

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

PINTEREST, INC.

Petitioners,

v.

OPENTV, INC.

Patent Owner.

Case No. IPR2026-00366

U.S. Patent No. 11,184,652

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 11,184,652**

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. MANDATORY NOTICES.....	1
A. Real Party-in-Interest (37 C.F.R. § 42.8(b)(1))	1
B. Related Matters (37 C.F.R. § 42.8(b)(2)).....	1
1. Related Patent Office Proceedings.....	1
2. Related Litigation.....	1
C. Lead and Backup Counsel and Service Information (37 C.F.R. §§ 42.8(b)(3)-(4)).....	1
D. Payment of Fees (37 C.F.R. § 42.15(a))	2
E. Requirements For <i>Inter Partes</i> Review (37 C.F.R. §§ 42.101(A)-(C), 42.104(A), and 42.018).....	2
III. STATEMENT OF RELIEF REQUESTED AND IDENTIFICATION OF CHALLENGE (37 C.F.R. § 42.104(B)).....	3
IV. BACKGROUND	3
A. '652 Patent	3
1. Earliest Priority Date.....	3
2. Specification	3
3. Prosecution History.....	6
B. Summary of the Primary References	8
1. Orr (EX1005).....	8
2. Hsu (EX1007)	8
3. Hervey (EX1009).....	9
V. PERSON OF ORDINARY SKILL.....	18
VI. CLAIM CONSTRUCTION.....	19
VII. GROUND 1: ORR IN COMBINATION WITH HSU RENDERS CLAIMS 1-20 OBVIOUS.....	19
A. Overview and Motivation to Combine	19
B. Limitation-By-Limitation Analysis	23
1. Claim 1	23
2. Claim 2.....	32

Petition for *Inter Partes* Review
U.S. Patent No. 11,184,652

3.	Claim 3.....	41
4.	Claim 4.....	43
5.	Claim 5.....	44
6.	Claim 6.....	46
7.	Claim 7.....	48
8.	Claim 8.....	49
9.	Claim 9.....	50
10.	Claim 10.....	50
11.	Claim 11.....	51
12.	Claim 12.....	52
13.	Claim 13.....	53
14.	Claim 14.....	53
15.	Claim 15.....	54
16.	Claim 16: The method of claim 1, wherein: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated; and the adjusting comprises increasing the first bitrate.....	54
17.	Claim 17: The method of claim 1, wherein: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking longer than anticipated; and the adjusting comprises decreasing the first bitrate.....	55
18.	Claim 18: The system of claim 6, wherein: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated; and the adjusting comprises increasing the first bitrate.....	56
19.	Claim 19: The system of claim 6, wherein the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download is taking longer than anticipated; and the adjusting comprises decreasing the first bitrate.....	56
20.	Claim 20.....	57
VIII.	GROUND 2: HERVEY IN COMBINATION WITH HSU RENDERS CLAIMS 1-20 OBVIOUS	57
A.	Overview and Motivation to Combine	57
B.	Limitation-By-Limitation Analysis	61

Petition for *Inter Partes* Review
U.S. Patent No. 11,184,652

1.	Claim 1	61
2.	Claim 2	67
3.	Claim 3	73
4.	Claim 4	75
5.	Claim 5	76
6.	Claim 6	78
7.	Claim 7	83
8.	Claim 8	83
9.	Claim 9	84
10.	Claim 10	84
11.	Claim 11	84
12.	Claim 12	85
13.	Claim 13	86
14.	Claim 14	86
15.	Claim 15	86
16.	Claim 16	87
17.	Claim 17	87
1.	Claim 18	88
2.	Claim 19	88
3.	Claim 20	88
IX.	CONCLUSION	89

EXHIBIT LIST

Exhibit	Description
EX1001	U.S. Patent 11,184,652 (“’652 Patent”)
EX1002	Declaration of Immanuel Freedman, Ph.D.
EX1003	Declaration of Edward Hervey
EX1004	File History of U.S. Patent 11,184,652
EX1005	U.S. Patent Application Publication 2009/0132725 (“Orr”)
EX1006	Intentionally Omitted
EX1007	U.S. Patent 8,516,144 (“Hsu”)
EX1008	Intentionally Omitted
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	U.S. Pat. Pub. No. 2008/0195745A1 (“Bowra”)
EX1020	Curriculum Vitae of Immanuel Freedman, Ph.D.

