

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

PINTEREST, INC.,

Petitioner,

v.

OPENTV, INC.

Patent Owner.

U.S. Patent No. 11,825,139

DECLARATION OF IMMANUEL FREEDMAN, PH.D., IN SUPPORT
OF PETITION FOR *INTER PARTES* REVIEW

TABLE OF CONTENTS

I.	Introduction.....	1
A.	Qualifications	1
B.	Preparation for this Declaration	4
II.	Legal Understanding.....	6
A.	Claim Construction	6
B.	Anticipation	6
C.	Obviousness.....	7
III.	The '139 patent	8
A.	Priority Date	8
B.	Specification.....	8
C.	Prosecution History	11
D.	Level of Ordinary Skill in the Art.....	12
E.	Claim Construction	13
IV.	Grounds.....	14
V.	Summary of Prior Art	14
A.	Orr.....	14
B.	Wu	15
C.	Hsu.....	16
D.	Lim	16
E.	Hervey	17
VI.	Ground 1: Orr Renders Obvious Claims 1-4, 8-9, 11-14, and 20	18
A.	Limitation-By-Limitation Analysis.....	23
1.	Claim 1	23
2.	Claim 2	40
3.	Claim 3: The method of claim 2, further comprising preserving an entirety of the playback pipeline in response to the audio decoder and the video decoder in	

	the playback pipeline being the types indicated by the codec information.	43
4.	Claim 4.....	44
5.	Claim 8.....	44
6.	Claim 9: The method of claim 1, wherein content types include HTTP Live Streaming (HLS) format and Dynamic Adaptive Streaming over HTTP (DASH) format.	47
7.	Claim 10.....	48
8.	Claim 11.....	49
1.	Claim 12.....	49
2.	Claim 14.....	51
3.	Claim 15.....	54
4.	Claim 16.....	54
5.	Claim 17.....	54
6.	Claim 18.....	55
7.	Claim 19.....	55
VII.	Ground 2: Orr in View of Wu and Hsu Renders Obvious Claims 6-7 and 13.....	57
A.	Motivation to Combine	57
B.	Limitation-By-Limitation Analysis.....	59
1.	Claim 6: The method of any one of claim 1, the transitioning from playback of the first piece of content to the new piece of content comprises initiating playback of the new piece of content using a bitrate used for the playback of the first piece of content; and.....	59
2.	Claim 7.....	61
3.	Claim 13.....	62
VIII.	Ground 3: Orr in View of Wu and Lim Renders Obvious Claim 5	64
A.	Motivation to Combine	64
B.	Limitation-By-Limitation Analysis.....	67
1.	Claim 5.....	67

IX.	Ground 4: Hervey Renders Obvious Claims 1-4, 8-9, 11-14, 18, and 20	69
A.	Limitation-By-Limitation Analysis.....	69
1.	Claim 1	69
2.	Claim 2	81
3.	Claim 3	83
4.	Claim 4.....	84
5.	Claim 8	86
6.	Claim 9	90
7.	Claim 10	91
8.	Claim 11	93
9.	Claim 12	93
10.	Claim 14	93
11.	Claim 15	98
12.	Claim 16	98
13.	Claim 17	99
14.	Claim 18	99
15.	Claim 19	99
X.	Ground 5: Hervey in View of Hsu Renders Obvious Claims 6-7 and 13	101
A.	Motivation to Combine	101
B.	Limitation-By-Limitation Analysis.....	104
1.	Claim 6: The method of any one of claim 1, wherein the transitioning to the new piece of content comprises initiating playback of the new piece of content using a bitrate used for playback of the first piece of content to maintain the bitrate for playback of the new piece of content.....	104
2.	Claim 7	106
3.	Claim 13	108
XI.	Ground 3: Hervey in View of Lim Renders Obvious Claims 5	110

IPR of U.S. Patent No. 11,825,139

Decl. of Dr. Immanuel Freedman

A.	Motivation to Combine	110
B.	Limitation-By-Limitation Analysis.....	113
1.	Claim 5	113
XII.	Conclusion	115

LIST OF EXHIBITS

Exhibit	Description
EX1001	U.S. Patent 11,825,139 (“139 patent”)
EX1002	Declaration of Immanuel Freedman, Ph.D.
EX1003	Declaration of Edward Hervey
EX1004	File History of U.S. Patent 11,825,139
EX1005	U.S. Patent Application Publication 2009/0132725 (“Orr”)
EX1006	Chinese Patent Application No. CN 106060628 A and certified translation
EX1007	U.S. Patent 8,516,144 (“Hsu”)
EX1008	U.S. Patent 9,628,543 (“Lim”)
EX1009	“ <i>decodebin3, Modern playback use-cases</i> ” presentation by Edward Hervey at the GStreamer Conference 2015, October 8 th , 2015 (“Hervey”) available at https://web.archive.org/web/20170226222859/https://gstreamer.freedesktop.org/data/events/gstreamer-conference/2015/Edward%20Hervey%20-%20decodebin3.pdf
EX1010	Declaration of Mina Ching
EX1011	“GStreamer Conference 2015,” available at https://web.archive.org/web/20160106210441/http://gstreamer.freedesktop.org/data/events/gstreamer-conference/2015/
EX1012	“Index of data/events/gstreamer-conference/2015,” available at https://web.archive.org/web/20160109011525/http://gstreamer.freedesktop.org/conference/2015/index.html
EX1013	“GStreamer Conference 2015, Schedule,” available at https://web.archive.org/web/20161018073419/https://gstreamer.freedesktop.org/conference/2015/schedule.html
EX1014	“GStreamer Conference 2015, Talks and Speakers,” https://web.archive.org/web/20160125231357/http://gstreamer.freedesktop.org/conference/2015/talks-and-speakers.html

Exhibit	Description
EX1015	LWN.net article “Upcoming work in and around GStreamer,” https://web.archive.org/web/20160330004434/https://lwn.net/Articles/661403/
EX1016	U.S. Pat. No. 8,289,370
EX1017	U.S. Pat. App. Pub. No. 2015/0104158
EX1018	“GstCaps” available at https://web.archive.org/web/20160111144202/http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/GstCaps.html
EX1019	“GstTypeFind” available at https://web.archive.org/web/20151026053357/http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/gstreamer-GstTypeFind.html
EX1020	Curriculum Vitae of Immanuel Freedman, Ph.D.

IPR of U.S. Patent No. 11,825,139

Decl. of Dr. Immanuel Freedman

I, Dr. Immanuel Freedman, declare as follows:

I. Introduction

1. I have been retained by Quinn Emanuel Urquhart & Sullivan, LLP on behalf of the Petitioner Pinterest, Inc. (“Petitioner”) as an independent expert in this *inter partes* review (this “Proceeding”) before the Patent Trial and Appeal Board of the United States Patent and Trademark Office (the “Board”) to review claims 1-19 (“the challenged claims”) of U.S. Patent No. 11,825,139 (“the ’139 patent”). I have been asked by the Petitioner to assist in evaluating the claims and the disclosure of the ’139 patent.

A. Qualifications

2. Ex. 1020 is a true and correct copy of my current CV, which describes my education, patents and publications, employment and research history, and professional activities and awards.

3. I received a Bachelor of Science and Ph.D. in Physics from the University of Durham in England in 1979 and 1986, respectively. I have over 30 years of industry experience, a substantial portion of which was spent working with video, imaging, modeling, simulation, systems analysis, design, development, and testing. I am a registered patent agent before the USPTO. Between obtaining my undergraduate and doctoral degree, I developed a microcomputer system for

IPR of U.S. Patent No. 11,825,139

Decl. of Dr. Immanuel Freedman

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.

4. 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.

5. I have worked as a chief systems engineer and have extensive experience in the scientific modeling and statistical data compression space. I received National Aeronautics and Space Administration Group Achievement Awards in 1990 and 1992.

6. I have also worked as a software engineer in contract and in senior positions, notably at the Federal National Mortgage Association in Washington, D.C., General Instrument Inc., Comstream Inc., and Armor Safe Technologies Inc.

7. From 1996-1997, 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. I then went on to work as an engineering contractor or consultant for various companies, working primarily on image processing systems and digital interactive television set-top boxes.

8. From 2000 to 2002, I worked as the chief systems engineer for Media Logic Systems Ltd. where I developed a live interactive television system for customers to communicate with human sales agents in video enabled call centers. I spent significant time researching and developing 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.

9. 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*.

10. I am also a Senior Member of the Institute of Electrical and Electronics Engineers (“IEEE”) and currently chair the Philadelphia Chapter of the Communications & Information Theory Societies. I am a former Chair of the American Association of Pharmaceutical Scientists Pharmacology and Predictive Modeling Communities. I also served as the 2019 Vice Chair of the IEEE P2673 Intelligence Augmentation for Medical Imaging Standards Working Group, currently lead the Trusted Analytics Sub-Group of the IEEE P2795 Shared Analytics for Secured and Unsecured Networks Standards Working Group, and serve as a member of the IEEE Dynamic Spectrum Access Standards Machine Learning Standards Working Group.

B. Preparation for this Declaration

11. In forming my opinions, I have considered the ’139 patent specification, including the Abstract, the figures, and the claim language itself, as would have been understood by a person of ordinary skill in the art as of the priority date of the ’139 patent (a “POSITA”). My understanding of “POSITA” and “priority date” are set forth below. I have also reviewed the file history of the ’139 patent, the Exhibits that are listed in the list of Exhibits, and any other material cited in this declaration.

12. In forming my opinions, I have relied on my personal knowledge and professional experience, and on the documents and information referenced in this declaration.

13. This declaration explains, based on facts and information available to me to date, the subject matter and opinions related to this Proceeding. As such, I am prepared to provide expert testimony regarding opinions formed resulting from my analysis of the issues considered in this declaration if asked about those issues by the Board or by the private parties' attorneys.

14. Additionally, I may discuss my own work, teachings, and knowledge of the state of the art in the relevant time period. I may rely on handbooks, textbooks, technical literature, and the like to demonstrate the state of the art in the relevant period and the evolution of relevant technologies.

15. Throughout this declaration, I refer to specific pages of the '139 patent and other documents. The citations are intended to be exemplary and are not intended to convey that the citations are the only source of evidence to support the propositions for which they are cited.

16. I am being compensated for my time spent on this matter at a rate of \$450 per hour, and my compensation is in no way contingent upon the outcome of this matter or on the opinions I offer. All of the opinions expressed in this declaration are my own.

II. Legal Understanding

17. In this section, I describe my understanding of certain legal standards that I have relied upon in forming my opinions set forth in this declaration. I have been informed of these legal standards by Petitioner's attorneys. I am not an attorney and I have not thoroughly researched the law on patent invalidity. I am relying only on instructions from Petitioner's attorneys for these legal standards.

A. Claim Construction

18. I have been instructed by counsel that claim construction is a matter of law. I understand that in an *inter partes* review, claims are construed using the same claim construction standard that would be used to construe the claim in a civil action, namely according to their plain and ordinary meaning to a POSITA.

19. I understand that a patent may include two types of claims, independent claims and dependent claims. An independent claim stands alone and includes only the limitations it recites. A dependent claim can depend on an independent claim or another dependent claim. I understand that a dependent claim includes all the limitations that it recites in addition to the limitations recited in the claim from which it depends.

B. Anticipation

20. I understand that a patent claim is anticipated when a single piece of prior art describes every element of the claimed invention, either expressly or

inherently, arranged in the same way as in the claim. For inherent anticipation to be found, it is required that the missing descriptive material is necessarily present in the prior art. I understand that, for the purpose of an *inter partes* review, prior art that anticipates a claim can include both patents and printed publications from anywhere in the world.

C. Obviousness

21. I understand that a patent claim is unpatentable and invalid if the subject matter of the claim as a whole would have been obvious to a POSITA as of the time of the invention at issue. My understanding of a POSITA is set forth below. I understand that the following factors must be evaluated to determine whether the claimed subject matter is obvious: (1) the scope and content of the prior art; (2) the difference or differences, if any, between each claim of the patent and the prior art; and (3) the level of ordinary skill in the art at the time the patent was filed. Unlike anticipation, which allows consideration of only one item of prior art, I understand that obviousness may be shown by considering more than one item of prior art. Moreover, I have been informed and I understand that the so-called objective indicia of non-obviousness, also known as “secondary considerations,” are also to be considered when assessing obviousness. These include: (1) commercial success; (2) long-felt but unresolved needs; (3) copying of the invention by others in the field; (4) initial expressions of disbelief by experts in the field; (5) failure of others to solve

the problem that the inventor solved; and (6) unexpected results. I also understand that evidence of objective indicia of non-obviousness must be commensurate in scope with the claimed subject matter.

22. At this time, I am not aware of any evidence of secondary considerations of non-obviousness, have not seen anything identified by the Patent Owner, and do not think anything would overcome the strong showing of obviousness, but I reserve the right to respond if Patent Owner presents evidence or argument of secondary considerations of non-obviousness.

III. The '139 patent

23. The '139 patent is titled "Bitrate and pipeline preservation for content presentation."

24. The '139 patent was filed as U.S. Patent Application No. 18/480,918 on September 4, 2023, and issued on February 18, 2025.

A. Priority Date

25. The '139 patent claims priority to Indian application 201741031917, filed on September 8, 2017. For purposes of this declaration, I have been asked to assume that the '139 patent is entitled to this priority date.

B. Specification

26. The '139 patent is directed to "[s]ystems and methods for optimizing a content change process" in which a "digital receiver receives selection of a new piece

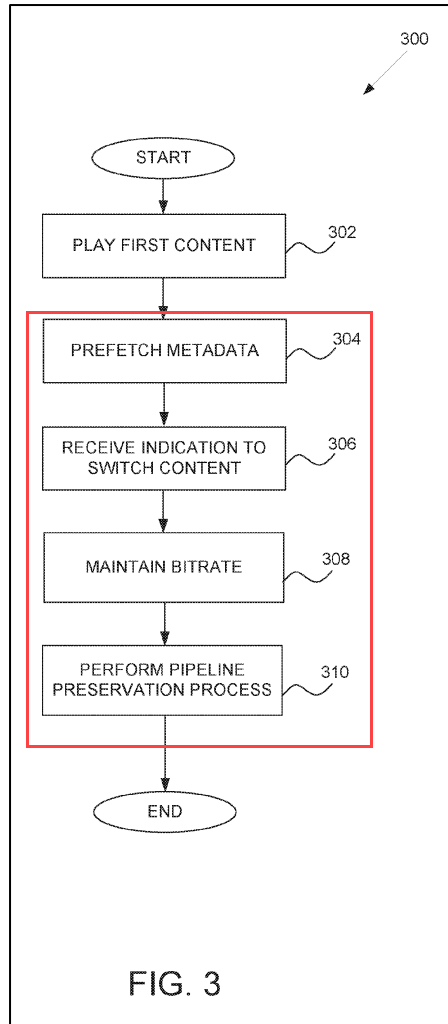
of content for playback,” and “maintains a bitrate used for playback of the first piece of content to initiate playback of the new piece of content.” EX1001, Abstract.

27. The patent identifies two conventional problems that arise at the moment of a content switch. First, with respect to bitrate, “every time a user switches from one content to the next, the playback will start with the lowest bitrate and slowly increase or improve according the capability of the network,” resulting in initially grainy video. (EX1001, 2:37-40.) Second, with respect to pipeline management, a content switch “typically causes a previous content delivery pipeline to be deconstructed (e.g., resources previously acquired are released),” requiring construction of an entirely new pipeline before playback of the new content can begin. (EX1001, 2:55-58.)

28. The alleged solution addresses the bitrate problem “by acquiring appropriate resources for obtaining and processing content data in order to playback selected content.” (EX1001, 5:13-15.) A pipeline 204 for playing content includes a source element 212 that “manages bitrate adaptation at the digital receiver 102” and “remembers a last bitrate that is used for playback of a current/previous piece of content, and continues to use that bitrate for playback of the new content.” (EX1001, 5:22-25.) If the new content cannot be delivered at that bitrate, the source element will lower it accordingly, but the system avoids the conventional penalty of always restarting from the lowest possible bitrate.

29. The pipeline preservation problem is addressed through a two-step analysis using prefetched metadata. The player first checks whether the new content is the same type as the current content (e.g., both HLS or both DASH), and if so, examines codec information to determine whether the same decoders can be reused. When both conditions are satisfied, “the entire pipeline 204 is preserved” and “the resources of the pipeline 204 are maintained and continued to be used to playback the new content.” (EX1001, 8:17–20.)

30. When the content types match but different decoders are required, a partial preservation approach applies: the decoders are released while “a remaining portion of the pipeline 204 (e.g., the source element 212 and the demux 214) are preserved or maintained,” and a new set of compatible decoders is then acquired and added to the preserved portion of the pipeline. (EX1001, 8:27-33.) If the content types do not match at all, all pipeline resources are released and a new pipeline is constructed from scratch. This process is shown, for example, in Fig. 3:



C. Prosecution History

32. The '139 patent was filed on November 9, 2021 as application 17/454,169.

33. The original claims of the '139 patent only required preservation of a portion of the pipeline (*i.e.* source element, demultiplexer, audio decoder, or video decoder) if the content type of the new content was the same as the first content. (EX1004, Nov. 9, 2021 Claims.)

34. On December 22, 2022, an office action issued rejecting pending claims 19-35 on the ground of nonstatutory double patenting as being unpatentable over claims 2-5, 7-10, and 12 of U.S. Patent No. 11,184,652. (EX1004, pages 4-5). The examiner also rejected the claims as anticipated by Shintani (U.S. Pat. Pub. No. 2017/0244924). (EX1004, Dec. 22, 2022 Non-Final Rejection.)

35. In response, the applicant amended the claims to add that when the audio and video decoder are not preserved, the source element and demultiplexer are nevertheless preserved. (EX1004, Mar. 22, 2023 Amendment and Remarks.) The applicant argued that Shintani does not disclose preserving a source element and demultiplexer when the audio and video decoders are replaced. (*Id.*)

36. Following the amendment, the claims were allowed. (EX1004, July 5, 2023 Notice of Allowance.)

D. Level of Ordinary Skill in the Art

37. I understand that a person of ordinary skill in the art (“POSITA”) is a hypothetical person who is presumed to be aware of all pertinent art, thinks along conventional wisdom in the art, and is a person of ordinary creativity—not an automaton. In deciding the level of ordinary skill, I understand that the following factors may be considered:

- The levels of education and experience of persons working in the field;
- The types of problems encountered in the field; and

- The sophistication of the technology.

38. I understand that asserted claims must be evaluated from the perspective of a POSITA. I understand that the relevant point in time for determining the qualifications of a person of ordinary skill in the state-of-the-art is the time of the alleged invention, which I assume to be the earliest effective filing date for the patent. Here, I understand that the earliest alleged priority date is October 30, 2013.

39. In my opinion, a person of ordinary skill in the art at the time of the '139 patent ("POSITA") had at least a Bachelor of Science in electrical engineering or a similar field, such as physics, and at least two years of practical experience in the field of video streaming applications. More education can supplement for less practical experience, and vice versa.

40. I meet these criteria now and met them at the time of the alleged invention. I have applied this level of skill in my analysis. My opinions would not change if a slightly higher or lower level of ordinary skill applied.

