.gif
convert spaceship.gif \
  \( -clone 0 -distort SRT '20,75 1.0,0.6 0' \) \
  \( -clone 0 -distort SRT '20,60 1 0 20,49' \) \
  \( -clone 0 -distort SRT '20,60 0.9 20 27,35' \) \
  \( -clone 0 -distort SRT '20,60 0.8 45 40,23' \) \
  \( -clone 0 -distort SRT '20,60 0.5 70 55,15' \) \
  \( -clone 0 -distort SRT '20,60 0.3 75 72,11' \) \
  \( -clone 0 -distort SRT '20,60 0.1 80 100,8' \) \
  -set delay 50 -loop 0 spaceship_launch.gif
```

IMU-Distorting, 16-18 (annotations added); Greenspun, ¶151 (“IMU further teaches how to incrementally apply IMV6’s effects to an image to produce a sequence of frames.... IMU provides an example of applying IMV6’s ‘SRT Distortion’ effect—*another ‘distortion’ effect that, like Shepard’s Distortion, uses ‘control points’...*”). The command generates a GIF animation, the individual frames of which I provided in my previous declaration and are provided again below:



Greenspun, ¶151. The GIF animation itself can be viewed at the following archive.org URL, which is embedded in IMU-Distorting’s archived webpage (IMU-Distorting, 18): https://web.archive.org/web/20120314080325/http://www.image-magick.org/Usage/distorts/spaceship_launch.gif. As IMU teaches, the above example depicts a space ship being created (frame 1), a “hunker-down” step being performed (frame 2), and a “handle” point being placed on the space ship at (20,60) and incrementally moved six times to coordinates (20,49), (27,35), (40,23), (55,15), (72,11), and (100,8), which together define a speed and direction for the control point (and thus space ship) to move across the animation—i.e., a “curved path through the sky” (frames 3-8).⁷ IMU-Distorting, 17-18; Greenspun, ¶¶151, 160; *see also* IMU-Distorting, 16 (explaining that the green-annotated coordinates represent

⁷ The space ship is also incrementally scaled and rotated. *See* IMU-Distorting, 16-18.

the “center” for the SRT transformations, while the blue-annotated coordinates “translate the ‘center’ to these coordinates”). And as IMU explains, “[t]hese points represent *control points by which the object can then be animated* in simple terms.” IMU-Distorting, 17.

59. As I explained in my previous declaration, the above teaches generally creating a *multi*-increment, *control-point-based* animation, which—when viewed with IMU’s above-discussed teachings regarding a *single* Shepard’s Distortion increment⁸—equally teaches specifically creating a *multi*-increment, *Shepard’s-Distortion-control-point-based* animation, where a Shepard’s Distortion control point is likewise incrementally moved to multiple specific coordinates in a speed and direction defining a curved path. Greenspun, ¶160 (“[A] POSITA would have recognized that, to animate an image in IMV6 using, e.g., Shepard’s Distortion, a user would write commands that place and incrementally move Shepard’s Distortion control points within the image to new coordinates for each successive frame, such

⁸ Thus, I did not say in my previous declaration that IMU’s SRT Distortion example alone “discloses identifying a first set of pixels that lie along the non-linear pathway and shifting the first set of pixels along the non-linear pathway.” See EX1050, 25.

that each control point moves across the animation at the user’s desired—and potentially non-linear—direction and speed.”).

60. Indeed, below I have written an example IMV6 command that creates one such *multi*-increment animation in which *Shepard’s Distortion control points* are moved along a curved path as taught IMU, and in which the above-discussed *single* Shepard’s Distortion increment from IMU is utilized:

```

0  convert koala.gif \
1      \( -clone 0 -distort Shepards (30,11)(20,11)(48,29)(58,29) \) \
2      \( -clone 0 -distort Shepards (30,11)(10,11)(48,29)(68,29) \) \
3      \( -clone 0 -distort Shepards (30,11)(10,21)(48,29)(68,19) \) \
4      -set delay 50 -loop 0 koala_curved.gif

```

As shown, the overall command resembles that of IMU’s multi-increment SRT Distortion example, except that the koala rather than the space ship is now being distorted as shown in line 0, and that, where “SRT” was previously called, “Shepards” is now being called instead as shown in lines 1-3. *Compare* IMU-Distorting, 17-18, *with* IMU-Distorting, 59. Additionally, as shown in line 1, the above-discussed *single* Shepard’s Distortion increment from IMU is used—i.e., Shepard’s Distortion control points are placed at (30,11) and (48,29) on the koala image and moved apart, each by 10 pixels, to (20,11) and (58,29), respectively. *See* IMU-Distorting, 59. And in line 2, an additional 10-pixel increment is performed, where the control points have been made to continue moving apart in their same respective directions, now to (10,11) and (68,29). In line 3, however, an additional 10-pixel increment is again performed but now in perpendicular directions such that

the control points are moved to (10,21) and (68,19), resulting in an overall curved path for both Shepard's Distortion control points.