LIST OF CHALLENGED CLAIMS

Claim	
1[pre]	A method comprising:
1[a]	causing playback, by a digital receiver, of a first piece of content at a first bitrate, the first bitrate being a bitrate adapted for playback of the first piece of content;
1[b]	receiving, by the digital receiver, a selection of a new piece of content for playback during the playback of the first piece of content;
1[c]	in response to the receiving of the selection, transitioning from ongoing playback of the first piece of content to the new piece of content by initiating playback of the new piece of content using the first bitrate used for playback of the first piece of content;
1[d]	during playback of the new piece of content using the first bitrate, detecting whether the new piece of content can maintain playback at the first bitrate; and
1[e]	in response to the detecting that the new piece of content cannot continue playback at the first bitrate, adjusting the first bitrate.
2[pre]	The method of claim 1, further comprising:
2[a]	determining whether the new piece of content is of a same content type as the first piece of content; and
2[b]	based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content, the preserving of at least the portion of the playback pipeline comprising preserving one or more of a source element, a demultiplexer, an audio decoder, or a video decoder.
3[pre]	The method of claim 2, wherein the determining whether to preserve comprises:
3[a]	detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and
3[b]	determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.
4	The method of claim 3, further comprising preserving the entire playback pipeline in response to the audio decoder and the video

Claim	
	decoder in the playback pipeline being the types indicated by the codec information.
5[pre]	The method of claim 3, further comprising:
5[a]	releasing the audio decoder and the video decoder in the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information; and
5[b]	obtaining a new audio decoder and a new video decoder based on the codec information.
6[pre]	A system comprising:
6[a]	one or more hardware processors; and
6[b]	a storage device storing instructions that configure the one or more hardware processors to perform operations comprising:
6[c]	causing playback, by a digital receiver, of a first piece of content at a first bitrate, the first bitrate being a bitrate adapted for playback of the first piece of content;
6[d]	receiving, by the digital receiver, a selection of a new piece of content for playback during the playback of the first piece of content;
6[e]	in response to the receiving of the selection, transitioning from ongoing playback of the first piece of content to the new piece of content by initiating playback of the new piece of content using the first bitrate used for playback of the first piece of content;
6[f]	during playback of the new piece of content using the first bitrate, detecting whether the new piece of content can maintain playback at the first bitrate; and
6[g]	in response to the detecting that the new piece of content cannot continue playback at the first bitrate, adjusting the first bitrate.
7[pre]	The system of claim 6, wherein the operations further comprise:
7[a]	determining whether the new piece of content is of a same content type as the first piece of content; and
7[b]	based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content, the preserving of at least the portion of the playback pipeline comprising preserving one or more of a source element, a demultiplexer, an audio decoder, or a video decoder.

Claim	
8 <pre></pre>	The system of claim 7, wherein the determining whether to preserve comprises:
8[a]	detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and
8[b]	determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.
9	The system of claim 8, wherein the operations further comprise preserving the entire playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.
10 <pre></pre>	The system of claim 8, wherein the operations further comprise:
10[a]	releasing the audio decoder and the video decoder in the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information; and
10[b]	obtaining a new audio decoder and a new video decoder based on the codec information.
11 <pre></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:
11[a]	causing playback, by a digital receiver, of a first piece of content, the first bitrate being a bitrate adapted for playback of the first piece of content;
11[b]	receiving, by the digital receiver, a selection of a new piece of content for playback during the playback of the first piece of content; and
11[c]	in response to the receiving of the selection, transitioning from ongoing playback of the first piece of content to the new piece of content by initiating playback of the new piece of content using the first bitrate used for playback of the first piece of content;
11[d]	during playback of the new piece of content using the first bitrate, detecting whether the new piece of content can maintain playback at the first bitrate; and
11[e]	in response to the detecting that the new piece of content cannot continue playback at the first bitrate, adjusting the first bitrate.