E. Claim Construction

41. I have been instructed by Petitioner to perform my technical analysis of the disclosures of the prior art by applying the plain and ordinary meaning of all claim terms in view of the specification and prosecution history. In my opinion, none of the limitations of the claims need to be expressly construed.

42. I reserve the right to provide additional opinions concerning claim construction or the application of certain claim constructions to the prior art, as appropriate, and to respond to any particular claim construction-related argument advanced by PO and/or its expert.

43. As described in more detail in the remainder of this declaration, the prior art discloses or renders obvious the challenged claims under any reasonable potential claim interpretation.

IV. Grounds

Ground	Basis	References	Claims
1	§103	Orr in combination with Wu	1-4, 8-9, 11-14, 18-20
2	§103	Orr and Wu in combination with Hsu	6-7, 10, 16-17, 19
3	§103	Orr and Wu in combination with Lim	5, 15
4	§103	Hervey	1-4, 8-9, 11-14, 18-20
5	§103	Hervey in combination with Hsu	6-7, 10, 16-17, 19
6	§103	Hervey in combination with Lim	5, 15

V. Summary of Prior Art

A. Orr

44. Orr is directed to a channel switching system that reduces or eliminates the latency experienced when a user on a device switches between real-time content channels. (EX1005, Abstract; ¶¶0003–05.) Orr identifies the same fundamental

problem addressed by the '139 patent: conventional media players “reinitialize” when a channel change is detected, tearing down the existing rendering pathway and constructing a new one, a process that “may take more than a few seconds” and produces delays, blank screens, and distorted images. (EX1005, ¶[0004].)

45. Orr’s solution is to determine whether the new channel’s content is compatible with the existing rendering pathway and, if so, to feed the new content directly into the existing pathway without reinitialization, thereby achieving a fast channel switch. (EX1005, ¶[0015]; ¶[0021-22].)

B. Wu

46. Wu is directed to seamless decoding of audio and video when the codec of the audio and video change. Wu teaches that “if encoding parameters change (such as the audio format changing from mp3 to aac ...), it is clearly not feasible to decode and play using the original decoder.” (EX1006, ¶[0005].) Wu teaches that its system therefore “obtain[s] the audio and video decoder parameters,” and determines “changes in encoding formats” or whether they “have not changed.” (*Id.*, ¶¶[0016], [0028].) If the format has changed, “the current audio and video decoders” are “[d]isconnect[ed],” and new audio/video decoders are “create[d] ... based on the aforementioned audio and video decoder parameters.” (*Id.*, ¶¶[0079]-[0080].) Wu further teaches that its demultiplexer does not change.

C. Hsu

47. Hsu is directed to an adaptive bitrate streaming media player that tracks historical bitrate data across sessions and uses that history to determine an appropriate startup bitrate for new streaming sessions, rather than defaulting to the lowest available bitrate. (EX1007; Abstract, 5:28-40, 6:1-30.) Hsu expressly identifies the same content-switch quality problem addressed by the '139 patent: conventional systems “default to the lowest bitrate in the dynamic profile set,” resulting in users “playing a lower quality video than what the current bandwidth to that device allows,” with the time to reach optimal quality taking “as long as fifteen (15) seconds.” (EX1007; 5:28-40.)

D. Lim

48. Lim is directed to a content playback system for electronic devices that prefetches content metadata (including type, format, and codec-relevant information) for content the user has not yet selected, based on navigational context during playback of current content. (EX1008; Abstract, claim 1, [0026–36.].)] Samsung's prefetch manager module fetches “preview content clips, cover art, and metadata for all on-deck content” from a content source while the device is playing other content, making that metadata available locally for immediate use upon a user's content selection. (EX1008; claim 1, [0026].)

E. Hervey

49. Hervey is a PowerPoint document that was publicly presented by Edward Hervey at the GStreamer Conference 2015 on October 8th, 2015 in Dublin, Ireland.

50. Hervey is directed to the design of a new GStreamer pipeline element, decodebin3, intended to overcome the limitations of its predecessor, decodebin2, with respect to stream switching and media playback efficiency. (EX1009, Hervey, *e.g.*, slides 8-12.)

51. In my opinion, Exhibit 1009 was publicly available by 2015, or at least by February 2017.

52. An identical copy of Exhibit 1009 can be found on the Internet Archive. I am familiar with the Internet Archive and have reviewed the declaration of Ms. Ching regarding the operation of the Internet Archive. EX1010.

53. The Internet Archive exhibits are printouts of web pages or files that were archived by the Wayback Machine. Each link to the Wayback machine has a date and timestamp which indicates the date and time that the link was archived. In my experience this date and time an accurate reflection of when the Wayback Machine archives that particular page. Being familiar with the operation of the World Wide Web and crawlers like those used by the Wayback Machine, a page could not be archived by the Wayback Machine unless it was available to the public.

Moreover, crawlers generally operate by accessing pages that were linked by other pages, and therefore a page that was archived by the Wayback Machine is typically archived because it was linked from at least one other webpage. This further indicates that the archived webpage was publicly available at the time it was archived.

54. The Internet Archive link of Exhibit 1009 states that it was archived on February 26, 2017. Thus, in my opinion, Hervey was publicly available via the Internet Archive by February 2017.

VI. Ground 1: Orr in View of Wu Renders Obvious Claims 1-4, 9-12, 15-17, and 19¹

A. Motivation to Combine

55. In my opinion, a POSITA would be motivated to modify Orr in view of Wu to incorporate teachings of preserving a demultiplexer when needing to replace the audio and video decoder. A POSITA would have been motivated by the known benefits of reducing latency and increasing processing efficiency when reusing a demultiplexer, as otherwise creating a new demultiplexer requires time and resources. A POSITA would have had a reasonable expectation that this combination would succeed.

¹ Unless noted otherwise, all emphases in quotes and annotations to figures from prior art references are added.

56. In my opinion, the integration of Wu’s teachings into Orr is technically straightforward. First, I explain how Orr’s playback pipeline operates. Orr describes a playback pipeline that includes a “source filter 214” which takes media received from the media manager and “extracts media data for presentation to the transform filter 216.” (EX1005, ¶[0024], Fig. 2.) The transform filter is a “pluggable filter” that uses decoders, like an “H264 decoder” (H.264 is a well-known compression/decompression algorithm for video) and decompresses the media data to be rendered by the renderer 218. (*Id.*) These components align with Wu’s system “for audio and video decoding and playback” where it uses “separation unit 311” which splits the data into separate audio and video data, and “video decoder 302” and “audio decoder 303,” which decompress the individual audio and video data. (EX1006, ¶[0076], Fig. 3.)

57. Orr teaches that its playback pipeline determines if “[c]ontent type of new channel [is] same as current channel,” and if so, then the system will “[i]nput content of new channel into current rendering pathway,” which as discussed above contains both the source filter (*i.e.* demultiplexer) and transform filter (*i.e.* decoders). (EX1005, Fig. 4, blocks 410 and 414.) In other words, Orr teaches checking both the container format and the compression format to decide whether to reuse those elements in the rendering pathway.

58. The proposed combination would involve modifying Orr's playback pipeline to instead first check if the container format is the same to determine whether to reuse the source filter (*i.e.* demultiplexer), and if so, then check if the compression format is the same to determine whether to reuse the transform filter (*i.e.* decoders). These are teachings found in Wu. When receiving new content that has the same container format as old content, but audio and video content that was compressed using different encoding formats, Wu teaches that if the "encoding parameters change (such as the audio format changing from mp3 to aac ...), it is clearly not feasible to decode and play using the original decoder." (EX1006, ¶[0005].) Wu teaches that its system therefore "obtain[s] the audio and video decoder parameters," and determines "changes in encoding formats" or whether they "have not changed." (*Id.*, ¶[0016], [0028].) If the format has changed, "the current audio and video decoders" are "[d]isconnect[ed]," and new audio/video decoders are "create[d] ... based on the aforementioned audio and video decoder parameters." (*Id.*, ¶[0079]-[0080].) However, the demultiplexer remains the same. Thus, the combination would modify Orr in view of Wu to check the encoding format of the audio and video stream, determine if they have changed from the old content, and disconnect and create new audio and video decoders if the format has changed.

59. A POSITA would have been motivated to modify Orr in view of Wu because of the known benefits of reusing a demultiplexer when the container format

is the same. Orr itself recognizes the benefits of reusing elements of the playback pipeline, stating that re-initialization of the playback pipeline “may take more than a few seconds” because the media player will “tear[] down its existing rendering pathway and construct[] a new rendering pathway.” (EX1005, ¶[0004].) As a result, the “user will experience a delay in viewing the selected channel, which may result in an unsatisfactory user experience.” (*Id.*) Orr therefore teaches the well-known concept that reusing components will mitigate delay. In a situation where new content uses the same container format as old content, but different encoding formats, Wu recognizes that it is “not feasible to decode and play using the original decoder.” (EX1006, ¶[0005].) But a POSITA would recognize that the demultiplexer, which is responsible for separating the video and audio content from the container, could be reused if the container format is the same. By reusing the demultiplexer, this reduces the latency for rendering the new content.

60. A POSITA would understand that creating a new demultiplexer requires re-establishing bitstream syncing and renegotiating the downstream media type (for example, RGB32 at a fixed resolution or PCM at a fixed sample rate). This requires more time and resources than simply replacing the decoders, which merely involves instantiating a new codec instance and feeding it the audio or video stream. Thus, reusing the demultiplexer saves time and increases processing efficiency. A POSITA would understand that whereas reinitializing the demultiplexer can cause

noticeable discontinuities in the playback, maintaining the demultiplexer even while reinitializing newer codecs allows for seamless playback.

61. In addition, to the extent Patent Owner argues that Orr does not expressly disclose a demultiplexer, it also would have been obvious to incorporate the functionality of Wu's separator (*i.e.* a demultiplexer) into Orr's source filter. Demultiplexers for extracting digital audio and video signals have been known since the 1950s, and are well-attested in the prior art. (*See, e.g.*, EX1017, ¶52 ("The stream demultiplexer 634 separates the AV bit stream AVS into a video bit stream VBS and an audio bit stream ABS")). Orr discloses that its content signals contain both a video and an audio signal (EX1005, ¶[0018]) as well as the use of software decoders (such as H264), which decode audio or video signals, but not both (EX1005, ¶[0024]). Thus, a POSITA would be motivated to demultiplex the audio and video signal in order that they can be decoded and rendered to the user. Without such demultiplexing, the decoders would be unable to decode the signal. A POSITA would have had a reasonable expectation of success because demultiplexers for digital signals have been in use for decades and well-within the skill of an ordinary artisan.

62. A POSITA would have had a reasonable expectation that the combination would succeed. Orr already describes performing a content type check to determine whether to reuse the rendering pathway—the proposed modification

would simply check whether the container format is the same before determining whether to reuse Orr's source filter (which performs the function of demultiplexing), then check whether the encoding formats are the same before determining whether to reuse Orr's transform filter (which performs the function of the decoders). Since Orr already describes reusing the demultiplexer and the decoders, the proposed modification would simply require a slight modification in the programming logic. This would have been well-within the skill of an ordinary artisan.

63. Orr and Wu are analogous art, because both are directed to rendering content. *See, e.g.*, EX1005, Abstract ("Methods and apparatus for fast channel switching between real time content on a device."); EX1006, Abstract ("This invention discloses a method supporting variable encoding based on DirectShow").

B. Limitation-By-Limitation Analysis

1. Claim 1

a) *1[pre]: A method comprising:*

64. In my opinion, to the extent the preamble is limiting, Orr discloses this limitation.

65. Orr is directed to methods and an apparatus for channel switching at a device that receives and renders real-time content. (EX1005, Title; Abstract.)

b) *1[a]: causing playback of a first piece of content using a playback pipeline;*

66. In my opinion, Orr discloses this limitation.

67. Orr discloses that the device includes several components that are arranged in a series and used to play back content, *i.e.* a ***playback pipeline***. For example, in Figure 2, Orr discloses that content is played back using a “protocol stack 206,” a “multimedia manager 204,” and “rendering logic 210.” (EX1005, ¶¶[0023-28]; FIG. 2.) These components arranged in a series constitute a ***playback pipeline*** and comprise, for example, a source element, demultiplexer, audio decoder, and video decoder.

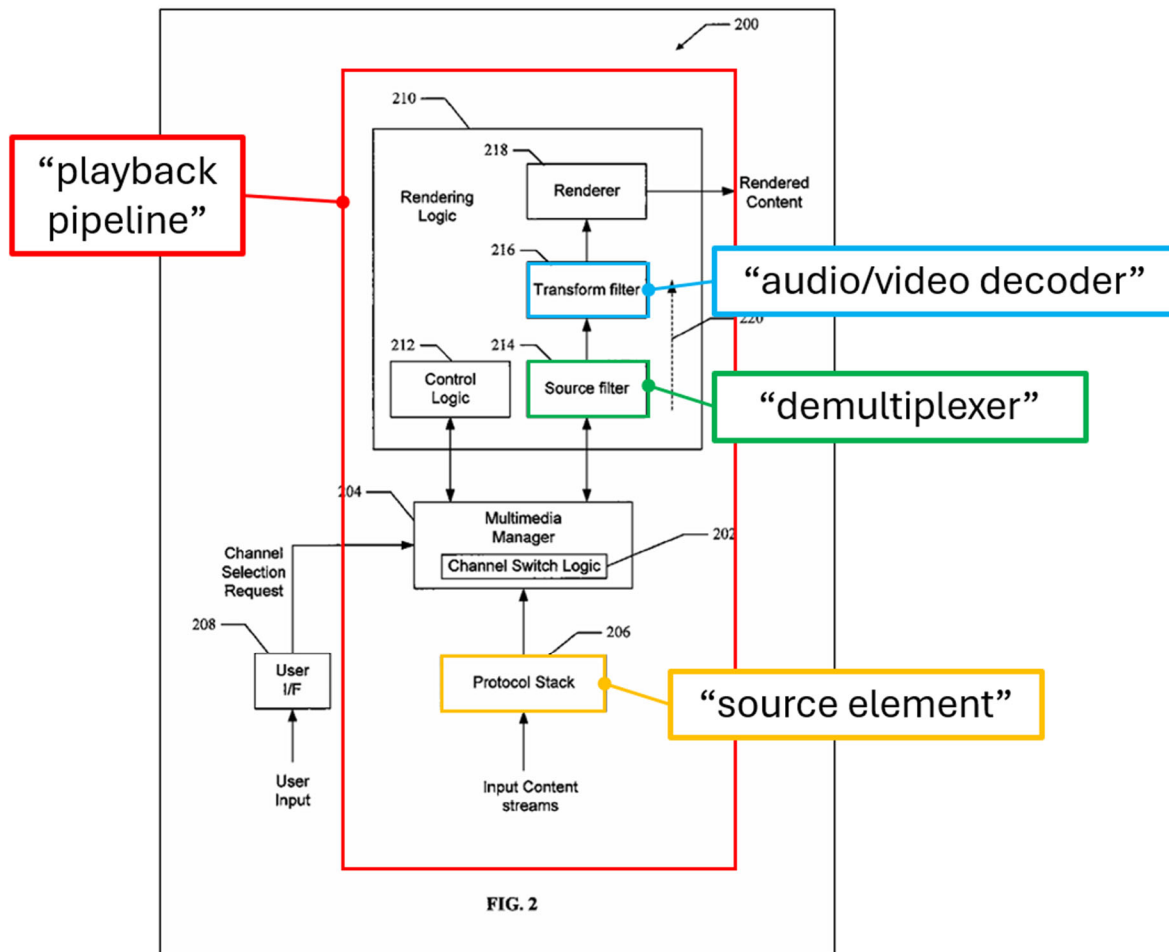


FIG. 2

68. Orr describes that this playback pipeline is used for *playback of a first piece of content*. When a user selects a channel, the “protocol stack 206” of the playback pipeline “receives content streams ... from a network and/or from another communication channel.” (EX1005, ¶¶[0023], [0036] (“[T]he processing logic 302 controls the protocol stack interface 310 to obtain the content for the selected channel.”)) The multimedia manager then “provides control signals to control logic 212 of rendering logic 210,” which “construct[s] a rendering pathway 220 that includes functional elements that operate to process content data associated with a selected content channel.” (EX1005, ¶[0024].) Within the rendering pathway, “[t]he source filter 214 understands the format of the received transport stream and extracts media data for presentation to the transform filter 216,” which “is a pluggable filter, such as an H264 decoder that takes in data from the source filter 214 and decompresses it.” (*Id.*) The decoded media “is then input to a renderer 218 that operates to render the content on the device,” *i.e. playback of a first piece of content*. (*Id.*)

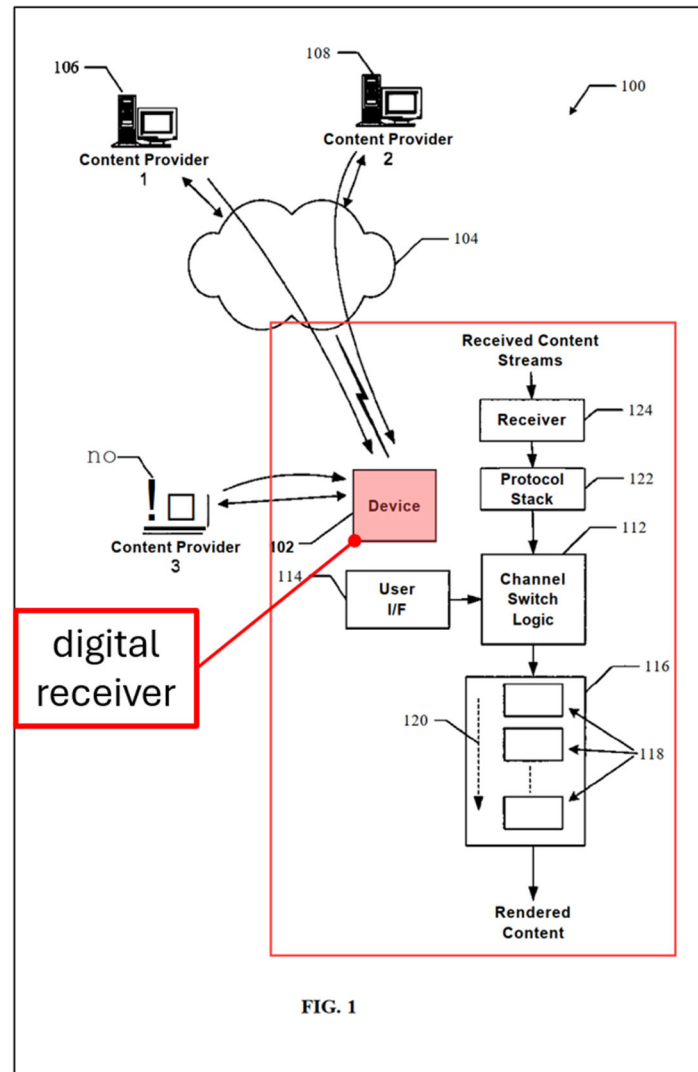
c) *1[b]: receiving, by a digital receiver, a selection of a new piece of content for playback during playback of the first piece of content;*

69. In my opinion, Orr discloses this limitation.

70. With respect to the limitation *digital receiver*, in my opinion, Orr discloses a digital receiver in the form of a device that receives digital media signals.