61. Running this command in IMV6 generates a GIF animation, the frames of which I extracted by applying IMV6's "-coalesce" operator in the same way as described in my previous declaration for extracting the frames of IMU's SRT Distortion example (*see* Greenspun, ¶151 n.13) and are provided below:



I also uploaded the animated GIF to archive.org for permanent hosting, which can be viewed at the following URL: https://archive.org/details/koala_20240217. As shown, the first and second frame of the animation are identical to the images provided in IMU's Shepard's Distortion example and illustrate the koala being pulled apart by its ears. *See* IMU-Distorting, 59. In the third frame, the koala is again pulled apart, but by a further amount. In the fourth frame, however, the koala's ears are moved perpendicularly, causing the koala to instead swirl, thus illustrating that in such an animation—which, as explained above, IMU teaches how to create—Shepard's Distortion control points are moved along a curved path.

62. Finally, given that Shepard's distortion is a "*point driven*" distortion that nonetheless "shift[s]" a "set of pixels" (*see* Section IV.A), the above-described

multi-incremental movement of an IMV6 Shepard's Distortion control *point* along a *curved path* is thus one way to meet—if not the only way to meet (when using control *points*)—the claimed shifting of a set of pixels specifically “along [a] non-linear pathway.” *See* Greenspun, ¶¶160, 195.

B. The IMU-Okabe Combination Repurposes Okabe's Strokes to Allow the Curved Path to be Defined Visually.

63. I further explained in my previous declaration that, in the IMU-Okabe Combination, the “*same result*” is still achieved: an IMV6 Shepard's Distortion control point is moved to multiple, specific coordinates in a speed and direction defining a curved path. Greenspun, ¶166. These coordinates, however, are specified visually using Okabe's strokes, rather than manually using IMV6's commands taught in IMU. Greenspun, ¶166.

64. Okabe does not teach that its strokes move control points—indeed, I never stated in my previous declaration that Okabe teaches “shifting” pixels at all. *See* Section 1. But, as I explained there, Okabe nonetheless teaches that its strokes are a visual, less-burdensome way of defining an animation's direction and speed. Greenspun, ¶¶164-65; Okabe, 1-2. And in view of these known advantages, as well as the known disadvantages of IMV6's lack of a GUI, a POSITA would have thus been motivated to use Okabe's stroke, not IMV6's written commands, to visually specify a IMV6 Shepard's Distortion control point's speed and direction. Greenspun, ¶¶157-67.

65. Although such amounts to a repurposing of Okabe's stroke, such would not have deterred a POSITA nor diminished any expectation of success. To the contrary, as I explained in my previous declaration, the mere fact that AEM exists demonstrates that a POSITA was both motivated and successful in making such a combination, as AEM teaches specifying the speed and direction of an AECS6 Deform *pin* (like an IMV6 Shepard's Distortion control *point*), but visually using a *motion path* (like Okabe's *stroke*). AEM, 218-19, 187-89, 193-95; Greenspun, ¶163. And even if other implementations existed, AEM demonstrates that a POSITA would have at least found suitable implementing Okabe's strokes in the IMU-Okabe Combination to function like AECS6's motion paths.

VI. STROKES IN THE IMU-OKABE COMBINATION WOULD HAVE BEEN USER-VIEWABLE.

66. In my previous declaration, I demonstrated that Okabe's strokes in the IMU-Okabe Combination would have been made user-viewable, not only by citing the express illustrations of strokes in Okabe (Greenspun, ¶166 (citing Okabe, 7, 2-3)), but also quoting a relevant teaching from AEM: that "[m]otion paths are simply an alternative *visual, spatial* way of *viewing* and working with spatial properties and their keyframes..." (Greenspun, ¶166 n.14 (quoting AEM, 187)).

67. Given that a POSITA would have implemented Okabe's strokes in the IMU-Okabe Combination to function like AECS6's motion paths (*see* Section V.B), Okabe's strokes would have thus been implemented similarly as a "visual, spatial

way of viewing” a Shepards Distortion control point’s pathway (Greenspun, ¶166 n.14 ; AEM, 187). And indeed, as I explained in my previous declaration, implementing such a “visual, spatial way of viewing” the control point’s pathway both aligns with and furthers the exact purpose of the IMU-Okabe Combination itself: to allow visualization and simplification of the animation process. *Compare* Greenspun, ¶165 (“[A] POSITA would have been motivated to modify IMV6 to include Okabe’s simple and graphical user commands ***to make the animation process easier, graphical, and less time consuming.***”), *with* Greenspun, ¶166 n.14 (“[A] POSITA would have nevertheless been motivated to make Okabe’s strokes user-viewable to aid users in visualizing the direction and speed of their animations, ***thereby making the animation process simpler, graphical, and more intuitive.***” (citing AEM, 187)).