Claim	
12[pre]	The machine storage medium of claim 11, wherein the operations further comprise:
12[a]	determining whether the new piece of content is of a same content type as the first piece of content; and
12[b]	based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to playback the first piece of content, the preserving of at least the portion of the playback pipeline comprising preserving one or more of a source element, a demultiplexer, an audio decoder, or a video decoder.
13[pre]	The machine storage medium of claim 12, wherein the determining whether to preserve comprises:
13[a]	detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and
13[b]	determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.
14	The machine storage medium of claim 13, wherein the operations further comprise preserving the entire playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.
15[pre]	The machine storage medium of claim 13, wherein the operations further comprise:
15[a]	releasing the audio decoder and the video decoder in the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information; and
15[b]	obtaining a new audio decoder and a new video decoder based on the codec information.
16[pre]	The method of claim 1, wherein:
16[a]	the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated; and
16[b]	the adjusting comprises increasing the first bitrate.

Claim	
17[pre]	The method of claim 1, wherein:
17[a]	the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking longer than anticipated; and
17[b]	the adjusting comprises decreasing the first bitrate.
18[pre]	The system of claim 6, wherein:
18[a]	the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated; and
18[b]	the adjusting comprises increasing the first bitrate.
19[pre]	The system of claim 6, wherein:
19[a]	the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download is taking longer than anticipated; and
19[b]	the adjusting comprises decreasing the first bitrate.
20[pre]	The machine storage medium of claim 11 wherein:
20[a]	the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated and the adjusting comprises increasing the first bitrate; or
20[b]	the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download is taking longer than anticipated and the adjusting comprises decreasing the first bitrate.

I. INTRODUCTION

Petitioners Pinterest (“Petitioner”) requests *Inter Partes* Review of claims 1-20 of U.S. Patent No. 11,184,652 (“’652 patent”) assigned to OpenTv, Inc. (“Patent Owner”). The ’652 patent is directed to “[s]ystems and methods for optimizing a content change process” in which a “digital receiver receives a 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.) As demonstrated below, adaptive bitrate streaming and playback pipeline preservation was ubiquitous in the prior art, and the ’652 patent claims no non-obvious features or implementation.

II. MANDATORY NOTICES

A. Real Party-in-Interest (37 C.F.R. § 42.8(b)(1))

The real parties-in-interest for Petitioner are Pinterest, Inc.

B. Related Matters (37 C.F.R. § 42.8(b)(2))

1. Related Patent Office Proceedings

The ’652 patent is in the same family as U.S. Patent Nos. 11,825,139 and 12,231,703. Petitioner has filed IPR petitions against each of these patents.

2. Related Litigation

There is presently no related litigation.

C. Lead and Backup Counsel and Service Information (37 C.F.R. §§ 42.8(b)(3)-(4))

Electronic service may be made on the email addresses identified below and in the accompanying Power of Attorney.

LEAD COUNSEL	BACK-UP COUNSEL
James M. Glass (Reg. No. 46,729) jimglass@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 51 Madison Avenue, 22 nd Floor New York, NY 10010 Tel: (212) 849-7000	Quincy Lu (Reg. No. 76,954) quincylu@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 1109 First Avenue, Suite 210 Seattle, WA 98101 Tel: (206) 905-7000 Ron Hagiz (<i>pro hac vice</i> forthcoming) ronhagiz@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 295 5 th Avenue, 9th Floor New York, NY 10016 Tel: (212) 849-7000

D. Payment of Fees (37 C.F.R. § 42.15(a))

The Office is authorized to charge the fee required for this Petition (and any additional fees) to Deposit Account No. 50-5708.

E. Requirements For *Inter Partes* Review (37 C.F.R. §§ 42.101(A)-(C), 42.104(A), and 42.018)

Petitioner certifies that the '652 patent is available for *inter partes* review, that Petitioner is not barred or estopped from challenging the '652 claims on the grounds

identified herein, and that the prohibitions of 35 U.S.C. §§ 315(a)-(b) are inapplicable.