Orr states that it is directed to methods and an apparatus for channel switching at a device that *receives* and renders real-time content. (EX1005, Title; Abstract.) Since Orr discloses that the content is delivered via a data network 104, the content is sent as a *digital* signal. (EX1005, ¶[0016]; ¶[0017] (“The content providers 106, 108 and 110 operate to provide content to the device 102.”); FIG. 1.) And Orr describes that the “circuits described in [Orr] may be implemented or performed with ... a digital signal processor (DSP),” further indicating that the signals are digital. (*Id.*, ¶[0053].)

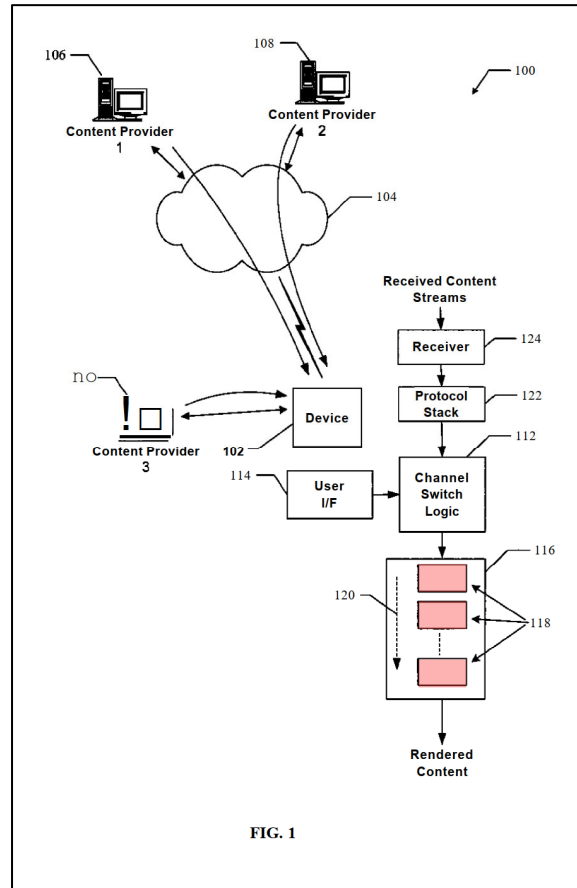
71. As I have shown below in annotated Figure 1, Orr describes its device 102 as including a receiver 124, channel switch logic 112, a user interface 114, a protocol stack 122, and rendering logic 116. (*Id.*, ¶[0018]; FIG. 1.) Orr’s “device 120” is therefore a disclosure of claim 1’s “*digital receiver*.”



72. With respect to the limitation *receiving a selection of a new piece of content*, Orr describes that device 102 (what I have mapped as the claimed “digital receiver”) *receives the selection*. Specifically, Orr describes that “channel switch logic 112 [which is part of device 102] operates to select a particular content channel and inputs data from this content data to the rendering logic 116.” (EX1005, ¶[0019].) Orr also describes that “[t]he channel switch logic 112 then operates to control rendering logic 116 to construct the functional elements 118 needed to

decode and render the content [including] content filters [118], transform filters [118], decoders [118], and/or any other type of functional element needed to process the content flow of the selected channel for rendering at the device 102.” (EX1005, ¶[0019].

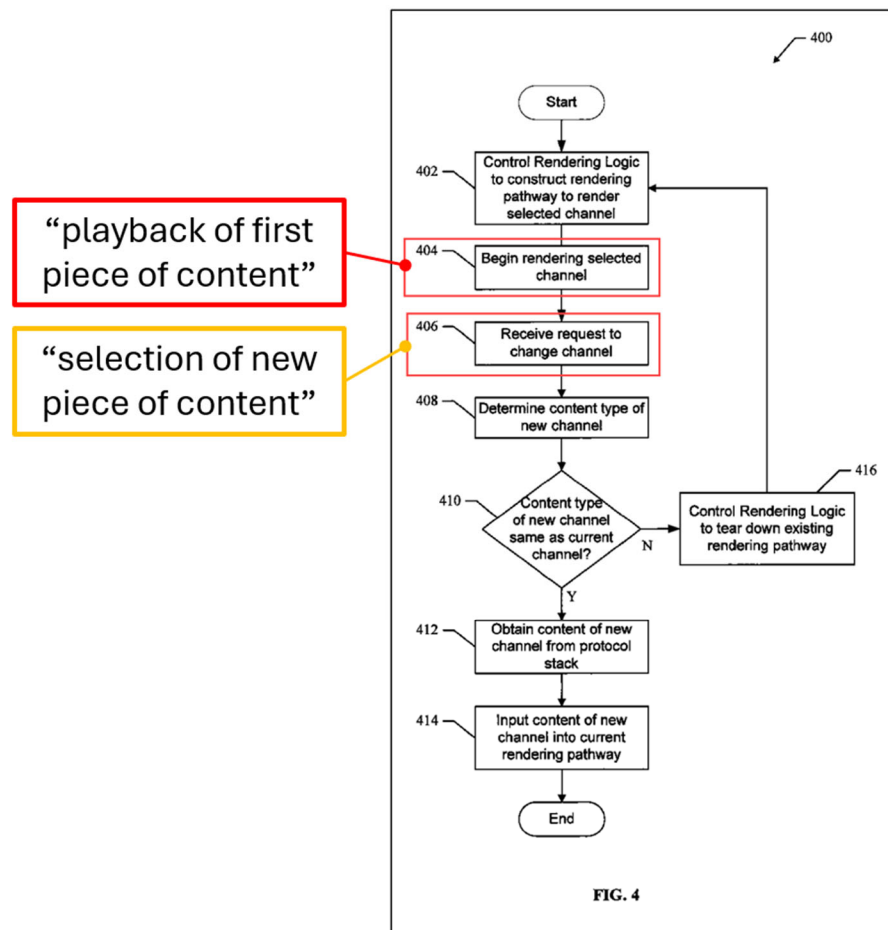
73. Thus, Orr explicitly describes that the device receives a selection of a new channel, which Orr describes contains a new *piece of content*. I have highlighted these elements in the below annotated Figure 1.



74. With respect to the limitation *during playback of the first piece of content*, Orr discloses that “[d]uring operation, the device user may request a channel

change to render content from a *different* channel.” (EX1005, ¶[0020]; ¶[0042] (“[A] channel request is received to render content on a *different* channel.”).) The fact that Orr says there is a “channel change” means that the user was watching a first channel with first content when the selection for a different channel with different content is made. Thereafter, the user interface receives a channel selection indicator from the user and passes it to the channel switch logic during active playback. (EX1005, ¶[0042]; claim 1.)

75. In addition, as I have annotated below, Figure 4 depicts that block 406 (“Receive request to change channel”) appears in the flow after block 404 (“Begin rendering selected channel”) confirming that the selection for a new piece of content is received while the first channel is actively being rendered. (EX1005, FIG. 4.)

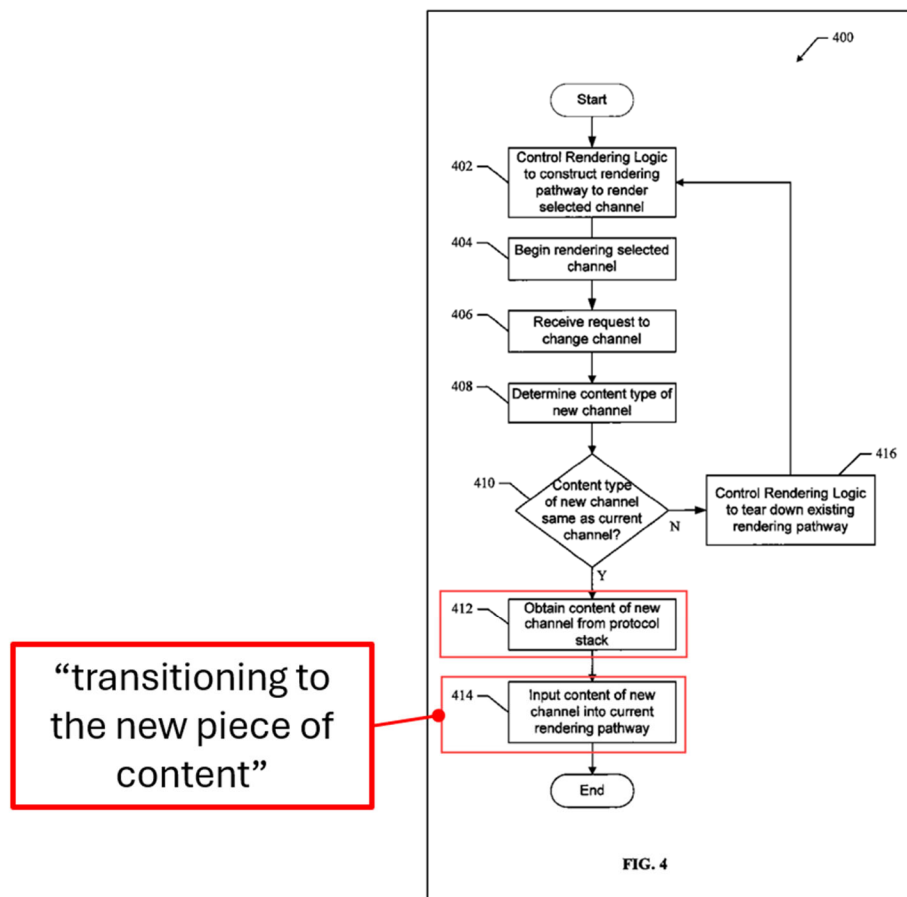


d) *l[c]*: in response to the receiving of the selection, transitioning from playback of the first piece of content to the new piece of content;

76. In my opinion, Orr discloses this limitation.

77. Orr teaches that upon receiving the channel selection indicator, the channel switch logic performs a series of operations that eventually results in the device obtaining the new content from the protocol stack and either inputting it into the existing rendering pathway or constructing a new pathway, both of which constitute the device *transitioning to the new content*. (EX1005, ¶[0022]; ¶[0046-

47]; FIG. 4, blocks 412–414.) The transitioning is in response to the channel selection because the transitioning to new content cannot occur until the new content is selected. Without such a selection, there is no new content to transition to. Orr’s Claim 1 also recites “inputting the second content into the first rendering pathway if it is determined that the second content can be rendered using the first rendering pathway.” As shown in the annotated figure below, FIG. 4, after receiving a “request to change channel” at step 406, Orr’s device performs several steps (as I describe below for limitations 1[c] and 1[d]) that results in “[o]btaining content of new channel” at step 412 and playing the content at step 414. (EX1005, ¶[0046-47].)

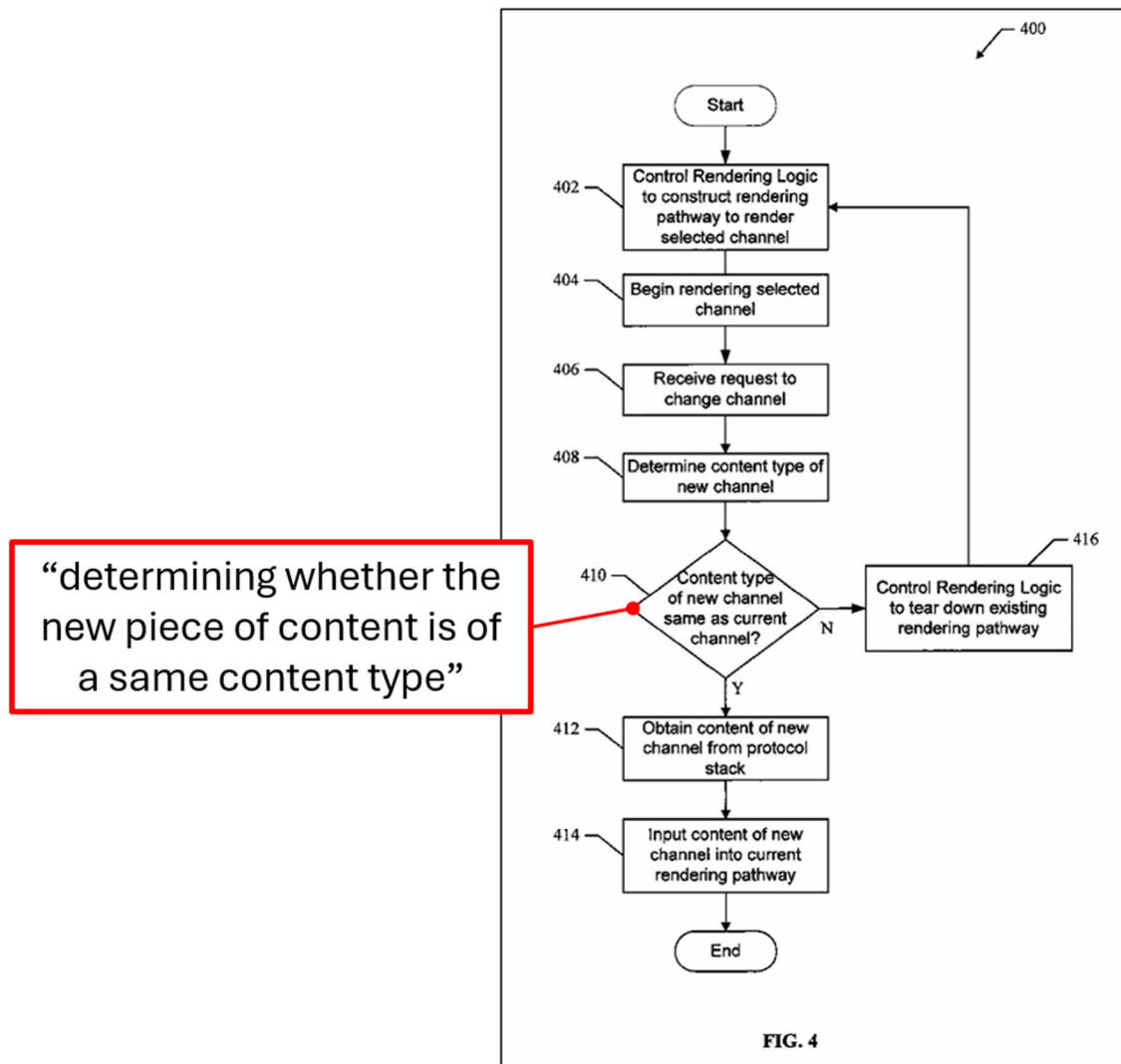


- e) *1[d]: determining whether the new piece of content is of a same content type as the first piece of content; and*

78. In my opinion, Orr discloses this limitation.

79. Orr teaches that the channel switch logic “determines the type of content on the new channel to be rendered” and then assesses “if a current rendering pathway can be used to render the content on the new channel.” (EX1005, ¶[0021]; *see also* ¶[0043]–¶[0044].) Orr describes that the determination of the content-type involves “determin[ing] if the content on the new channel *has the same format* as the content currently being rendered.” (EX1005, ¶[0026].) As I have shown below

in annotations, Figure 4 depicts this determination explicitly at block 410: “Content type of new channel *same as current channel?*” (EX1005, FIG. 4, block 410.)



80. As I discussed above, in the proposed modification in view of Wu, this determination first checks if the new content has the same container format as the old content, *i.e. determine if they have the same content format*. And as I will

discuss in the next limitation, the proposed combination then checks if the new content has the same encoding formats as the old content.

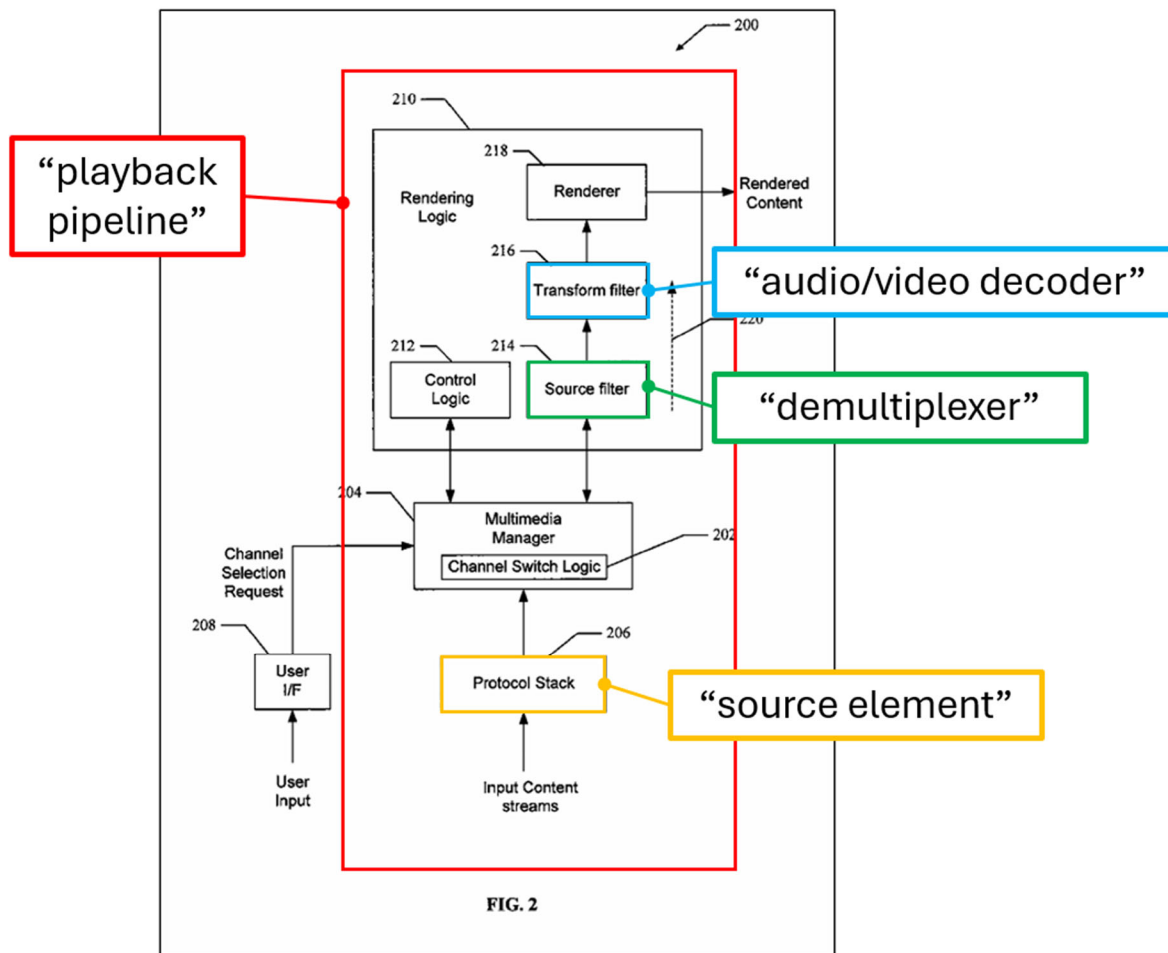
- f) *l[e]: based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder or a video decoder of the playback pipeline used to play back the first piece of content should be preserved; and*
- g) *l[f]: based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the playback pipeline while replacing the audio decoder and the video decoder.*

81. In my opinion, Orr in view of Wu renders obvious this limitation.

82. These limitations describe (1) a “playback pipeline” for the first piece of content that has a source element, a demultiplexer, an audio decoder, and a video decoder, and (2) based on determining the new content has the same content type, determining whether to preserve an audio decoder or a video decoder of the “playback pipeline,” and (3) based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer while replacing the audio and video decoder.