VII. THE IMU-OKABE COMBINATION’S MATTE PERFORMS “MASK[ING]” OF PIXELS.

68. In my previous declaration, I explained that, in the IMU-Okabe Combination, a “matte” specifies—like in Okabe—“a region of interest in the image the user desires to animate.” Greenspun, ¶¶188 (citing Okabe, 1), 167 (same); Okabe, 1. By confining animation to a specified region accordingly, this matte as explained thus also specifies a reciprocal region—i.e., a “mask[ed]” region as claimed—in which animation does not occur (and is thus prevented from occurring). Greenspun, ¶188 (“Inherently, the matte also specifies a region that includes pixels

the user does not desire to animate.”). Indeed, the POR admits as much. *See* POR, 38 (“[The POR] notes that *movement of pixels* in [Okabe’s] patches *is confined to be within the discrete patches themselves.*”).

VIII. AECS6’S MASK/MATTE PERFORMS “MASK[ING]” OF PIXELS AND “SELECTING”/“IDENTIFYING” OF A “SET OF PIXELS.”

69. In my previous declaration, I explained that AEM teaches applying AECS6’s “mask”/“matte” to make certain pixels of a layer transparent (and thus excluded from the Puppet effect’s mesh) but not others, which teaches not only the claimed “mask[ing]” of the certain pixels but also, per the specification, the claimed “selecting”/“identifying” of the other pixels. *See* Greenspun, ¶¶110-12, 99-103. And in doing so, I sufficiently explained how these other pixels would, in common instances, meet the claimed “set of pixels.”

A. AEM Teaches Applying AECS6’s Mask/Matte, Which Performs the Claimed “mask[ing].”

70. As I explained in my previous declaration, AEM teaches that AECS6’s Puppet effect “outlin[es],” “mesh[es],” and “move[s]” the “*nontransparent*” pixels of a layer, which is determined by the layer’s “alpha channel.” Greenspun, ¶55 (citing AEM, 219, 220-21). As I also explained, AEM teaches how to modify a layer’s alpha channel—i.e., make certain pixels transparent and others

nontransparent—by applying AECS6’s “mask”/“matte.” Greenspun, ¶¶52 (citing AEM, 315, 318⁹), 100 (same).

71. Thus, at least through syllogism (i.e., “if A, then B; and if B, then C; then if A, then C”), since AEM teaches **applying a mask/matte to modify (A) a layer’s alpha channel (B)** (“if A, then B”), while also teaching that **the layer’s alpha channel (B) determines which pixels of the layer form the Puppet effect’s mesh (C)** (“and if B, then C”), AEM thus teaches **applying a mask/matte to modify (A) which pixels of the layer form the Puppet effect’s mesh (C)** (“then if A, then C”). And, as I explained in my previous declaration, when applying a mask/matte accordingly, the certain pixels made *transparent*—and thus *excluded* from (and prevented from moving with) the mesh—are “mask[ed]” as claimed. Greenspun, ¶¶110-12.

B. AEM’s Mask/Matte Disclosure Likewise Teaches the Claimed “selecting”/“identifying” of a “set of pixels.”

72. In my previous declaration, I also explained that, when “mask[ing]” certain pixels, the opposite act of *not* “mask[ing]” other pixels qualifies, per the specification, as a way of instead “selecting”/“identifying” those other pixels as claimed. ’017 Patent, 6:49-55 (“[A] first of pixels may comprise a group of pixels

⁹ While AEM’s page 315 discusses “[c]ompositing,” it also discusses the separate concept of “*transparency*.” Additionally, AEM’s page 318 does not discuss compositing but instead only “[a]lpha channels, masks, and mattes.”

that intersect with a link between a starting point and an ending point *and that are not covered by a mask.*”), *quoted in* Greenspun, ¶99. And indeed, I explained in my previous declaration that, when applying AECS6’s mask/matte to make certain pixels *transparent*, the other pixels that are oppositely made *nontransparent*—and thus included in (and moved with) the mesh—are “select[ed]”/”identif[ied]” as claimed. *Compare* Greenspun, ¶¶110-12, *and* Section VIII.A, *with* Greenspun, ¶¶99-103.

73. I further stated in my previous declaration that, when applying a mask/matte as such to make only certain pixels *transparent*, the other pixels made *nontransparent* would “in common instances” meet the claimed “set of pixels.” Greenspun, ¶102. And as factual support for this statement, I not only explained how a POSITA would have animated an image of “smoke rising from [a] smokestack” based on my own experiences in the art, but I also compared such to Okabe, which *expressly* depicts animating a “waterfall” image *in an equivalent way*. Greenspun, ¶102 & n.9 (comparing my animation of a smokestack image to Okabe, 7, 2). As I explained, both instances show applying a mask/matte that excludes certain non-smoke/non-waterfall pixels from animation, but ensures that the animation still includes—*all the while drawing motion paths/strokes through*—the other smoke/waterfall pixels, exemplifying what a POSITA would have recognized

IPR2023-00153

U.S. Patent No. 10,346,017

to be “common instances” where “select[ed]”/“identif[ied]” pixels meet the claimed
“set of pixels.” Greenspun, ¶102 & n.9.

IX. CONCLUSION

74. All statements made herein of my own knowledge are true, all statements made herein on information and belief are believed to be true, and these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001.

Dated: Feb. 16, 2024



Philip Greenspun, Ph.D.