III. STATEMENT OF RELIEF REQUESTED AND IDENTIFICATION OF CHALLENGE (37 C.F.R. § 42.104(B))

Petitioner respectfully requests that *inter partes* review of claims 1-20 (“Challenged Claims”) on the following grounds:

Ground	Basis	References	Claims
1	§103	Orr in combination with Hsu	1-20
2	§103	Hervey in combination with Hsu	1-20

IV. BACKGROUND

A. ’652 Patent

1. Earliest Priority Date

The ’652 patent’s earliest priority date is September 8, 2017 via Indian Application Series No. 201741031917 (EX1001.) (EX1002, ¶¶23-25.)

2. Specification

The ’652 patent is directed to “[s]ystems and methods for optimizing a content change process” in which a “digital receiver receives a 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.) (EX1002, ¶26.)

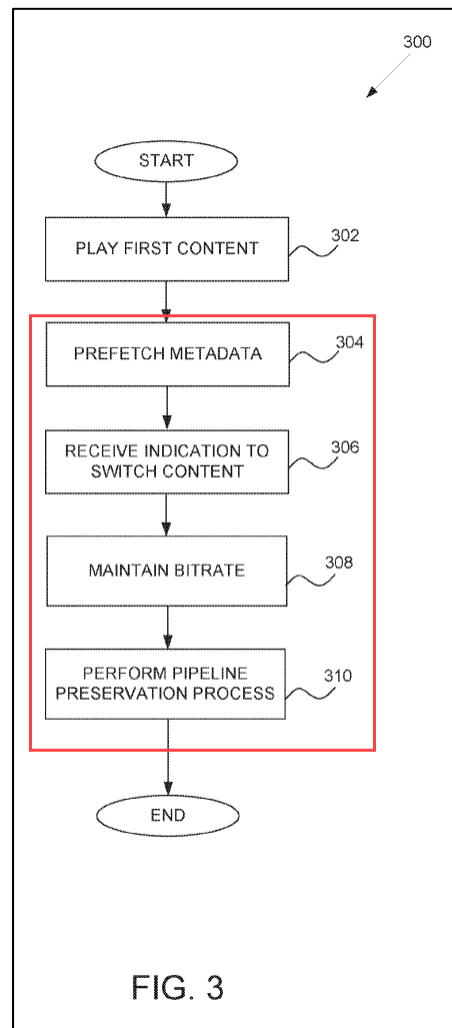
The patent identifies two conventional problems that arise at the moment of a content switch. First, with respect to bitrate, “every time a user switches from one content to the next, the playback will start with the lowest bitrate and slowly increase or improve according the capability of the network,” resulting in initially grainy video. (EX1001, 2:39-41.) 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, 2:53-55.) (EX1002, ¶27.)

The alleged solution addresses the bitrate problem through a source element that “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, 5:20-22.) 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. (EX1002, ¶28.)

The pipeline preservation problem is addressed through a two-step analysis. 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, 8:16–19.) (EX1002, ¶29.)

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, 8:24-30.) If the content types do not match at all, all pipeline resources are released and a new pipeline is constructed from scratch. (EX1002, ¶30.) This process is shown, for example, in Fig. 3:



3. Prosecution History

The '652 patent was filed on August 14, 2018 as application 16/645,598.

On January 7, 2021 an office action issued rejecting pending claims 11-15 and 20 under 35 U.S.C. § 101 as not being within one of the four statutory categories of invention. Claims 1-15 were rejected as being unpatentable under 35 U.S.C. § 103

as being unpatentable over Orr US 2011/0296047¹ in view of Shintani US 2017/0244924. Claims 16-20 were rejected as being unpatentable over Orr US 2011/0296047, Shintani, and Melnyk US 2009/0254657. (EX1004, January 7, 2021, Office Action.) (EX1002, ¶32.)