83. With respect to the ***playback pipeline***, Orr discloses that the device includes several components that are arranged in a series to render and play content. Figure 2 depicts these components, which include a “protocol stack 206,” a “multimedia manager 204,” and “rendering logic 210.” (EX1005, ¶¶[0023-28]; FIG.

2.) These components arranged in a series constitute a *playback pipeline* and comprise the claimed source element, demultiplexer, audio decoder, and video decoder. Below, I have annotated Figure 2 to indicate which of these components meet which of the claimed limitations.



84. For the source element, in my opinion, Orr discloses that its protocol stack 206 and protocol stack interface 310 meets this limitation. (EX1005, ¶¶[0025], [0033].) The protocol stack “receive[s] channel content from a variety of sources and operates to make this channel content available to the protocol stack interface

310,” including “content data associated with a selected content channel,” *i.e.* the ***new piece of content***. (EX1005, ¶[0033]; *see also* EX1005, ¶[0023] (“[T]he protocol stack 206 may receive the content streams from a network and/or from another communication channel.”). Thus, in my opinion, it is the ***source element*** because it receives content from content sources.

85. For the demultiplexer, Orr discloses that the playback pipeline includes a demultiplexer, which is a component that separates the video and audio signal. (EX1001, 5:36-38 (“The demux demultiplexes the content data ... to obtain an audio signal and a video signal.”). Orr teaches that content “compris[es] video, audio, captioning, and overlays etc.” (EX1005, ¶[0018].) Orr further discloses ***demultiplexing*** of the audio and video performed by a “source filter 214” that is part of the “rendering logic 210,” *i.e.* a portion of the ***playback pipeline***. (EX1005, ¶[0024].) Specifically, the source filter “extracts media data for presentation to the transform filter 216.” (EX1005, ¶[0024].) Orr explains that the “transform filter 216 is a pluggable filter” and uses, for example, “an H264 decoder that takes in data from the source filter 214 and decompresses it.” (EX1005, ¶[0024].) H.264 is a known ***video*** compression/decompression algorithm—and decoders generally decompress audio or video but not both. (EX1016, 2:44-47 (“H.264 standard video codecs”).) Thus, because the decoders can only operate on either a video or audio signal, but not both, a POSITA would understand that the “source filter” “extracts media data”

in the form of a separate video and audio signal for decoding by the transform filter using respective video and audio decoders. (EX1005, ¶[0024].) This is further confirmed by Orr's disclosure that the transform filter sends the decompressed signal to "the renderer 218 [which] operates to render *video, audio*, and overlays that are included in the content data." (EX1005, ¶[0024].)

86. In addition, as I discussed above, it would have been obvious to incorporate Wu's separator into Orr's source filter. Wu's separator separates the audio and video signal and is therefore a *demultiplexer*.

87. For the audio decoder, in my opinion, Orr discloses that its playback pipeline includes an audio decoder. Orr teaches that content data includes "audio" and that the "rendering logic" includes elements "needed to decode and render the content." (EX1005, ¶¶[0018]-[0019].) In addition, Orr discloses that the rendering logic includes a "transform filter 216," which "is a pluggable filter" that "decompresses" (*i.e. decodes*) media content using a "decoder." (EX1005, ¶[0024].) The pluggable filter sends the decoded signals to the "renderer" which "render[s] ... audio." (EX1005, ¶[0024].) In other words, Orr teaches that content includes audio and video, further teaches the rendering logic renders the content, and teaches that a decoder is used. Since an audio decoder is needed to decode the audio, a POSITA would therefore understand that the transform filter of the rendering logic includes an *audio decoder*.

88. In my opinion, it also would have been obvious to include an audio decoder in Orr's playback pipeline. Audio decoders to decode digital audio signals for media playback were well-known, including those often used with the H.264 video decoder like AAC. (EX1017, ¶¶[0055] ("The stream demultiplexer ... outputs ... the audio bit stream ABS to the audio decoder 612"); ¶¶[0091]-[0092] ("H.264 ... may be similarly used ... MPEG-2 AAC is used as an encoding scheme for speech signals.")). Orr already teaches that the rendering logic includes a decoder for rendering content, which includes audio. (EX1005, ¶¶[0018]-[0019].) A POSITA would be motivated to give effect to Orr's teachings by using an audio decoder, because that would allow the audio to be decoded and rendered for playback to the user. A POSITA would have had a reasonable expectation of success because audio decoders were ubiquitous at the time of the '139 patent and it was well-within the skill of an ordinary artisan to implement an audio decoder in a playback pipeline.

89. For the video decoder of the playback pipeline, Orr discloses that its "transform filter 216 is a pluggable filter, such as an H264 decoder," a well-known video decoder. (EX1005, ¶[0024].)

90. For the limitation *based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder or a video decoder of the playback pipeline used to play back the first piece of content should be preserved*, Orr discloses this. Orr teaches that if the "[c]ontent

type of new channel [is] same as current channel,” then the system will “[i]nput content of new channel into current rendering pathway.” (EX1005, Fig. 4, blocks 410 and 414.) Because the rendering pathway includes both the source filter (*i.e. demultiplexer*) and transform filter (*i.e. decoders*), a POSITA would understand that this determination checks both the container format and the encoder format. As discussed above, in the proposed combination, this teaching in Orr is modified to first check the container format, and if they have the same container format (*i.e. new content being the same content type as the first content*), then checking if the encoder formats are the same (*i.e. determine whether the audio/video decoders should be preserved*).

91. Wu teaches how to check the encoder formats: by “[o]btaining the audio and video decoder parameters” and determining whether there were “changes in encoding formats,” or whether they “have not changed.” (EX1006, ¶¶[0016], [0028].) If the encoding formats have changed, it is “not feasible to decode and play using the original decoder.” (EX1006, ¶[0005].) Thus, checking whether the encoder formats are the same is a *determination whether an audio decoder or a video decoder of the playback pipeline should be preserved*.

92. For the limitation *based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the playback pipeline while replacing the audio decoder and the*

video decoder, Wu teaches that if it determines that the encoder formats are not the same (*i.e. determine that audio and video decoders should not be preserved*), then “the current audio and video decoders” are “[d]isconnect[ed],” and new audio/video decoders are “create[d] ... based on the aforementioned audio and video decoder parameters,” *i.e. replace the audio and video decoders*. (*Id.*, ¶¶[0079]-[0080].) In Wu’s system, the separator (*i.e. demultiplexer*) is preserved. (*Id.*, Fig. 3.) And in Orr’s system, the protocol stack (*i.e. source element*) is preserved when the decoders are replaced. (EX1005, ¶[0044].) As I discussed above, it would have been obvious to incorporate these teachings into Orr.

2. Claim 2

- a) 2[pre] / 2[a]: *The method of claim 1, wherein the determining whether to preserve comprises: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and*

93. In my opinion, Orr and Wu disclose this limitation.

94. Orr teaches ***detecting the format and codec requirements of the new content as part of the content-type determination***. Orr describes its channel switch logic as determining “whether or not the functional elements 118 that currently exist on the rendering pathway 120 are able to process and render the content on the new channel.” (EX1005, ¶[0021].) A POSITA would understand that the format and codec requirements of content are included in metadata of that content. In order to

“process and render content,” the rendering software must have the correct software compatible with the content—*i.e.* the correct software for handling the content’s format and the correct codec for decoding the content based on how it was compressed.

95. Thus, determining whether the elements can *process and render* content requires ascertaining the codec and format requirements of the new channel’s content, since a determination must be made that the rendering pathway has the right software and codecs to process and render the content. Orr confirms this when it states that “there is no guarantee that the source 214 and transform 216 filters to *decode* the new channel will be the same as those being used for the current channel” (EX1005, ¶[0026]), confirming that the system evaluates what codec types the new content requires. Orr’s claim 4 also expressly recites “determining that the second content can be rendered using the first rendering pathway if the second content is of the same format as the first content.” This necessarily involves detecting the format and codec requirements of the new content. (EX1005, claim 4.)

96. In addition, Wu also teaches that its system “obtain[s] the audio and video decoder parameters,” and determines “changes in encoding formats” or whether they “have not changed.” (EX1006, ¶¶[0016], [0028].) It would have been obvious to incorporate this teaching into Orr, as I discussed above.

- b) *2[b]: determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.*

97. In my opinion, Orr and Wu disclose this limitation.

98. Orr teaches this determination as part of its format compatibility check.

Having determined the format and codec requirements of the new content, Orr's processing logic determines whether the existing functional elements (including the transform filter and decoders) are capable of processing content in that format: "channel switch logic 112 determines whether or not the functional elements 118 that currently exist on the rendering pathway 120 are able to process and render the content on the new channel." (EX1005, ¶[0021].) As I discussed above in claim 1, Orr's transform filter, which is part of the rendering pathway, uses "a pluggable filter" involving codecs. (EX1005, ¶[0024].) In order for the rendering pathway to be able to "process and render content," the transform filter must have the compatible codec for the new content, including audio and video decoders. Thus, Orr teaches determining whether the rendering pathway has the audio and video decoders indicated by the codec information.

99. Wu also teaches that its system "obtain[s] the audio and video decoder parameters," and determines "changes in encoding formats" or whether they "have not changed." (EX1006, ¶¶[0016], [0028].) It would have been obvious to incorporate this teaching into Orr, as I discussed above.

3. Claim 3: The method of claim 2, further comprising preserving an entirety of the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.

100. In my opinion, Orr discloses this limitation.

101. Orr teaches preserving the entirety of the existing rendering pathway when the content types match. As I discussed above in limitation 1[e], Orr discloses a *playback pipeline* in the form of a protocol stack (*i.e. source element*), a multimedia manager, and rendering logic (*i.e. demultiplexer and audio/video decoders*). Orr teaches that when the processing logic determines that the existing pathway can render the new content, “the channel switch logic 112 does not re-initialize the rendering logic 116 but instead, feeds the content from the new channel into the current rendering pathway.” (EX1005, ¶[0022].) FIG. 4 reflects this at block 414: the entire rendering pathway (source filter, transform filter, and renderer) is preserved and used for the new content without teardown. (EX1005, ¶[0028]; ¶[0047]; FIG. 4, blocks 412–414.) Orr's claim 1 recites “inputting the second content into the first rendering pathway if it is determined that the second content can be rendered using the first rendering pathway,” confirming that the entire existing pathway is preserved and reused. (EX1005, claim 1.) By contrast, if the new content is “formatted differently from the content currently being rendered, the channel switch logic 202 notifies the control logic 212 to construct a new rendering pathway 220 to decode the content on the new channel.” (EX1005, ¶[0027].) Thus, for any

new content, the protocol stack and multimedia manager of the playback pipeline are preserved, and if the new content can be decoded by the preexisting audio and video decoders, then the rendering logic is also preserved. Therefore, the entire playback pipeline is preserved.

4. Claim 4

- a) *4[pre]/4[a]/4[b]: The method of claim 2, further comprising: in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information, obtaining a new audio decoder and a new video decoder based on the codec information.*

102. In my opinion, Orr in view of Wu render obvious this limitation. As I discussed above for limitations 1[e] and 1[f], Orr and Wu disclose obtaining a new audio and video decoder based on the content's codec information.

5. Claim 8

- a) *8[pre]/8[a]: The method of claim 1, further comprising: based on the first piece of content and the new piece of content not being the same content type, deconstructing the playback pipeline by releasing the source element, the demultiplexer, the audio decoder, and the video decoder; and*

103. In my opinion, Orr discloses or renders obvious this limitation.

104. Orr teaches full deconstruction of the rendering logic when content types differ, including when both the container and encoding types differ. At FIG. 4, block 416, when the content types are determined not to match (block 410, "N[o]"), the processing logic controls the rendering logic interface to signal the

rendering logic to tear down the existing rendering pathway. (EX1005, ¶[0044]–¶[0045]; FIG. 4, block 416.) Orr’s claim 6 recites “tearing down the first rendering pathway” in this scenario. (EX1005, claim 6.) The teardown releases all elements of the rendering pathway including the transform filter (corresponding to the demultiplexer, audio decoder, and video decoder), and the renderer. (EX1005, ¶[0045]; FIG. 2.)

105. Orr also discloses that the protocol stack is released. Orr describes that the protocol stack can use RTP to manage downloading of the content. (EX1005, ¶[0017].) However, Orr also describes that content can be provided “as files in (MP4) format,” and a POSITA would further understand that other formats and streaming protocols (*i.e.* container formats) could be used. (EX1005, ¶[0017].) Orr also states that the “multimedia manager 204 communicates with the protocol stack 206 to receive real time content and *also* functions to multiplex in *streams from other transports if needed.*” (*Id.*, ¶[0025].) Thus, Orr teaches that “if needed,” *i.e.* if the streaming format or protocol is different from that supported by the current protocol stack, a new “transport” or protocol stack would be created as it would be needed. Accordingly, in the event the new piece of content is not the same content type because it also uses a completely different protocol or format, it would be necessary to release and create a new protocol stack that is compatible with the new protocol or format.

106. A POSITA would also have considered it obvious to release and create a new protocol stack in the event the new content uses a different protocol. A POSITA would understand that different container formats are supported using different APIs. Therefore, if a different protocol or format is encountered, the multimedia manager would need to use a different API than the current protocol stack. In creating a new protocol stack by calling the new API, a POSITA would be motivated to release the current protocol stack, as it would save computing resources. A POSITA would also have a reasonable expectation of success, as releasing the protocol stack is easily implemented using standard programming techniques and well-within the skill of an ordinary artisan.

b) *8[b]: creating a new playback pipeline by acquiring other resources that are applicable to a content type of the new piece of content.*

107. In my opinion, Orr discloses this limitation.

108. Following the pathway teardown at block 416, FIG. 4 of Orr returns to block 402 (“Control Rendering Logic to construct rendering pathway to render selected channel”), at which point new resources appropriate to the new content’s format are acquired to construct a fresh rendering pathway. (EX1005, ¶[0040]; ¶[0045-46]; FIG. 4.) This includes acquiring a new demultiplexer, audio codec, and video codec—the latter two I discuss above for limitation 4[b]. In addition, Orr’s claim 6 recites “constructing a second rendering pathway to render the second

content” which inherently includes acquiring new elements applicable to the new content type. (EX1005, claim 6.)

109. Further, as I discussed above for limitation 8[a], a POSITA would be motivated to release and create a new protocol stack.

6. Claim 9: The method of claim 1, wherein content types include HTTP Live Streaming (HLS) format and Dynamic Adaptive Streaming over HTTP (DASH) format.

110. In my opinion, Orr discloses or renders obvious this limitation.

111. Orr teaches a channel switching system for devices that receive and render content streams over data networks, including wireless networks and the Internet. (EX1005, ¶¶[0014-17].) Orr’s protocol stack 310 is designed to receive content “from a variety of sources” delivered over any such network. (EX1005, ¶[0033].) Orr further teaches that its system handles content delivered in various formats, including RTP and MP4 formats explicitly referenced as examples. (EX1005, ¶[0017].) The ’139 patent identifies HLS and DASH as examples of adaptive streaming content types. (EX1001, 5:1-10.) HLS and DASH are adaptive HTTP streaming protocols that were well known to POSITAs by the priority date as the predominant formats for streaming content delivery over IP networks, and were a natural and obvious choice of content types to implement in a channel-switching system like Orr’s that targets IP-networked consumer devices. (*See, e.g.*, EX1009, slide 15.) A POSITA would have been motivated to implement Orr’s channel-

switching system using HLS and DASH formats with a reasonable expectation of success, as both formats involve the same type of format-compatibility determination and pipeline reuse that Orr teaches.

7. Claim 10

- a) *The method of claim 1, wherein the receiving the selection, transitioning to the new piece of content, determining whether the new piece of content is of the same content type as the first piece of content, and determining whether to preserve is performed by a player.*

112. In my opinion, Orr discloses this limitation.

113. The '139 patent describes the “player” as the component that constructs the pipeline, manages it, and decides whether to preserve or reconstruct it upon a content switch: “The player 202 is configured to manage the pipeline 204.... the player 20[2] [sic] also decides whether to maintain at least a portion of the pipeline 204 or construct a new pipeline when a switch in content occurs.” (EX1001, 4:66-5:10.) The player thus serves as the top-level control component that receives content selections, initiates transitions, and makes pipeline preservation decisions.

114. Orr’s channel switch logic 202 (which may be embedded within the multimedia manager 204) (EX1005, ¶[0023]; FIG. 2.) performs each of the operations recited in this limitation. It receives channel selection requests from the user interface (EX1005, ¶[0025]; ¶[0042]); determines the content type of the new channel and whether it is compatible with the existing rendering pathway (EX1005,

¶[0021]; ¶[0026]; ¶[0043-44]); and controls the rendering logic to either preserve the existing pathway or construct a new one. (EX1005, ¶[0022]; ¶[0037]; ¶[0044-47].) Orr's claim 7 further confirms this mapping, reciting an “apparatus for channel switching” comprising “processing logic configured to determine whether the real time content can be rendered using an existing rendering pathway” and “a rendering interface configured to input the real time content into the existing rendering pathway.” (EX1005, claim 7.)

8. Claim 11

- a) *11[pre]/11[a]/11[b]: The method of claim 1, further comprising: based on the first piece of content and the new piece of content not being the same content type, releasing the source element; and creating a new playback pipeline by acquiring a different source element that is applicable to a content type of the new piece of content.*

115. In my opinion, Orr discloses this claim for the same reasons I discussed above for claim 8. Claim 8 describes replacing the entire pipeline, including the source element, and therefore my analysis there applies to this claim.

1. Claim 12

- a) *12[pre]/12[a]: The method of claim 1, further comprising: based on the first piece of content and the new piece of content not being the same content type, releasing the demultiplexer; and*

116. As I discussed above in limitation 8[a], when content types are determined not to match at block 410 (“N[o]”), Orr’s channel switch logic signals

the rendering logic to tear down the existing rendering pathway in its entirety at block 416. (EX1005, ¶¶[0044-45]; FIG. 4, block 416.) Orr’s claim 6 recites “tearing down the first rendering pathway” in this scenario. (EX1005, claim 6.) The teardown releases all elements of the rendering pathway including the demultiplexer. (EX1005, ¶[0045]; FIG. 2.)

- b)** *12[b]: creating a new playback pipeline by acquiring a different demultiplexer that is applicable to a content type of the new piece of content.*

117. In my opinion, Orr discloses this limitation.

118. Following the teardown at block 416, Orr returns to block 402 to construct a new rendering pathway with functional elements suited to the new content type. (EX1005, ¶¶[0040], [0045-46]; FIG. 4.) As discussed above in limitation 1[e], Orr’s source filter 214 performs demultiplexing functions by extracting media data from the received transport stream for presentation to the transform filter. (EX1005, ¶[0024].) Orr teaches that the source filter “understands the format of the received transport stream” in performing this function. (EX1005, ¶[0024].) Because the source filter’s demultiplexing function is tied to understanding the format of the content stream, a POSITA would understand that when a new rendering pathway is constructed for content of a different type, the new source filter acquired for that pathway must be one capable of understanding and extracting media data from the new content type, *i.e.*, a different element performing

demultiplexing that is applicable to the new content type. This is further confirmed by Orr's teaching that different content types require different rendering pathways constructed with appropriate functional elements: "the channel switch logic 202 notifies the control logic 212 to construct a new rendering pathway 220 to decode the content on the new channel." (EX1005, ¶[0027].) Orr's claim 6 recites "constructing a second rendering pathway to render the second content" following teardown, which inherently includes acquiring a new source filter suited to the new content's format. (EX1005, claim 6.)

2. Claim 14

a) 14[pre]: A system comprising:

119. In my opinion, for the reasons I discussed above for limitation 1[pre], Orr discloses this limitation.

b) 14[a]: one or more hardware processors; and

120. In my opinion, Orr discloses this limitation. Orr's device contains processing logic which is described as comprising "at least one of a CPU, processor, gate array, hardware logic, memory elements, virtual machine, and/or hardware executing software." (EX1005, ¶[0031].) Orr's apparatus claims similarly recite "processing logic configured to determine whether the real time content can be rendered using an existing rendering pathway." (EX1005, claim 7.) A POSITA would understand Orr's processing logic as constituting "one or more hardware processors" within the meaning of this limitation.

- c) *14[b]: a storage device storing instructions that configure the one or more hardware processors to perform operations comprising:*

121. In my opinion, Orr discloses this limitation. Orr teaches that its channel switching system “comprises a computer program product having one or more program instructions (‘instructions’) or sets of ‘codes’ stored or embodied on a machine-readable medium,” and that “[w]hen the codes are executed by at least one processor ... their execution causes the processor to provide the functions of the channel switching system described herein.” (EX1005, ¶[0038].) Orr further describes specific example storage media including “a floppy disk, CDROM, memory card, FLASH memory device, RAM, ROM, or any other type of memory device or machine-readable medium.” (*Id.*) Orr’s claim 19 recites a corresponding computer program product comprising “a machine-readable medium encoded with codes executable to: receive a channel selection indicator ...; determine whether the real time content can be rendered using an existing rendering pathway; and input the real time content into the existing rendering pathway.” (EX1005, claim 19.)

- d) *14[c]: causing playback of a first piece of content using a playback pipeline;*

122. In my opinion, for the reasons I discussed above for limitation 1[a], Orr discloses this limitation.

- e) *14[d]: receiving a selection of a new piece of content for playback during playback of the first piece of content;*

123. In my opinion, for the reasons I discussed above for limitation 1[b], Orr discloses this limitation.

- f) *14[e]: in response to the receiving of the selection, transitioning from playback of the first piece of content to the new piece of content;*

124. In my opinion, for the reasons I discussed above for limitation 1[c], Orr discloses this limitation.

- g) *14[f]: determining whether the new piece of content is of a same content type as the first piece of content;*

125. In my opinion, for the reasons I discussed above for limitation 1[d], Orr discloses or renders obvious this limitation.

- h) *14[g]: based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder and a video decoder of the playback pipeline used to play back the first piece of content should be preserved; and*

126. In my opinion, for the reasons I discussed above for limitation 1[e], Orr discloses or renders obvious this limitation.

- i) *14[h]: based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the*

playback pipeline while replacing the audio decoder and the video decoder.

127. In my opinion, for the reasons I discussed above for limitation 1[f], Orr discloses or renders obvious this limitation.

3. Claim 15

- a) *The system of claim 14, wherein the determining whether the audio decoder and the video decoder should be preserved comprises: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.*

128. In my opinion, for the reasons I discussed above for claim 2, Orr discloses this limitation.

4. Claim 16

- a) *The system of claim 15, wherein the operations further comprise: preserving an entirety of the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.*

129. In my opinion, for the reasons I discussed above for claim 3, Orr discloses this limitation.

5. Claim 17

- a) *The system of claim 15, wherein the operations further comprise: in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information,*

obtaining a new audio decoder and a new video decoder based on the codec information.

130. In my opinion, for the reasons I discussed above for claim 4, Orr discloses this limitation.

6. Claim 18

- a) *The system of claim 14, wherein the operations further comprise: based on the first piece of content and the new piece of content not being the same content type, deconstructing the playback pipeline by releasing the source element, the demultiplexer, the audio decoder, and the video decoder; and creating a new playback pipeline by acquiring other resources that are applicable to a content type of the new piece of content.*

131. In my opinion, for the reasons I discussed above for claim 8, Orr discloses this limitation.

7. Claim 19

- a) *19[pre]: A machine-storage medium storing instructions that when executed by one or more hardware processors of a machine, cause the machine to perform operations comprising:*

132. In my opinion, for the reasons I discussed above in limitations 14[a] and 14[b], Orr discloses this limitation.

- b) *19[a]: causing playback of a first piece of content using a playback pipeline;*

133. In my opinion, for the reasons I discussed above in limitation 1[a], Orr discloses this limitation.

- c) *19[b]: receiving a selection of a new piece of content for playback during playback of a first piece of content;*

134. In my opinion, for the reasons I discussed above in limitation 1[b], Orr discloses this limitation.

- d) *19[c]: in response to the receiving of the selection, transitioning from playback of the first piece of content to the new piece of content;*

135. In my opinion, for the reasons I discussed above in limitation 1[c], Orr discloses this limitation.

- e) *19[d]: determining whether the new piece of content is of a same content type as the first piece of content;*

136. In my opinion, for the reasons I discussed above in limitation 1[d], Orr in view of Wu renders obvious this limitation.

- f) *19[e]: based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder of the playback pipeline used to play back the first piece of content should be preserved; and*

137. In my opinion, for the reasons I discussed above in limitation 1[e], Orr in view of Wu renders obvious this limitation.

- g) *19[f]: based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the playback pipeline while replacing the audio decoder and the video decoder.*

138. In my opinion, for the reasons I discussed above in limitation 1[f], Orr in view of Wu renders obvious this limitation.

VII. Ground 2: Orr in View of Wu and Hsu Renders Obvious Claims 6-7 and 13

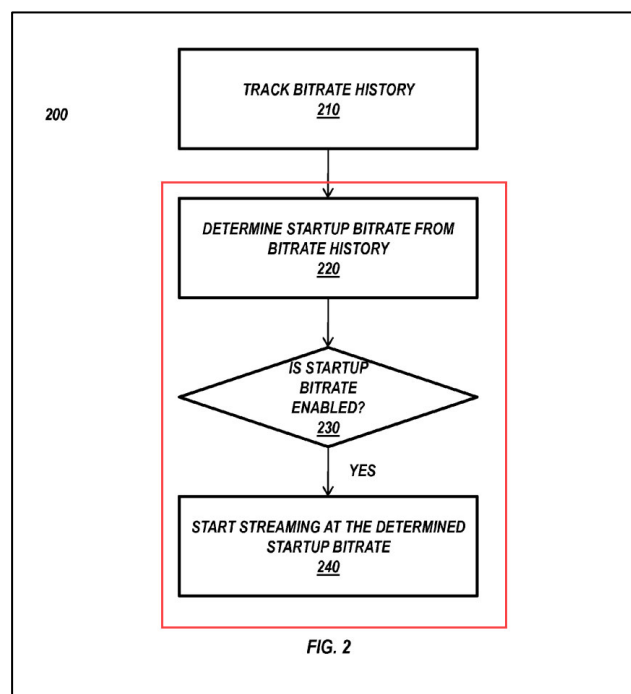
A. Motivation to Combine

139. In my opinion, a POSITA would have been motivated to modify Orr and Wu in further view of Hsu to incorporate teachings of bitrate initialization and would have had a reasonable expectation of success. A POSITA would have been motivated by the known benefits of reducing latency and degradation of the stream quality by reusing the current bitrate for new content, as changing to the default bitrate for new content requires time and resources, and also causes the new content to initially stream at a lower quality.

140. In my proposed combination, Orr's protocol stack continues to manage the retrieval of content data from the content source, even when combined with Wu. (EX1005, ¶[0033].) Hsu teaches initializing a new content download session at a historically derived bitrate rather than the system default. (EX1007, 6:1-5.) A POSITA would implement Hsu's bitrate initialization teaching in the protocol stack of Orr's pipeline, configuring it to begin downloading new content at the previously achieved bitrate. This requires no architectural modification to Orr and produces the entirely predictable result of faster, higher-quality content transitions. Moreover, the modification is well-within the skills of an ordinary artisan, as bitrate initialization was well-known and straightforward to implement. Neither reference teaches away

from the combination. A POSITA would therefore have a reasonable expectation of success.

141. Both Orr and Hsu address the same fundamental user experience problem: the latency and quality degradation experienced by a user when switching between pieces of content on a streaming device. Orr addresses the pipeline reconstruction component of this problem: eliminating the delay caused by tearing down and rebuilding the rendering pathway. (EX1005, ¶[0004]–¶[0005]; ¶[0015].) As shown in FIG. 2 below, Hsu addresses the bitrate initialization component: eliminating the quality degradation and latency caused by defaulting to the lowest bitrate at the start of each new session. (EX1007, 5:33-39; FIG. 2.) This would therefore motivate a POSITA to further improve the user experience consistent with Orr’s goals, which increases streaming quality and reduces latency.



142. Orr and Hsu are therefore complementary solutions: Orr operates at the pipeline and rendering layer, while Hsu operates at the download and bitrate layer. A POSITA designing a comprehensive content-switch optimization system would naturally consider and implement both.

143. A POSITA combining Orr and Hsu would implement Hsu's bitrate initialization teaching in the source element of Orr's pipeline. For example, by configuring the source element to initiate the new content download at the bitrate derived from Hsu's historical tracking mechanism, rather than at the system default lowest bitrate. This integration is architecturally natural and technically straightforward, as the source element is the component that manages download sessions and bitrate selection in Orr's pipeline.

144. In addition, both references are directed to the same class of devices. Orr is directed to devices receiving and rendering real-time content streams on wireless networks and Internet-connected consumer devices. (EX1005, ¶[0014-16].) Hsu is directed to adaptive bitrate streaming media players on client devices explicitly including "set-top box, Internet-enabled television" and other consumer electronics. (EX1007, 4:50-58.) A POSITA working on content-switch optimization for any such device would routinely consult both references.

B. Limitation-By-Limitation Analysis

- 1. Claim 6: The method of any one of claim 1, the transitioning from playback of the first piece of content to the new piece of**

content comprises initiating playback of the new piece of content using a bitrate used for the playback of the first piece of content; and

145. In my opinion, Hsu discloses this limitation and therefore in combination with Orr renders obvious this limitation.

146. Hsu addresses bitrate maintenance across content transitions. Hsu discloses that, upon initiating playback of new content, the player uses a bitrate derived from the history of prior sessions, specifically the bitrate used during prior playback, as the startup bitrate for the new content: “The technology can use the most recent bitrate as the startup bitrate.” (EX1007, 6:1-5.) Hsu’s claim 1 recites “determining a startup bitrate based on the bitrate history, the startup bitrate comprising an initial bitrate for streaming the media content ... wherein the streaming begins at the determined startup bitrate.” The bitrate “used for playback of the first piece of content” in claim 6 of the ’139 patent corresponds directly to the “most recent bitrate” tracked by Hsu’s player, which is used to initialize the new streaming session.

- a) *6[b]: the determining whether to preserve occurs during playback of the new piece of content using the bitrate.*

147. In my opinion, Hsu discloses this limitation and therefore in combination with Orr renders obvious this limitation.

148. Hsu teaches continuous monitoring of whether the current bitrate can be maintained during playback. Hsu's adaptive bitrate player "determin[es] bitrate values used at a plurality of different times for providing the streaming media content" during playback, tracking whether the current bitrate is sustainable given available network bandwidth. (EX1007, claim 1). This continuous monitoring is the mechanism by which the adaptive bitrate system detects that the current bitrate is or is not achievable for the new content. When combined with Orr, the combination renders obvious this limitation.

2. Claim 7

- a) *7[pre]/7[a]: The method of claim 6, further comprising: during playback of the new piece of content using the bitrate, detecting whether the new piece of content can maintain playback at the bitrate; and*

149. In my opinion, Hsu discloses this limitation and therefore in combination with Orr renders obvious this limitation.

150. Hsu teaches continuous monitoring of whether the current bitrate can be maintained during playback. Hsu's adaptive bitrate player "determin[es] bitrate values used at a plurality of different times for providing the streaming media content" during playback, tracking whether the current bitrate is sustainable given available network bandwidth. (EX1007, claim 1). This continuous monitoring is the

mechanism by which the adaptive bitrate system detects that the maintained bitrate is or is not achievable for the new content.

- b)** *7[b]: in response to the detecting that the new piece of content cannot continue playback at the bitrate, adjusting the bitrate.*

151. In my opinion, Hsu discloses this limitation and therefore in combination with Orr renders obvious this limitation.

152. Hsu teaches downward bitrate adjustment when the current bitrate cannot be sustained. Hsu describes a capping mechanism that accounts for users moving between higher and lower bandwidth environments, and the standard adaptive bitrate mechanism taught by Hsu lowers the bitrate when download throughput is insufficient to sustain the current level: the player “adjusts the quality of a video stream accordingly” when bandwidth weakens (EX1007, 3:62-4:5) and describes the system's response to lower-bandwidth environments as selecting a lower bitrate. (EX1007, 6:16-26.)

3. Claim 13

- a)** *The method of claim 1, wherein the source element initiates the playback of the new piece of content using a same bitrate as the first piece of content and adjusts the bitrate in response to detecting that the new piece of content cannot continue playback at the bitrate.*

153. I note that this claim combines the requirements of claims 6 and 7 and attributes them specifically to the source element. Thus, for the same reasons I discussed above for claims 6 and 7, Hsu discloses this limitation, and it would have

been obvious to incorporate these teachings into Orr's protocol stack (*i.e. source element*).

154. With respect to *the source element initiating playback of the new piece of content using the same bitrate as the first piece of content*, Hsu teaches that the player initiates new content streaming at the historically derived startup bitrate: "The technology can use the most recent bitrate as the startup bitrate." (EX1007, 6:1-5.) Hsu's claim 1 further recites "determining a startup bitrate based on the bitrate history, the startup bitrate comprising an initial bitrate for streaming the media content... wherein the streaming begins at the determined startup bitrate." (EX1007, claim 1.)

155. With respect to *the source element adjusting the bitrate when the new content cannot continue playback at that bitrate*, Hsu expressly teaches that the adaptive bitrate system adjusts the quality of the stream downward when bandwidth conditions cannot support the current bitrate: the system "detect[s] a playback device bandwidth and other factors in real time and adjust[s] the quality of a video stream accordingly." (EX1007, 3:62-4:5; *see also*, 6:16-26.)

156. A POSITA integrating Hsu into Orr would be motivated to implement both of these teachings in the source filter of Orr's pipeline, the component responsible for obtaining content data from the content source and presenting it for downstream processing. A POSITA would have readily incorporate these teaching

into Orr to ensure that a new piece of content played smoothly without quality degradation. The source filter is a natural and logical integration point for Hsu's bitrate teachings within Orr's architecture. It is the upstream component that controls how content data is retrieved and the component at which an initial bitrate parameter would be set and at which bitrate adjustments would be made in response to network conditions. A POSITA applying Orr's pipeline reuse architecture to an adaptive HTTP streaming context would understand that the source filter in such an implementation manages the adaptive bitrate download session, and would apply Hsu's startup bitrate and adjustment teachings to that component as a matter of ordinary skill.

VIII. Ground 3: Orr in View of Wu and Lim Renders Obvious Claim 5

A. Motivation to Combine

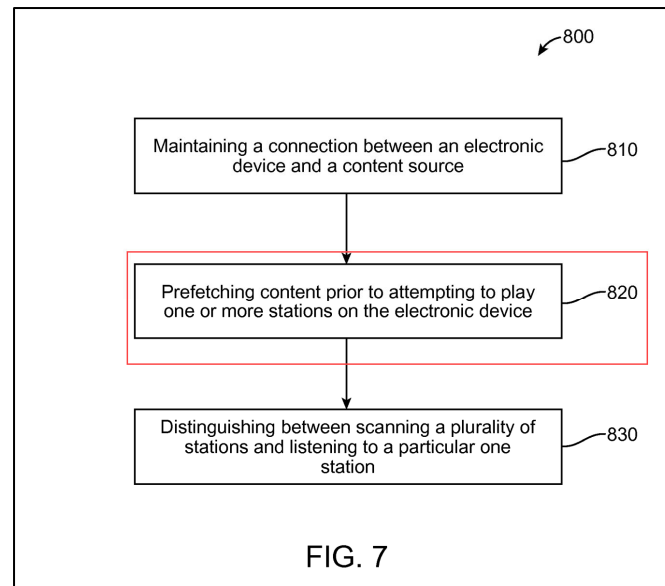
157. In my opinion, a POSITA would have been motivated to modify Orr and Wu in further view of Lim to incorporate Lim's teachings of prefetching of metadata into Orr, specifically Orr's multimedia manager (as modified by Wu), and would have had a reasonable expectation of success in doing so.

158. A POSITA would be motivated by the fact that Lim's prefetched metadata reduces latency by supplying in advance the content-type information that Orr's format compatibility determination requires. Rather than obtaining this information at the time of the channel switch, a POSITA would have integrated

Lim's prefetched metadata to perform Orr's compatibility check instantaneously before receiving the user's selection, thereby reducing the latency typically needed when switching to new content and allowing for a seamless transition to video. This is the straightforward application of prefetching, a well-known technique for reducing access latency, to the specific bottleneck of on-demand content-type retrieval in Orr's channel-switch system.

159. Orr and Lim address sequential and complementary steps in the same content-switch optimization workflow. Orr's channel switch logic performs a format compatibility determination at the time of the channel switch (after the user has made a selection) to decide whether the existing rendering pathway can be reused to reduce latency. (EX1005, ¶¶[0021]; ¶¶[0043-44].) To make this determination, Orr's processing logic must have access to information about the new content's format. (EX1005, ¶¶[0026].)

160. Lim teaches prefetching this kind of content information (metadata including type, format, and codec-relevant data) during playback of current content, before the user makes a selection. (EX1008, *e.g.*, 3:51-67; 5:23-39; 15:37-46; FIG. 2; claim 1.) This reduces latency between a user's content selection and the commencement of playback, because "when the user changes the content source (*e.g.*, radio stations, video stations, etc.), the mobile device starts playing without reduced buffering time." (EX1008, 4:1-7.) This is shown below in Figure 7 of Lim:



161. Instead of waiting for Orr’s channel switch logic to perform a format compatibility determination at the time of the channel switch (after the user has made a selection), a POSITA would have been motivated to integrate Lim’s prefetching of metadata *before* the user makes a selection to produce faster, higher-quality content transitions between content.

162. Both references are directed to reducing the latency experienced when a user switches between content channels on a playback device, and both are premised on the same insight: that preparing the system in advance of a user's selection reduces the delay after that selection is made. Orr prepares the rendering pathway by evaluating whether existing components can be reused; Lim prepares the content metadata so it is available the moment a selection occurs. A POSITA working on content-switch optimization would recognize these as naturally complementary techniques operating at different points in the same workflow.