On April 7, 2021, applicant amended the claims and filed a response to the January office action (EX1004, April 7, 2021, Amendment). Specifically, applicant added the limitations that the first bitrate is “adapted for playback of the first piece of content,” and argued that Orr 2011/0296047 performs seamless media playback by converting the original media to a standard bit rate, instead of using the original bit rate. *Id.* The applicant also added the limitation that the transition occur during playback, arguing that Orr 2011/0296047 transitions to new media after the first media’s completion. *Id.* Finally, the applicant added the limitation of determining whether the new content can use the first bitrate, arguing that Shintani uses codec detection, not bitrate matching. *Id.* The claims were then allowed. (EX1004, July 21, 2021, Notice of Allowance.) (EX1002, ¶33.)

¹ Orr US 2011/0296047 is not related to the Orr reference used in the grounds of challenge.

B. Summary of the Primary References

1. Orr (EX1005)

Orr was not before the examiner during prosecution of the '652 patent, and is prior art under 35 U.S.C. § 102(a)(1).

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].)

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].) (EX1002, ¶¶41-42.)

2. Hsu (EX1007)

Hsu was not before the examiner during prosecution of the '652 patent. It is prior art under 35 U.S.C. § 102(a)(1).

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 '652 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.) (EX1002, ¶43.)

3. Hervey (EX1009)

Hervey is a PowerPoint document that was publicly presented by Edward Hervey at the GStreamer Conference 2015 on October 8th, 2015 in Dublin, Ireland. It is prior art under 35 U.S.C. § 102(a)(1). (EX1002, ¶¶44-47.) Hervey was not before the examiner during prosecution of the '139 patent.

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.)

a) Hervey is Authentic

Hervey (EX1009) was presented on October 8th, 2015 at the GStreamer Conference 2015, which was held on October 8-9th in Dublin Ireland at the Convention Center Dublin. Hervey is a true and correct copy that was presented at

that conference. According to Mr. Hervey, the author of the presentation, Exhibit 2009 is a true and authentic copy of the document he presented in 2015. EX1003.

In addition, Ms. Ching of the Internet Archive confirms that Exhibit 1009 is an authentic copy of the Hervey presentation made publicly available on the Internet by February 26, 2017. EX1010, 1.

b) Hervey Qualifies as a Printed Publication

“The statutory phrase ‘printed publication’ has been interpreted to mean that before the critical date the reference must have been sufficiently accessible to the public interested in the art; dissemination and public accessibility are the keys to the legal determination whether a prior art reference was ‘published.’” *In re Klopfenstein*, 380 F.3d 1345, 1348 (Fed. Cir. 2004). As explained below, Hervey is a printed publication because, *first*, it was publicly available on the Internet by February 26, 2017, and, *second*, it was presented in 2015 at the GStreamer Conference.

(1) *Hervey Was Publicly Available On the Internet By February 26, 2017*

Hervey was publicly accessible and available on the Internet by February 26, 2017, when the Internet Archive’s Wayback Machine archived the document by archiving the publicly available link to the presentation.

According to Ms. Ching, the Wayback Machine archives webpages “by use

of web archiving software that automatically stores copies of files available via the Internet, each file preserved as it existed at a particular point in time.” (EX1010, ¶4.) Each Internet Archive URL contains the date and time when the webpage was archived. (*Id.*, ¶5.) Hervey was and is accessible and available on the Internet Archive via the link <https://web.archive.org/web/20170226222859/https://gstreamer.freedesktop.org/data/events/gstreamer-conference/2015/Edward%20Hervey%20-%20decodebin3.pdf>. Accordingly, per Ms. Ching, Hervey was publicly available by February 26, 2017, when the Wayback Machine archived the link.

Dr. Freedman, who has over 30 years’ experience with video imaging and compression and authored the “Video Compression” chapter for the *Internet Encyclopedia*, is familiar with the Wayback Machine and agrees with Ms. Ching that the Internet Archive exhibits are archives of publicly accessible and available webpages at the time they were archived as indicated in the exhibits’ URL, and that those exhibits are true and correct copies. EX1002, ¶¶48-49. Moreover, the Federal circuit has found that the Wayback Machine is inherently reliable and “highly persuasive” on the issue of public accessibility. *Valve Corp. v. Ironburg Inventions, Ltd.*, 8 F.4th 1364, 1374-75 (Fed. Cir. 2021).