163. In my opinion, a POSITA would have reasonably expected the combination to succeed. Lim provides detailed disclosures regarding implementation of prefetching, and it would have been well-within the skill of an ordinary artisan to implement such functionality into the software or hardware of Orr's multimedia manager. Such a modification would only require ordinary programming knowledge.

B. Limitation-By-Limitation Analysis

1. Claim 5

- a)** *The method of claim 2, wherein the detecting the codec information for the new piece of content comprises examining prefetched metadata for the new piece of content.*

164. Claim 5 adds to the codec detection requirement of claim 2 the specific limitation that the codec information is obtained by examining metadata that was prefetched, such as retrieved and stored locally before the user selects the content.

165. As I discussed above for claim 2, Orr in view of Wu teaches detecting of codec and format information. Specifically, Orr's channel switch logic determines the content format of the new channel and assesses whether the existing pipeline elements can handle it.

166. Lim teaches prefetching metadata for candidate next-play content during playback of current content. Lim's claim 1 recites "prefetching, by the electronic device, content prior to attempting to play at least one station on the

electronic device,” and claim 2 expressly specifies that “prefetching content comprises prefetching preview content clips, cover art, and metadata for all on-deck content for the plurality of stations.” (EX1008, claims 1-2.) Lim’s specification confirms this teaching: Lim describes “the client 305 efficiently prefetch[ing] the preview clips, cover art, and metadata for all the on-deck songs of all the stations” using a cold start function while current content plays. (EX1008, 9:6-10.) Lim further describes the prefetch manager module 450 implementing a “Prefetch Song” process that “downloads a song’s preview clip and cover art and stores them in the prefetch inventory 455 and image cache 470,” and a “CheckOnDeckPrefetch” process that triggers a prefetch of the next on-deck song’s data while the current song still has sufficient buffer remaining, *i.e.*, while current content is still playing. (EX1008, 12:44-55.) Lim further describes the metadata prefetched for each song as including the “song URL and cover art URL,” and notes that the aggregate station binary downloaded at cold start contains “all of the stations’ first song metadata, preview clip, and cover-art.” (EX1008, 10:27-41; 12:37-42.)

167. In the combination with Orr, Orr’s format compatibility check uses metadata prefetched per Lim’s method. Lim’s prefetch manager module 450 fetches and stores in the prefetch inventory 455 the metadata for candidate next-play content during playback of current content. When the user makes a channel selection, Orr’s processing logic reads from the locally cached prefetch inventory rather than waiting

to obtain format information from the incoming stream, enabling the compatibility determination to be made instantaneously upon selection.

IX. Ground 4: Hervey Renders Obvious Claims 1-4, 8-9, 11-14, 18, and 20

A. Limitation-By-Limitation Analysis

1. Claim 1

a) *l[pre]: A method comprising:*

168. In my opinion, and to the extent limiting, Hervey discloses the preamble of claim 1.

169. Hervey describes GStreamer as a software framework for implementing playback pipelines on electronic devices that receive and process digital content streams, precisely the function of a *digital receiver*. Specifically, Hervey describes playbin as a “convenience pipeline” that manages uridecodebin, decodebin, and playsink to receive a URI, decode the content, and present it to a sink, respectively. This is the complete sequence of operations performed by a digital receiver. (EX1009, slide 4.)

Playback in GStreamer

- playbin (convenience pipeline)
 - uridecodebin
 - decodebin
 - playsink
- And some non-reusable code

170. A POSITA familiar with GStreamer would understand that the framework described in Hervey is a software framework that is deployed on devices in order to receive and render digital content, and would understand the methods described in Hervey as performed by such a device.

- b)** *1[a]: causing playback of a first piece of content using a playback pipeline;*

171. In my opinion, Hervey discloses this limitation.

172. Hervey describes a GStreamer-based playback framework in which content is played back through a pipeline of functional elements. Hervey describes playbin as a “convenience pipeline” that incorporates uridecodebin, decodebin, and playsink to receive a URI, decode the content, and present it to a sink for rendering. (EX1009, 4.) Hervey further describes uridecodebin as obtaining “the proper source element to use for the URI” and performing “buffering/queueing” of network

streams, which constitutes managing the downloading of content data for playback.

(EX1009, 13.) The output of uridecodebin feeds into decodebin3, which comprises GstParseBin – a component that recursively identifies and instantiates the demuxer and parser elements needed to parse the content stream (EX1009, 25), followed by a multiqueue and decoder elements including a video decoder (VDec) and audio decoders (ADec). (EX1009, 24, 26.) These components arranged in series from content acquisition through parsing and decoding to rendering constitute a ***playback pipeline*** within the meaning of this limitation. A POSITA familiar with GStreamer would understand the playbin/decodebin3 stack described in Hervey as a software framework deployed on devices to receive and render digital content through such a pipeline.

173. The causing of playback of a first piece of content using this pipeline is reflected throughout Hervey’s description of active playback as the baseline state from which stream switching occurs. For example, Hervey describes the scenario of switching streams during active playback (“we might switch streams later on ... we don’t want the other stream to be drained completely”) confirming that the pipeline is actively playing back a first piece of content prior to any switch. (EX1009, 7.)

- c) *1[b]: receiving, by a digital receiver, a selection of a new piece of content for playback during playback of a first piece of content;*

174. Hervey describes receiving a new stream selection during active playback of a first stream. For example, Hervey describes receiving a GST_EVENT_SELECT_STREAMS event during ongoing playback which “[c]ompare[s] available, active, pending and requested streams.” The reference to “active” streams confirms that the selection is received while first content is playing. (EX1009, 28.)

Switching Streams

- GST_EVENT_SELECT_STREAMS
 - Ex : “video1”, “audio2”
 - Compare available, active, pending and requested streams.
 - Need to switch audio1 to audio2
 - Put audio2 in pending selection
 - Set idle probe on multiqueue output of audio1
 - Unlinks from decoder
 - Send GST_EVENT_RECONFIGURE to audio2 multiqueue source pad
 - Audio2 multiqueue source pad will use same logic as before

175. Hervey further describes the system receiving a new content input while current content plays: “When EOS reaches all multiqueue slot input, emit ‘about-to-finish,’ [u]sers have a chance to change the input of decodebin3.”

(EX1009, 32.) A POSITA would understand that the “about-to-finish” flag indicates that the current playing content is about to finish.

Gapless playback in decodebin3

- Can we re-use more elements ?
- Just re-use elements for gapless playback
- When EOS reaches all multiqueue slot input, emit 'about-to-finish'
 - Users have a chance to change the input of decodebin3
 - Earlier than with legacy playbin (EOS on output of decodebin)
- GstParseBin reconfigurable
 - Handles the reconfiguration
 - And we just get new streams... as if it was a demuxer update

d) *l[c]: in response to the receiving of the selection, transitioning from playback of the first piece of content to the new piece of content;*

176. Hervey teaches transitioning to the newly selected stream in direct response to the stream selection event. Upon receiving GST_EVENT_SELECT_STREAMS, decodebin3 performs the stream comparison and pipeline reconfiguration steps and then begins delivering the new content through the pipeline: “Send GST_EVENT_RECONFIGURE to audio2 multiqueue source pad,” and “Audio2 multiqueue source pad will use same logic as before.” (EX1009, 28.) Because the transition to the new content cannot occur until the new content is selected, the transitioning is therefore in response to the selection.

Switching Streams

- `GST_EVENT_SELECT_STREAMS`
 - Ex : “video1”, “audio2”
 - Compare available, active, pending and requested streams.
 - Need to switch audio1 to audio2
 - Put audio2 in pending selection
 - Set idle probe on multiqueue output of audio1
 - Unlinks from decoder
 - Send `GST_EVENT_RECONFIGURE` to audio2 multiqueue source pad
 - Audio2 multiqueue source pad will use same logic as before

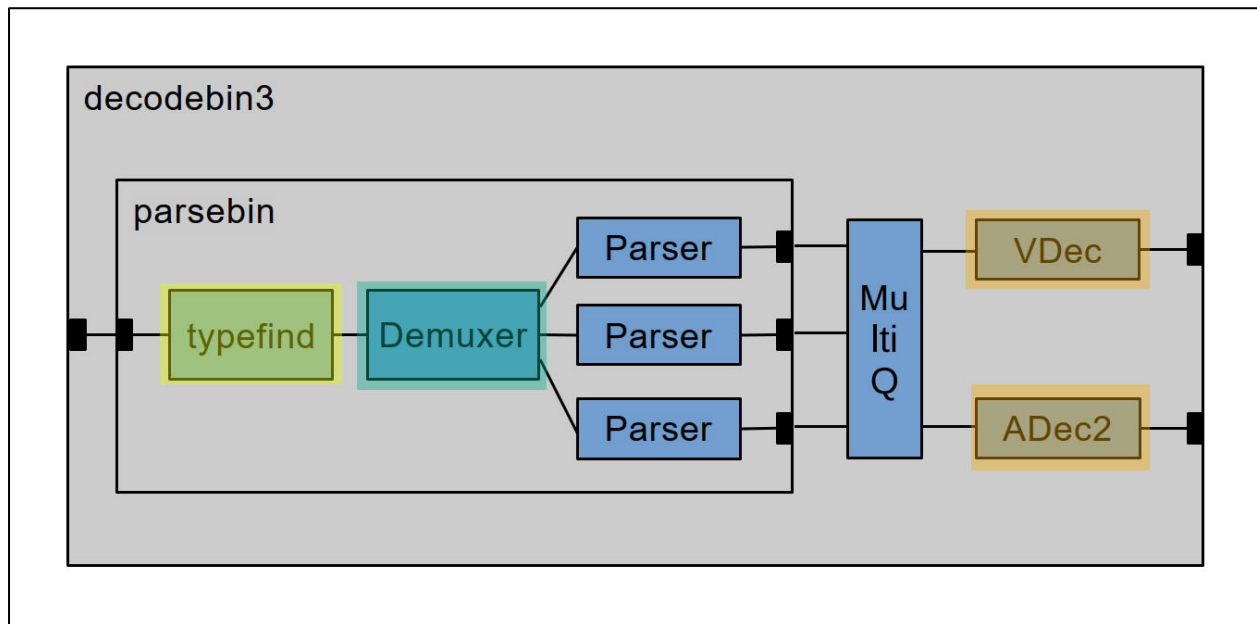
177. The gapless playback slide similarly describes switching to new content upon the “about-to-finish” signal. (EX1009, 32.)

Gapless playback in decodebin3

- Can we re-use more elements ?
- Just re-use elements for gapless playback
- When EOS reaches all multiqueue slot input, emit 'about-to-finish'
 - Users have a chance to change the input of decodebin3
 - Earlier than with legacy playbin (EOS on output of decodebin)
- `GstParseBin` reconfigurable
 - Handles the reconfiguration
 - And we just get new streams... as if it was a demuxer update

- e) *1[d]: determining whether the new piece of content is of a same content type as the first piece of content; and*

178. Hervey teaches that its pipeline first detects the content type of a new, incoming stream. Specifically, Hervey teaches that its pipeline contains an element called “parsebin,” which has a pipeline that uses a “single input sink pad” which inputs into a “typefind” element. (EX1009, 24-25.) The “typefind” element connects to a “Demuxer.” (*Id.*)



179. A POSITA would understand that the “typefind” element detects the content type of the new incoming stream. Contemporaneous GStreamer documentation discloses that the “typefind” element’s purpose is to “detect the media type of an unknown stream.” (EX1019 (GstTypeFind documentation).)

180. Hervey teaches that once the pipeline’s “parsebin” detects the new content’s content type, it will “[r]ecursively figure[] out ‘decodable’ elements

needed,” including the “[d]emuxer[.]” (EX1009, 25.) Hervey specifically teaches that for “input changes,” the “parsebin” element “re-use[s] elements if possible.” (*Id.*) Thus, Hervey teaches that its “parsebin” element will **determine whether the new content has the same container type as the old content** and, if it does, reuse the demuxer. Of note, Hervey explains that its “parsebin” does not determine which decoder to use. (EX1009, 25.) As discussed below in limitations 1[e] and 1[f], a different portion of the pipeline performs that function

181. Hervey teaches determining whether a new stream is compatible with the existing pipeline, which a POSITA would understand as determining whether the new content is of the same content type as the current content. The “Switching Streams” slide describes comparing “available, active, pending and requested streams” to determine what action to take. (EX1009, 28.)

Switching Streams

- GST_EVENT_SELECT_STREAMS
 - Ex : “videol”, “audio2”
 - Compare available, active, pending and requested streams.
 - Need to switch audiol to audio2
 - Put audio2 in pending selection
 - Set idle probe on multiqueue output of audiol
 - Unlinks from decoder
 - Send GST_EVENT_RECONFIGURE to audio2 multiqueue source pad
 - Audio2 multiqueue source pad will use same logic as before

182. The “Re-using decoders” slide describes examining the CAPS of the new stream: “STREAM_START / CAPS on multiqueue source pad.” For example, this “[c]hecks if in requested/pending selection,” “[c]heck[s] if there is an output available to be re-used,” and “[c]heck[s] if decoder can accept CAPS.” (EX1009, 29.)

Re-using decoders

- STREAM_START / CAPS on multiqueue source pad
 - Checks if in requested / pending selection
 - Check if there is an output available to be re-used
 - Check if decoder can accept CAPS
 - If so, just link
 - If not, unlink, insert new decoder, link

183. CAPS in GStreamer is the descriptor of stream format and content type. (EX1018, 2 (“Caps (capabilities) are lightweight refcounted objects describing media types.”) A POSITA would understand that checking whether a decoder can accept a stream’s CAPS determines whether the new content is of the same content type as the content currently being decoded. This is because a decoder of one particular type can only decode audio or video streams of that same type. Accordingly, checking “if decoder can accept CAPS” refers to checking if the new CAPS contains content that can be decoded by the decoder compatible with the first content, *i.e. has the same content type*.

- f) *l[e]: based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder or a video decoder of the playback pipeline used to play back the first piece of content should be preserved; and*
- g) *l[f]: based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the playback pipeline while replacing the audio decoder and the video decoder.*

184. Hervey discloses these limitations.

185. These limitations require (1) a “playback pipeline” for the first piece of content that has a source element, a demultiplexer, an audio decoder, and a video decoder, and (2) based on determining the new content has the same content type, determining whether to preserve an audio decoder or a video decoder of the “playback pipeline,” and (3) based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer while replacing the audio and video decoder.

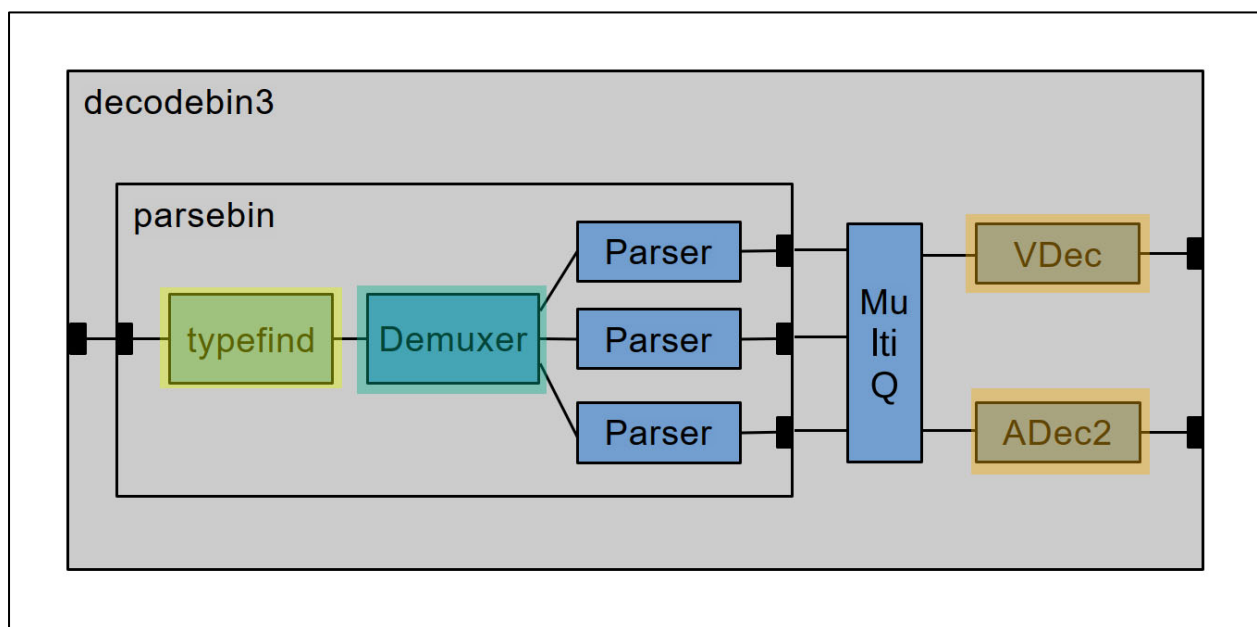
186. I will first discuss the limitation of a *playback pipeline* that *comprises a source element, a demultiplexer, an audio decoder, and a video decoder*. Hevey’s decodebin3 architecture explicitly discloses a *playback pipeline* comprising the claimed *source element, demultiplexer, audio decoder, and video decoder*. I discuss each component below.

187. For the “[s]ource element,” Hervey teaches that decodebin3’s *playback pipeline* has a “single input sink pad,” *i.e.* a single element for receiving media input. (EX1009, 25.) Hervey teaches that decodebin3 has a “typefind” element that connects to a “Demuxer.” (EX1009, 24.)

188. For the “demultiplexer,” Hervey discloses that decodebin3’s *playback pipeline* has a “Demuxer,” which is a demultiplexer. (EX1009, 24.) The demuxer outputs a “GstStreamCollection,” which is a “collection” of streams, indicating that it demultiplexes the audio and video signals. (EX1009, 20.)

189. For the “audio decoder,” Hervey discloses that decodebin3’s *playback pipeline* has an “ADec2,” *i.e.* an *audio decoder*. (EX1009, 24.)

190. For the “video decoder,” Hervey discloses that decodebin3’s *playback pipeline* has a “VDec,” *i.e.* a *video decoder*. (EX1009, 24.)



191. For the limitation *based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder or a video decoder of the playback pipeline used to play back the first piece of content should be preserved*, as I discussed above in limitation 1[d], Hervey teaches that if the new content's container type (*i.e. content type*) is the same as the old content, Hervey will reuse the demultiplexer. Subsequently, Hervey teaches that it will determine if *the audio and video decoders used for the old content should be preserved*. Specifically, Hervey teaches its pipeline will "[r]e-use decoders," specifically: "Check if decoder can accept CAPS. If so, just link. If not, unlink, insert new decoder, link." (EX1009, 29.)

Re-using decoders

- STREAM_START / CAPS on multiqueue source pad
 - Checks if in requested / pending selection
 - Check if there is an output available to be re-used
 - Check if decoder can accept CAPS
 - If so, just link
 - If not, unlink, insert new decoder, link

192. This is a determination of whether to preserve the existing audio and video decoders. If the decoders can decode the new content, the decoder is preserved ("just link"); if not, it is replaced ("unlink, insert new decoder").

193. For the limitation *based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the playback pipeline while replacing the audio decoder and the video decoder*, as I just discussed, Hervey teaches that based on whether the “decoder can[not] accept CAPs,” *i.e. a determination that the audio and video decoders should not be preserved*, it will “unlink” the old decoders and “insert new decoder[s],” *i.e. replace the audio and video decoder*. (EX1009, 29.)

194. As I also discussed above for limitation 1[d], Hervey discloses preserving the *demultiplexer* if the new content’s container type (*i.e. content type*) is the same as the old content’s container type.

195. Hervey also discloses preserving the *source element*. Specifically, Hervey discloses “[o]ne single input sink pad.” (EX1009, 25.) In addition, Hervey teaches that it attempts to “[r]e-use as many elements as possible.” (EX1009, 34.)

2. Claim 2

- a) 2[pre]/2[a]: *The method of claim 1, wherein the determining whether the audio decoder and the video decoder should be preserve comprises: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and*

196. In my opinion, Hervey discloses this limitation.

197. Hervey teaches detecting codec information for new content. In GStreamer, a stream’s CAPS describes the media type of that stream. (EX1018, 2

“Caps (capabilities) are lightweight refcounted objects describing media types.”).

Hervey describes examining the CAPS of new streams at the multiqueue output:

“STREAM_START / CAPS on multiqueue source pad.” “Check if in requested/pending selection.” “Check if there is an output available to be re-used.”

“Check if decoder can accept CAPS.” (EX1009, 29.)

Re-using decoders

- STREAM_START / CAPS on multiqueue source pad
 - Checks if in requested / pending selection
 - Check if there is an output available to be re-used
 - Check if decoder can accept CAPS
 - If so, just link
 - If not, unlink, insert new decoder, link

198. Because a decoder of one particular media type can only decode audio or video streams of that same type (for example, H.264, AAC), a POSITA would understand that examining a stream’s CAPS to determine decoder compatibility detects codec information in the stream’s CAPS to determine the decoder needed to decode the stream.

- b)** *2[b]: determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.*

199. In my opinion, Hervey teaches this determination. The “Re-using decoders” slide describes checking whether the existing decoder can accept the

CAPS of the new stream: “Check if decoder can accept CAPS. If so, just link.”

(EX1009, 29.) This check, comparing the existing decoder’s capabilities against the CAPS (codec requirements) of the new stream, is the determination recited in this limitation. If the decoder’s capabilities match the CAPS, the decoder is “of the types indicated by the codec information.” A POSITA would recognize this as a disclosure of this limitation.

Re-using decoders

- **STREAM_START / CAPS on multiqueue source pad**
 - Checks if in requested / pending selection
 - Check if there is an output available to be re-used
 - Check if decoder can accept CAPS
 - If so, just link
 - If not, unlink, insert new decoder, link

3. Claim 3

- a) *The method of claim 2, further comprising preserving an entirety of the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.*

200. In my opinion, Hervey discloses this limitation.

201. Hervey expressly teaches full pipeline preservation when the CAPS check succeeds. When “decoder can accept CAPS,” the result is “just link.” Meaning, the existing decoders are preserved in place and connected directly to the

new stream. (EX1009, 29.) Hervey further teaches that, when the video and audio streams are format-compatible, the demuxer and source elements are also preserved: “Video stream remains the same, but other streams change. No need to remove/add pad for the video stream. No breakage in data stream.” (EX1009, 30.)

Demuxer handling

- Decodebin3 backwards compatible with current behaviour
- Demuxers (ex: tsdemux) can be smarter
 - Ex: Video stream remains the same, but other streams change
 - No need to remove/add pad for the video stream
 - No breakage in data stream
- Any element can add/remove streams at any time
 - Ex: [CC] parsing in video parsers

4. Claim 4

- a) *4[pre]/4[a]: The method of claim 2, further comprising: in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information,*

202. In my opinion, Hervey discloses this limitation.

203. As I discussed above with respect to claim 2, Hervey teaches examining the CAPS of each stream at the multiqueue output to determine whether the existing decoder can accept the new stream’s CAPS. (EX1009, 29.) When the existing decoder cannot accept the CAPS, *i.e.* the existing decoder is of a different type than that indicated by the codec information, Hervey teaches: “unlink, insert new

decoder, link.” (EX1009, 29.) Hervey’s decodebin3 architecture maintains separate decoder slots for audio and video streams, as shown in slide 24, which depicts distinct VDec and ADec components in the pipeline. (EX1009, 24.) Slide 26 further confirms that decoder selection occurs independently for each stream type at the multiqueue output based on that stream’s CAPS: “Only thing that (might) need to be plugged is a decoder. By default only expose/decode one stream of each type.” (EX1009, 26.)

b) *4[b]: obtaining a new audio decoder and a new video decoder based on the codec information.*

204. In my opinion, Hervey discloses this limitation.

205. Hervey teaches obtaining new decoders when the existing decoders are incompatible. Following the “unlink” step, Hervey teaches “insert new decoder.” A new decoder is instantiated and inserted into the pipeline.

Re-using decoders

- STREAM_START / CAPS on multiqueue source pad
 - Checks if in requested / pending selection
 - Check if there is an output available to be re-used
 - Check if decoder can accept CAPS
 - If so, just link
 - If not, unlink, insert new decoder, link

206. The new decoder is selected based on the CAPS of the new stream: the GstStreamCollection mechanism and GStreamer’s auto-plugging logic identify the

appropriate decoder type based on the stream's codec descriptor. (*See, e.g.*, EX1009, 20-21, 31.)

5. Claim 8

- a) *8[pre]/8[a]: The method of claim 1, further comprising: based on the first piece of content and the new piece of content not being the same content type, deconstructing the playback pipeline by releasing the source element, the demultiplexer, the audio decoder, and the video decoder; and*

207. In my opinion, Hervey discloses this limitation.

208. Hervey teaches full pipeline deconstruction when content types are incompatible through decodebin3's stream switching and decoder management logic. The “Re-using decoders” slide teaches that when the existing decoder cannot accept the new stream’s CAPS (meaning the encoding types are incompatible), decodebin3 executes “unlink, insert new decoder, link.”

Re-using decoders

- STREAM_START / CAPS on multiqueue source pad
 - Checks if in requested / pending selection
 - Check if there is an output available to be re-used
 - Check if decoder can accept CAPS
 - If so, just link
 - If not, unlink, insert new decoder, link

209. The “unlink” step releases the existing decoder from the pipeline. For cases of fundamental content-type incompatibility extending beyond the decoders to

the demuxer and source elements (such as the new content having a new container type), Hervey's "Demuxer handling" slide teaches that when streams change at the demuxer level (meaning the new content requires a different demuxer configuration) decodebin3 handles that reconfiguration by releasing and replacing the affected elements: "Any element can add/remove streams at any time." (EX1009, 30.)

Demuxer handling

- Decodebin3 backwards compatible with current behaviour
- Demuxers (ex: tsdemux) can be smarter
 - Ex: Video stream remains the same, but other streams change
 - No need to remove/add pad for the video stream
 - No breakage in data stream
- Any element can add/remove streams at any time
 - Ex: [CC] parsing in video parsers

210. Hervey further teaches through the GstParseBin mechanism that when input changes require complete reconfiguration, GstParseBin handles the full pipeline reconfiguration, releasing existing elements and acquiring new ones: "GstParseBin reconfigurable; Handles the reconfiguration; And we just get new streams." (EX1009, 32.)

Gapless playback in decodebin3

- Can we re-use more elements ?
- Just re-use elements for gapless playback
- When EOS reaches all multiqueue slot input, emit 'about-to-finish'
 - Users have a chance to change the input of decodebin3
 - Earlier than with legacy playbin (EOS on output of decodebin)
- GstParseBin reconfigurable
 - Handles the reconfiguration
 - And we just get new streams... as if it was a demuxer update

211. A POSITA would understand that in the scenario where new content has a different container and encoding format, decodebin3's reconfiguration mechanism releases all of these pipeline elements.

- b)** *8[b]: creating a new playback pipeline by acquiring other resources that are applicable to a content type of the new piece of content.*

212. In my opinion, Hervey discloses this limitation.

213. Hervey's GstParseBin reconfiguration mechanism handles rebuilding of the pipeline: "GstParseBin reconfigurable. Handles the reconfiguration. And we just get new streams." (EX1009, 32.)

Gapless playback in decodebin3

- Can we re-use more elements ?
- Just re-use elements for gapless playback
- When EOS reaches all multiqueue slot input, emit 'about-to-finish'
 - Users have a chance to change the input of decodebin3
 - Earlier than with legacy playbin (EOS on output of decodebin)
- GstParseBin reconfigurable
 - Handles the reconfiguration
 - And we just get new streams... as if it was a demuxer update

214. The phrase “we just get new streams” reflects that GstParseBin, upon reconfiguration, automatically identifies and instantiates the elements needed to process the new content, acquiring resources applicable to the new content type. Hervey further teaches through the GstParseBin description that it “recursively figures out ‘decodable’ elements needed. Demuxers, depayloaders, parsers... but not decoders” (EX1009, 25) for any given input stream, and that the output of GstParseBin feeds the multiqueue at which new decoders are selected and instantiated based on the stream’s CAPS. (EX1009, 26.)

GstParseBin

- One single input sink pad
- Recursively figures out “decodable” elements needed
 - Demuxers, depayloaders, parsers ... but not decoders
- No queueing
- Creates/Posts GstStreamCollection/GstStream if the element didn't create it
 - All pipelines can get new API support \o/
- Longer term : Reconfigurable
 - On input changes, re-use elements if possible, else switch

decodebin3

- Output of GstParseBin(s) fed to multiqueue
 - Parsed Elementary Streams
 - Multiqueue “slots” (sink + src pad) are typed (audio, video,...)
 - If available slot of proper type, re-use, else create new one
- Selection is done post-multiqueue
 - Only thing that (might) need to be plugged is a decoder
 - By default only expose/decode one stream of each type
 - But you can also expose everything...
 - Or just what you selected via GST_EVENT_SELECT_STREAMS

6. Claim 9

- a) *The method of claim 1, wherein content types include HTTP Live Streaming (HLS) format and Dynamic Adaptive Streaming over HTTP (DASH) format.*

215. In my opinion, Hervey discloses this limitation.

216. Hervey expressly identifies HLS and DASH as content types that decodebin3's stream selection architecture is designed to handle. Slide 22 identifies "Alternate HLS/DASH" as an example of hidden streams that can be activated using the GST_EVENT_SELECT_STREAMS API. (EX1009, 22.)

Selecting Streams

- GST_EVENT_SELECT_STREAMS
 - List of stream-id to be selected
- Elements can now reliably know which streams will be needed downstream
 - Avoid processing (decoding anyone ?)
 - Hidden streams to activate (Alternate HLS/DASH, switching DVB channel, ...)
- Not tied to playbin
 - Custom pipeline ? Win again

7. Claim 10

- a) *The method of claim 1, wherein the receiving the selection, transitioning to the new piece of content, determining whether the new piece of content is of the same content type as the first piece of content, and determining whether to preserve is performed by a player.*

217. In my opinion, Hervey discloses this limitation.

218. Hervey's playbin/decodebin3 stack collectively performs all of the operations recited in claim 1 and corresponds to the "player" as described in the '139 patent. Playbin is the top-level player component that manages stream listing and selection and handles content switching: "Uses uridecodebin. Output of

uridecodebin to inputselector(s) (to switch streams),” and “[a]llows stream listing/selection.” (EX1009, 14.)

PlayBin

- Uses uridecodebin
 - Output of uridecodebin to inputselector(s) (to switch streams)
 - Into playsink
- Allows stream listing/selection
- Extra-uri (for subtitle in separate URI)
 - Creates a new parallel uridecodebin
- Gapless support (about-to-finish / next-uri)
 - Prerolls another uridecodebin and switches

219. Playbin receives content selections and manages the transition to new content through its gapless and stream-switching mechanisms. (EX1009, 14; “Gapless support (about-to-finish / next-uri). Prerolls another uridecodebin and switches”). The content-type determination and pipeline preservation decision are performed by decodebin3, which operates within and under the management of playbin: “Shifted most of the logic to decodebin3 (stream selection, gapless, multi-input support).” (EX1009, 33.)

220. This passage also describes that stream selection and gapless switching logic (including the preservation determination) is part of the integrated player stack managed by playbin.

8. Claim 11

- a) *11[pre]/11[a]/11[b]: The method of claim 1, further comprising: based on the first piece of content and the new piece of content not being the same content type, releasing the source element; and creating a new playback pipeline by acquiring a different source element that is applicable to a content type of the new piece of content.*

221. In my opinion, Hervey discloses this claim for the same reasons I discussed above for claim 8. Claim 8 describes replacing the entire pipeline, including the source element, and therefore my analysis there applies to this claim.

9. Claim 12

- a) *12[pre]/12[a]: The method of claim 1, further comprising: based on the first piece of content and the new piece of content not being the same content type, releasing the demultiplexer; and creating a new playback pipeline by acquiring a different demultiplexer that is applicable to a content type of the new piece of content.*

222. In my opinion, Hervey discloses this claim for the same reasons I discussed above for claim 8. Claim 8 describes replacing the entire pipeline, including the demultiplexer, and therefore my analysis there applies to this claim.

10. Claim 14

- a) *14[pre]: A system comprising:*

223. In my opinion, for the reasons I discussed above for limitation 1[pre], Hervey discloses this limitation.

b) *14[a]: one or more hardware processors; and*

224. In my opinion, Hervey discloses this limitation.

225. Hervey expressly references the CPU as a hardware resource consumed by the media playback pipeline. The “Summary” slide identifies reducing “CPU” usage as a primary goal of decodebin3: “Avoid as much unneeded processing as possible (CPU, MEM, I/O, ...).” (EX1009, 34.)

Summary

- Avoid as much unneeded processing as possible (CPU, MEM, I/O, ...)
- Re-use as many elements as possible
- Still WIP

226. The “Decodebin2: Stream Switching” slide similarly identifies “Useless CPU/MEM/IO usage” as a problem with the prior decodebin2 architecture that decodebin3 is designed to address. (EX1009, 7.)

Decodebin2: Stream Switching

We might switch streams later on ...

We don't want the other stream to be drained completely

Else we will have a gap when switching
multiqueue throttles unlinked streams

But ... we're decoding everything :(

Useless CPU/MEM/IO usage

227. These references to CPU confirm that the playback pipeline described in Hervey executes on a CPU, a hardware processor, and that the operations performed by decodebin3's stream switching, codec detection, and pipeline reuse logic consume CPU resources. A POSITA would understand these explicit references to CPU usage as disclosing that Hervey's system operates on one or more hardware processors within the meaning of this claim.

- c) *14[b]: a storage device storing instructions that configure the one or more hardware processors to perform operations comprising:*

228. In my opinion, Hervey discloses this limitation.

229. Hervey expressly references memory (MEM) as a hardware resource consumed and managed by the decodebin pipeline: "Avoid as much unneeded processing as possible (CPU, MEM, I/O, ...)." (EX1009, 34.)

Summary

- Avoid as much unneeded processing as possible (CPU, MEM, I/O, ...)
- Re-use as many elements as possible
- Still WIP

230. The “Decodebin2: Stream Switching” slide (EX1009, 7) and the "Chain pitfalls" slide identifies "Increased memory usage (multiqueue)" and "Increased CPU usage (duplicated elements)" as specific resource problems that decodebin3's design addresses by reducing the number of instantiated pipeline elements. (EX1009, 11.)

Chain pitfalls

- New “pending” DecodeGroup
 - Increased memory usage (multiqueue)
 - Increased CPU usage (duplicated elements)
- Input and output of decodebin is no longer fully linked
 - Ex : seek event ending nowhere :(
- Want to just add/remove a stream ?
 - Still need to re-create a new bag of source pads
 - Breaks playback (switch video decoder in GOP)
 - Note: wasn't possible in 0.10 due to SEGMENT limitations

231. These references confirm that decodebin3's pipeline elements are software components that occupy memory and execute on the CPU. A POSITA would understand that software components of this kind that consume CPU and memory resources are necessarily stored as instructions on a storage device that, when executed by the CPU, configure it to perform the pipeline operations described in Hervey.

- d) *14[c]: causing playback of a first piece of content using a playback pipeline;*

232. In my opinion, for the reasons I discussed above for limitation 1[a], Hervey discloses this limitation.

- e) *14[d]: receiving a selection of a new piece of content for playback during playback of the first piece of content;*

233. In my opinion, for the reasons I discussed above for limitation 1[b], Hervey discloses this limitation.

- f) *14[e]: in response to the receiving of the selection, transitioning from playback of the first piece of content to the new piece of content;*

234. In my opinion, for the reasons I discussed above for limitation 1[c], Hervey discloses this limitation.

- g) *14[f]: determining whether the new piece of content is of a same content type as the first piece of content;*

235. In my opinion, for the reasons I discussed above for limitation 1[d], Hervey discloses or renders obvious this limitation.

- h)** *14[g]: based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder and a video decoder of the playback pipeline used to play back the first piece of content should be preserved; and*

236. In my opinion, for the reasons I discussed above for limitation 1[e],

Hervey discloses or renders obvious this limitation.

- i)** *14[h]: based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the playback pipeline while replacing the audio decoder and the video decoder.*

237. In my opinion, for the reasons I discussed above for limitation 1[f],

Hervey discloses or renders obvious this limitation.

11. Claim 15

- a)** *The system of claim 14, wherein the determining whether the audio decoder and the video decoder should be preserved comprises: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.*

238. In my opinion, for the reasons I discussed above for claim 2, Hervey discloses this limitation.

12. Claim 16

- a)** *The system of claim 15, wherein the operations further comprise: preserving an entirety of the playback pipeline in response to the audio decoder and the video*

decoder in the playback pipeline being the types indicated by the codec information.