(2) *Hervey Was Presented to the Public On October 8, 2015*

Hervey is a printed publication because, according to the author, Mr. Hervey, it was presented to the public on October 8, 2015 at the 2015 GStreamer Conference with no obligation of confidentiality. Numerous websites that were contemporaneously available confirm that the 2015 GStreamer Conference took place in 2015 and was open to the public. According to Ms. Ching, these websites were archived by the Wayback Machine at around the same time the GStreamer Conference took place. EX1011-EX1014; EX1010.

The GStreamer Conference is a well-known annual technical conference attended by practitioners in the multimedia software and embedded systems fields, and its proceedings are routinely consulted by persons of ordinary skill in the art.

The GStreamer Conference indicates that “[a]nyone interested in open source cross-platform multimedia is encouraged to join us at the GStreamer Conference 2015.” The Conference was open to “hobbyist[s],” “student[s],” and those who are “using or looking to use GStreamer in their products.” In addition, registration was open to the public through to the conference. (EX1011.)

Who should attend?

Anyone interested in open source cross-platform multimedia is encouraged to join us at the GStreamer Conference 2015. Whether you are a hobbyist or a student interested in the field of multimedia, or working for a company using or looking to use GStreamer in their products, this is the perfect event for you to network at and to learn more about the latest technological breakthroughs in and around GStreamer. The talks will target both developers and managers.

Registration

Registration is now open and will stay open until the conference.

[Sign up for the GStreamer Conference here](#)

There will be separate registration options for both professionals and for students and hobbyists:

Professional	275 USD incl. lunch (until 31 July)
Professional	385 USD incl. lunch (from 1 August)
Student/hobbyist	100 USD incl. lunch

The registration fee includes continental breakfast and lunch at the venue on both conference days.

For those uncertain about whether to register as a professional or hobbyist, the general rule is that if your company pays for you or sponsors you, you are a professional, and if you personally pay you are a student or hobbyist. If in doubt drop us a mail or just register as hobbyist. The important thing is that you'll make it to the conference!

(EX1011; captured January 9, 2016.)

The GStreamer Conference 2015 Schedule of events shows that “Decodebin3, or dealing with modern playback use-cases,” was scheduled to be presented by Edward Hervey on October 8, 2015 at 11:20am. The below snapshot of the Schedule was captured by the Internet Archive on October 18, 2016.) (EX1012.)

Petition for *Inter Partes* Review
U.S. Patent No. 11,184,652

GStreamer Conference 2015 Schedule			
Dublin, Ireland, 8-9 October 2015			
Back to conference main page			
Schedule for Thursday 8 October 2015			
Schedule for Friday 9 October 2015			
Thursday 8th of October 2015			
	Liffey Meeting Room 2a	Liffey Meeting Room 2b	
08.30	Registration and Breakfast Buffet		08.30
09.00-09.10	Opening and welcome to the 2015 GStreamer Conference Liffey Meeting Room 2		09.00-09.10
09.10-09.40	Keynote GStreamer State of the Union Tim-Philipp Müller, Centricular Liffey Meeting Room 2		09.10-09.40
09.40-10.00	mini break		09.40-10.00
10.00-10.30	Tiled Streaming of UHD video in real-time Arjen Veenhuizen, TNO	Kurento Media Server: experiences bringing GStreamer capabilities to WWW developers José Antonio Santos, Kurento	10.00-10.30
10.35-10.55	GStreamer-VAAP: Hardware-accelerated encoding and decoding on Intel hardware Victor M. Jaquez L., Igalia	ogv.js: bringing open codecs to Safari and IE with emscripten Brion Vibber, Wikimedia	10.35-10.55
10.55-11.20	Coffee Break		10.55-11.20
11.20-12.05	Decodebin3, or dealing with modern playback use-cases Edward Hervey (bilboed), Centricular	Chromium: a new media Backend based on GStreamer Julien Iserce, Samsung	11.20-12.05
12.10-12.30	GstPlayer - A simple cross-platform API for all your media playback needs Sebastian Dröge (slomo), Centricular	GStreamer and WebKit Philippe Normand (philn), Igalia	12.10-12.30
12.30-13.30	Lunch Break: Buffet Lunch for GStreamer Conference attendees		12.30-13.30
13.30-14.00	Going Live with ToolsOnAir's GStreamer-based broadcast mixing architecture Heinrich Fink, ToolsOnAir	The HeliosTv Distributed DVB stack Romain Picard, Soft@Home	13.30-14.00
14.05-14.25	How to Contribute to GStreamer Luis de Bethencourt (luisbg), Samsung	Stereoscopic 3D Video in GStreamer Redux Jan Schmidt (thaytan), Centricular	14.05-14.25
14.25-14.45	mini break		14.25-14.45
14.45-15.15	Faster transcoding with GStreamer in the cloud Thiago Sousa Santos, Samsung	Colorspaces and HDMI Hans Verkuil, Cisco	14.45-15.15
15.15-15.40	Coffee Break		15.15-15.40
15.40-16.10	Demystifying the allocation query Nicolas Dufresne (stormer), Collabora	Video Filters and their Applications Sanjay Narasimha Murthy, Samsung	15.40-16.10
16.15-16.45	Camera Sharing and Sandboxing with Pinos Wim Taymans (wtay), RedHat	Hardware-accelerated multimedia on TI's Jacinto 6 SoC Pooja Prajod, Texas Instruments	16.15-16.45
16.45-17.05	mini break		16.45-17.05
17.05-18.15	Lightning Talks - part 1 Liffey Meeting Room 2		17.05-18.15
18.15	End of Day		18.15
	Social Event: Drinks and Food at the Toners Pub , 139 Lower Baggot Street, Dublin 2, D02 N231		

(EX1012; captured October 18, 2016.)

The Talks and Speakers page (EX1013) for the 2015 GStreamer Conference lists Edward Hervey's presentation.

Decodebin3, or dealing with modern playback use-cases, Edward Hervey (bilboed), Centricular

decodebin2 (decodebin in 1.x) is 10 years old. While it did serve its purpose, there are a number of limitations in regards to handling modern use-cases, features that are in playbin that should be present in decodebin, non-optimal memory usage, and so forth. This talk might include apologies from the original decodebin2 author *cough*.

After looking at the current design and limitation of stream handling in general, and decodebin2 in particular, we will look at a proposal for a new way of handling stream listing, stream selection, how this help with having predictable behaviour and drop all 0.10-ism that still remain in decodebin2.

The new design will allow reducing cpu/memory usage by only using the decoders actually needed, re-use decoders when switching streams (where possible), and better deal with adaptive streaming and dynamic MPEG-TS use-cases in general, amongst other things.

Edward Hervey has been contributing for over 12 years to GStreamer, ending up there after starting the PiTiVi video editor and then maintaining various components over the years. After having started Collabora Multimedia in 2007, attempting to go on sabbatical, and doing various freelancing, Edward Hervey is currently a part-time consultant for Centricular.

(EX1013; captured January 25, 2016.)

The GStreamer Conference page states that “[s]lides for some of the talks and the lightning talks can be found in the GStreamer Conference 2015 slides folder.” (EX1011.) Below is a list of the presentations presented at the 2015 Conference. Edward Hervey’s presentation is highlighted in the screen capture below. As shown, the presentation was last modified on October 9, 2015 (the day after his presentation). (EX1014.)