239. In my opinion, for the reasons I discussed above for claim 3, Hervey discloses this limitation.

13. Claim 17

- a) *The system of claim 15, wherein the operations further comprise: in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information, obtaining a new audio decoder and a new video decoder based on the codec information.*

240. In my opinion, for the reasons I discussed above for claim 4, Hervey discloses this limitation.

14. Claim 18

- a) *The system of claim 14, wherein the operations further comprise: based on the first piece of content and the new piece of content not being the same content type, deconstructing the playback pipeline by releasing the source element, the demultiplexer, the audio decoder, and the video decoder; and creating a new playback pipeline by acquiring other resources that are applicable to a content type of the new piece of content.*

241. In my opinion, for the reasons I discussed above for claim 8, Hervey discloses this limitation.

15. Claim 19

- a) *19[pre]: A machine-storage medium storing instructions that when executed by one or more*

hardware processors of a machine, cause the machine to perform operations comprising:

242. In my opinion, for the reasons I discussed above in limitations 14[a] and 14[b], Hervey discloses this limitation.

b) *19[a]: causing playback of a first piece of content using a playback pipeline;*

243. In my opinion, for the reasons I discussed above in limitation 1[a], Hervey discloses this limitation.

c) *19[b]: receiving a selection of a new piece of content for playback during playback of a first piece of content;*

244. In my opinion, for the reasons I discussed above in limitation 1[b], Hervey discloses this limitation.

d) *19[c]: in response to the receiving of the selection, transitioning from playback of the first piece of content to the new piece of content;*

245. In my opinion, for the reasons I discussed above in limitation 1[c], Hervey discloses this limitation.

e) *19[d]: determining whether the new piece of content is of a same content type as the first piece of content;*

246. In my opinion, for the reasons I discussed above in limitation 1[d], Hervey discloses this limitation.

f) *19[e]: based on the new piece of content being the same content type as the first piece of content, determining whether an audio decoder of the playback*

pipeline used to play back the first piece of content should be preserved; and

247. In my opinion, for the reasons I discussed above in limitation 1[e], Hervey discloses this limitation.

g) *19[f]: based on a determination that the audio decoder and the video decoder should not be preserved, preserving a source element and a demultiplexer of the playback pipeline while replacing the audio decoder and the video decoder.*

248. In my opinion, for the reasons I discussed above in limitation 1[f], Hervey discloses this limitation.

X. Ground 5: Hervey in View of Hsu Renders Obvious Claims 6-7 and 13

A. Motivation to Combine

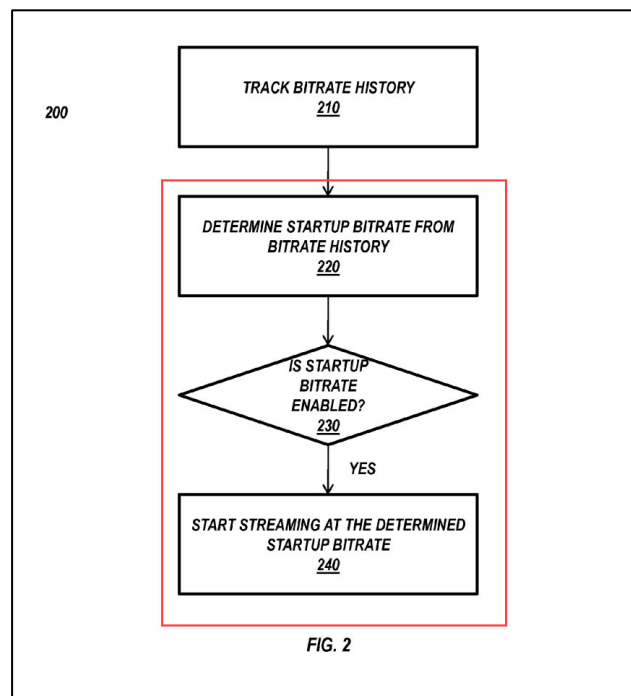
249. In my opinion, a POSITA would have been motivated to modify Hervey in view of Hsu to incorporate teachings of bitrate initialization and would have had a reasonable expectation of success. A POSITA would have been motivated by the known benefits of reducing latency and degradation of the stream quality by reusing the current bitrate for new content, as changing to the default bitrate for new content requires time and resources, and also causes the new content to initially stream at a lower quality.

250. In my proposed combination, Hervey's source element (*i.e.* sink pad) continues to manage the retrieval of content data from the content source. (EX1005, ¶[0033].) Hsu teaches initializing a new content download session at a historically

derived bitrate rather than the system default. (EX1007, 6:1-5.) A POSITA would implement Hsu's bitrate initialization teaching in the sink pad of Hervey's pipeline, configuring uridecodebin to begin downloading new content at the previously achieved bitrate. This requires no architectural modification to Hervey and produces the entirely predictable result of faster, higher-quality content transitions. Moreover, the modification is well-within the skills of an ordinary artisan, as bitrate initialization was well-known and straightforward to implement. Neither reference teaches away from the combination. A POSITA would therefore have a reasonable expectation of success.

251. Both Hervey and Hsu address the same fundamental user experience problem: the latency and quality degradation experienced by a user when switching between pieces of content on a streaming device. Hervey describes reusing pipeline elements to avoid the overhead of teardown and reconstruction when switching streams, which reduced buffering and latency to provide reduced gaps and smoother transitions between content. *See, e.g.*, EX1009 at slide 7 (“We might switch streams [but w]e don’t want the other stream to be drained completely. Else we will have a gap when switching.”); slide 11 (just adding/removing a stream “[b]reaks playback.”); and slide 23 (the new decodebin3 “[r]e-use[s] as many elements as possible,” and “[r]educe[]s buffering.”) As shown in FIG. 2 below, Hsu addresses the bitrate initialization component: eliminating the quality degradation and latency

caused by defaulting to the lowest bitrate at the start of each new session. (EX1007, 5:33-39; FIG. 2.) Hsu instead teaches using the current bitrate for a new piece of content, allowing for a seamless transition and increase in streaming quality. (e.g., EX1007, 6:1-5.) This would therefore motivate a POSITA to further improve the user experience consistent with Orr's goals.



252. Hervey and Hsu are therefore complementary solutions: Hervey operates at the pipeline and rendering layer, while Hsu operates at the download and bitrate layer. A POSITA designing a comprehensive content-switch optimization system would naturally consider and implement both and have a reasonable expectation of success.

253. A POSITA combining Hervey and Hsu would implement Hsu's bitrate initialization teaching in the source element of Hervey's pipeline. For example, by

configuring the source element to initiate the new content download at the bitrate derived from Hsu's historical tracking mechanism, rather than at the system default lowest bitrate. This integration is architecturally natural and technically straightforward, as the source element is the component that manages download sessions and bitrate selection in Hervey's pipeline.

254. In addition, both references are directed to adaptive HTTP streaming contexts. Hervey expressly addresses HLS and DASH pipelines. (EX1009, slide 22.) Meanwhile, Hsu is directed to HTTP-based adaptive bitrate streaming. (EX1007, 3:62-4:27.) A POSITA working in adaptive HTTP streaming would consult both references as directly applicable to the same technical context, and would recognize the bitrate initialization teaching of Hsu as a natural complement to the pipeline reuse teaching of Hervey.

B. Limitation-By-Limitation Analysis

- 1. Claim 6: The method of any one of claim 1, wherein the transitioning to the new piece of content comprises initiating playback of the new piece of content using a bitrate used for playback of the first piece of content to maintain the bitrate for playback of the new piece of content.**

255. In my opinion, Hsu discloses this limitation and therefore in combination with Hervey renders obvious this limitation.

256. Hsu address bitrate maintenance across content transitions. Hsu discloses that, upon initiating playback of new content, the player uses a bitrate

derived from the history of prior sessions, specifically the bitrate used during prior playback, as the startup bitrate for the new content: “The technology can use the most recent bitrate as the startup bitrate.” (EX1007, 6:1-5.) Hsu’s claim 1 recites “determining a startup bitrate based on the bitrate history, the startup bitrate comprising an initial bitrate for streaming the media content ... wherein the streaming begins at the determined startup bitrate.” The bitrate “used for playback of the first piece of content” in claim 6 of the ’139 patent corresponds directly to the “most recent bitrate” tracked by Hsu’s player, which is used to initialize the new streaming session.

257. A POSITA combining Hervey and Hsu would implement Hsu’s bitrate initialization teaching in the source element of Hervey’s pipeline. For example, by configuring the source element to initiate the new content download at the bitrate derived from Hsu’s historical tracking mechanism, rather than at the system default lowest bitrate. This integration is architecturally natural and technically straightforward, as the source element is the component that manages download sessions and bitrate selection in Hervey’s pipeline.

258. With respect to *the determining whether to preserve occurring during playback of the new piece of content using the bitrate*, this temporal relationship follows naturally from the combination of Hervey and Hsu. The ’139 patent’s FIG. 3 depicts the relevant sequence: operation 308 (“Maintain Bitrate”) precedes operation

310 (“Perform Pipeline Preservation Process”), confirming that the bitrate is maintained first and the pipeline preservation determination follows as a subsequent step performed while the new content is playing. (EX1001, 7:1-25; FIG. 3.) In the combined system, Hsu’s startup-bitrate initialization governs the initiation of the new content download session; streaming begins at the maintained bitrate at the moment the new session starts. (EX1007, 6:1-5; claim 1.) Hervey’s CAPS-based decoder compatibility check, meanwhile, occurs at the multiqueue output after the new content stream has begun flowing through the pipeline. (EX1009, slides 26, 29.) Because Hsu’s bitrate initialization necessarily precedes the arrival of stream data at the multiqueue output, the pipeline-preservation determination performed by Hervey’s CAPS check necessarily occurs during playback of the new content at the maintained bitrate. This temporal ordering is the natural and predictable consequence of implementing the two teachings together and requires no inventive step.

2. Claim 7

- a)** *7[pre]/7[a]: The method of claim 6, further comprising: during playback of the new piece of content using the bitrate, detecting whether the new piece of content can maintain playback at the bitrate; and*

259. In my opinion, Hsu discloses this limitation and therefore in combination with Hervey renders obvious this limitation.

260. Hsu teaches continuous monitoring of whether the current bitrate can be maintained during playback. Hsu's adaptive bitrate player "determin[es] bitrate values used at a plurality of different times for providing the streaming media content" during playback, tracking whether the current bitrate is sustainable given available network bandwidth. (EX1009, claim 1). This continuous monitoring is the mechanism by which the adaptive bitrate system detects that the maintained bitrate is or is not achievable for the new content. A POSITA would understand this as disclosing "detecting whether the new piece of content can maintain playback at the bitrate" as recited in this limitation.

b) *7[b]: in response to the detecting that the new piece of content cannot continue playback at the bitrate, adjusting the bitrate.*

261. In my opinion, Hsu discloses this limitation and therefore in combination with Hervey renders obvious this limitation.

262. Hsu teaches downward bitrate adjustment when the current bitrate cannot be sustained. Hsu describes a capping mechanism that accounts for users moving between higher and lower bandwidth environments, and the standard adaptive bitrate mechanism taught by Hsu lowers the bitrate when download throughput is insufficient to sustain the current level: the player "adjusts the quality of a video stream accordingly" when bandwidth weakens. (EX1009, 3:66-4:5.) Hsu's claim 7 further describes the player "recording the bitrate on a per session

basis,” which is the mechanism by which the system detects and responds to bitrate sustainability. A POSITA combining Hervey and Hsu would implement this downward adjustment in the source element of Hervey’s pipeline, consistent with standard adaptive bitrate behavior.

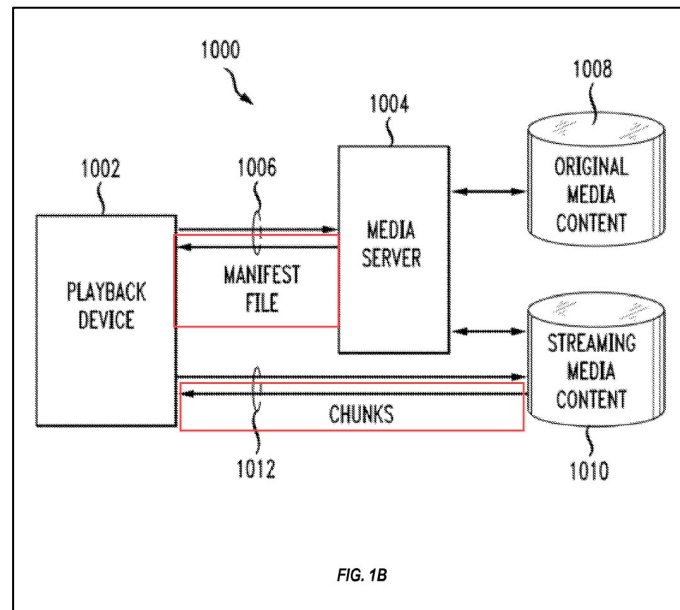
3. Claim 13

- a)** *The method of claim 1, wherein the source element initiates the playback of the new piece of content using a same bitrate as the first piece of content and adjusts the bitrate in response to detecting that the new piece of content cannot continue playback at the bitrate.*

263. I note that this claim combines the requirements of claims 6 and 7 and attributes them specifically to the source element. Thus, for the same reasons I discussed above for claims 6 and 7, Hsu discloses this limitation, and it would have been obvious to incorporate these teachings into Hervey’s source element.

264. Hsu’s FIG. 1B shows the playback device 1002 as the component that requests and receives streaming media content from the media server 1004. Specifically, the playback device 1002 receives the manifest file 1006 from the media server 1004 (the file that specifies the available bitrates and segment URLs for the content) and receives content chunks 1012 directly. (EX1007, FIG. 1B.) The manifest file is the entry point for any adaptive bitrate streaming session: it is what the playback device consults to know what bitrates are available and where to obtain each segment. The playback device’s receipt of the manifest and subsequent

download of chunks is precisely the content data acquisition function performed by the source element in a GStreamer-based or similar adaptive streaming pipeline.



265. Hsu describes the playback device as the component that controls bitrate selection. Hsu teaches that “By default ... each playback device starts playing at the lowest bitrate” and “the playback device then switches to a higher bitrate as conditions allow.” (EX1007, 5:33-37.) As described above, Hsu also describes that “[t]he technology can use the most recent bitrate as the startup bitrate.” (EX1009, 6:1-5; *see also*, claim 1) and “adjusts the quality of a video stream accordingly” when bandwidth weakens. (EX1009, 3:66-4:5.) That is, Hsu describes the playback device as determining a startup bitrate from its bitrate history and adjusts the bitrate as needed.

266. As noted with respect to claims 6 and 7, a POSITA would implement Hsu's bitrate initialization and adjustment teaching in Hervey's source element. The source element initiates the new session at the historically derived bitrate (e.g., EX1009, 6:1-5, claim 1, FIG. 1B) and adjusts downward when the bitrate cannot be maintained. (e.g., EX1009, 3:66-4:5.) This claim is thus rendered obvious by the combination of Hervey and Hsu for the same reasons set forth above.

XI. Ground 3: Hervey in View of Lim Renders Obvious Claims 5

A. Motivation to Combine

267. A POSITA would have been motivated to integrate Lim's prefetching of metadata into Hervey, and would have had a reasonable expectation of success in doing so.

268. A POSITA would be motivated by the fact that Lim's prefetched metadata reduces latency by supplying in advance the stream and codec information needed when Hervey switches to new HLS or DASH content. Rather than obtaining this information at the time of switching to HLS or DASH content, a POSITA would have integrated Lim's prefetched metadata to obtain the stream and codec information before switching, thereby reducing the latency typically needed when switching to new content and allowing for a seamless transition to video. This is the straightforward application of prefetching, a well-known technique for reducing

access latency, to the specific bottleneck of on-demand content-type retrieval in Hervey's stream-switch system.

269. Hervey and Lim are both motivated by the same user-facing problem: the delay experienced when a user switches from one piece of content to another on a playback device. Hervey describes reusing pipeline elements to avoid the overhead of teardown and reconstruction when switching streams, which reduces buffering and latency to provide reduced gaps and smoother transitions between content. *See, e.g.,* EX1009 at slide 7 (“We might switch streams [but w]e don’t want the other stream to be drained completely. Else we will have a gap when switching.”); slide 11 (just adding/removing a stream “[b]reaks playback.”); and slide 23 (the new decodebin3 “[r]e-use[s] as many elements as possible,” and “[r]educe[]s buffering.”) Lim is similarly directed to reducing the delay between a user's content selection and the commencement of playback: “when the user changes the content source (e.g., radio stations, video stations, etc.), the mobile device starts playing without reduced buffering time.” (EX1008, 4:1-7.) Both references thus share the same design objective—minimizing perceived latency at content switch time—and a POSITA working on that problem would naturally consider both as relevant to the same design challenge, and be motivated by Lim’s teachings to further Hervey’s goals.

270. Hervey identifies a limitation in its stream selection architecture that Lim’s prefetching teaching directly addresses. The “Pitfalls of Stream Selection”

slide (EX1009, slide 15) identifies the potential absence of pads (*e.g.*, HLS/DASH) as a problem. That is, when switching to new HLS or DASH content, stream and codec information may not yet be available from source pads at the time of the switch, because the new stream has not yet been opened. Here, the prefetching the metadata for candidate next-play content, as taught by Lim, provides direct value. If the codec and stream information for the new content were already locally available from a prefetch operation performed while prior content was playing, Hervey's stream selection logic could make its compatibility determination immediately without waiting for source pads to be created. (*See, e.g.*, EX1009, slide 27 ("Startup example: ... Do a pre-emptive decision of which streams to use.") A POSITA reading both references together would immediately recognize that Lim's prefetching architecture fills an information gap that Hervey itself identifies.

271. Hervey and Lim are analogous art as they are both directed to processing video streams.

272. A POSITA would have reasonably expected the combination to succeed. Lim provides detailed disclosures regarding implementation of prefetching, and it would have been well-within the skill of an ordinary artisan to implement such functionality into the software or hardware of Hervey's decodebin3. Such a modification would only require ordinary programming knowledge.

B. Limitation-By-Limitation Analysis

1. Claim 5

- a) *The method of claim 2, wherein the detecting the codec information for the new piece of content comprises examining prefetched metadata for the new piece of content.*

273. In my opinion, Hervey in view of Lim renders obvious this limitation.

274. Claim 5 adds to the codec detection requirement of claim 2 the specific limitation that the codec information is obtained by examining metadata that was prefetched, such as retrieved and stored locally before the user selects the content.

275. As I discussed above for claim 2, Hervey teaches detecting of codec and format information. Specifically, Hervey teaches examining the CAPS of the new stream to determine decoder compatibility. (EX1009, slide 29.) Hervey teaches obtaining this CAPS information at the time the stream data arrives at the multiqueue output, rather than from prefetched metadata.

276. Lim teaches prefetching metadata for candidate next-play content during playback of current content. Lim's claim 1 recites "prefetching, by the electronic device, content prior to attempting to play at least one station on the electronic device," and claim 2 expressly specifies that "prefetching content comprises prefetching preview content clips, cover art, and metadata for all on-deck content for the plurality of stations." (EX1008, claims 1-2.) Lim's specification confirms this teaching: Lim describes "the client 305 efficiently prefetch[ing] the

preview clips, cover art, and metadata for all the on-deck songs of all the stations” using a cold start function while current content plays. (EX1008, 9:6-10.) Lim further describes the prefetch manager module 450 implementing a “Prefetch Song” process that “downloads a song’s preview clip and cover art and stores them in the prefetch inventory 455 and image cache 470,” and a “CheckOnDeckPrefetch” process that triggers a prefetch of the next on-deck song's data while the current song still has sufficient buffer remaining — *i.e.*, while current content is still playing. (EX1008, 12:44-55.) Lim further describes the metadata prefetched for each song as including the “song URL and cover art URL,” and notes that the aggregate station binary downloaded at cold start contains “all of the stations’ first song metadata, preview clip, and cover-art.” (EX1008, 10:27-41; 12:37-42.)

277. A POSITA combining Hervey and Lim would implement Hervey’s codec detection step of claim 2 using metadata prefetched per Lim. Rather than waiting for stream data to arrive at the multiqueue output after the user has made a selection, the system would read the codec information from the locally cached metadata fetched by Lim’s prefetch manager during playback of the prior content.

278. This is a straightforward integration. Lim’s prefetched metadata provides the input, and Hervey’s CAPS check provides the processing and produces the predictable result of allowing the pipeline reuse determination to be made immediately upon receiving the user’s selection.

XII. Conclusion

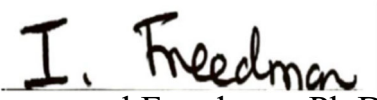
279. In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

* * *

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

Respectfully submitted,

Date: May 22, 2026


Immanuel Freedman, Ph.D.