Name	Last modified	Size	Description
Parent Directory	-	-	-
Andres Colubri - Interactive video playback and capture in the Processing Language via GStreamer.pdf	13-Oct-2015 11:09	7.6M	
Balázs Kreith - Multi-path RTP in GStreamer.pdf	29-Oct-2015 05:49	2.1M	
Brion Vibber - ogvjs.pdf	07-Oct-2015 22:21	1.2M	
Dimitrios Katsaros - Hyperspectral Imagery in GStreamer.pdf	29-Oct-2015 05:49	1.0M	
Edward Hervey - Continuous Integration: An update.pdf	29-Oct-2015 05:49	104K	
Edward Hervey - decodebin3.pdf	09-Oct-2015 04:33	276K	
Florent Thierry - gst-gtklaunch-1.0.pdf	29-Oct-2015 05:49	783K	
GStreamer Conference 2015 - Brochure.pdf	05-Oct-2015 06:35	787K	
Håvard Graff - PFAO performance: Audiocodes, Jitterbuffers, and an old friend.pdf	29-Oct-2015 05:54	740K	
Hans Verkuil - Colorspaces and HDMI.pdf	12-Oct-2015 09:28	188K	
Jan Schmidt - Distributed Acoustic Triangulation.pdf	10-Oct-2015 01:52	1.3M	
Jan Schmidt - Stereoscopic 3D Redux.pdf	10-Oct-2015 01:51	1.5M	
Jussi Pakkanen - Done in 6.0 second - a new build system for GStreamer?.pdf	29-Oct-2015 05:49	232K	
Justin Kim - Another Android Camera Source for GStreamer.pdf	29-Oct-2015 05:49	758K	
Luis de Bethencourt - How to Contribute to GStreamer.pdf	12-Oct-2015 09:03	1.2M	
Marcin Kolny - Remote GStreamer debugger.pdf	29-Oct-2015 05:49	1.3M	
Marcin Kolny - gstreamermm.c++ wrapper.pdf	29-Oct-2015 05:49	147K	
Marianna Buschle - Industrial Application Pipelines with GStreamer.pdf	29-Oct-2015 05:49	3.3M	
Matthew Waters - OpenGL for the GStreamer pipeline.pdf	14-Oct-2015 05:25	257K	
Melissa Montero - GST Daemon and Interpipes: A simpler way to get your applications done .pdf	29-Oct-2015 05:49	2.0M	
Nicolas Dufresne - Demystifying allocation query.pdf	04-Jan-2016 07:10	283K	
Nicolas Dufresne - Y4L2 GStreamer elements.pdf	29-Oct-2015 05:49	50K	
Ramesh V - Handling interleaved and non-interleaved streams with GStreamer.pdf	29-Oct-2015 05:59	386K	
Romain Picard - The HeliosTv Distributed DVB stack.pdf	19-Oct-2015 01:48	3.8M	
Sebastian Dröge - GstPlayer Cross-platform Playback API.pdf	10-Oct-2015 01:50	494K	
Sebastian Dröge - Synchronised multi-room media playback and distributed live media processing and mixing with GStreamer.pdf	10-Oct-2015 01:51	1.8M	
Thiago Santos - Analyzing Caps Negotiation using GstTracer.pdf	29-Oct-2015 05:49	229K	
Thibault Saunier - Non Linear Engine: replacement for GnonLin.pdf	29-Oct-2015 05:49	149K	
Thibault Saunier - What is new in GstValidate 1.6.pdf	29-Oct-2015 05:49	59K	
Tim-Philipp Müller - GStreamer State of the Union.pdf	19-Oct-2015 15:29	77K	
Yanessa Chipirrás Navalón - OpenCV and GStreamer (Demo Video).mkv	29-Oct-2015 05:49	6.5M	
Yanessa Chipirrás Navalón - OpenCV and GStreamer.pdf	29-Oct-2015 05:49	307K	
Wim Taymans - Camera Sharing and Sandboxing with Pinos.pdf	13-Oct-2015 04:07	320K	
Wim Taymans - ORC experiments.pdf	29-Oct-2015 05:49	280K	

(EX1014, annotated; captured January 6, 2016.)

Selecting the link in EX1011 leads to Hervey (EX1009) which was captured by the Internet Archive on February 26, 2017.

<https://web.archive.org/web/20170226222859/https://gstreamer.freedesktop.org/data/events/gstreamer-conference/2015/Edward%20Hervey%20-%20decodebin3.pdf>

(EX1009, annotated.)

Thus, Hervey is a printed publication because it was presented publicly at the GStreamer Conference 2015 without any requirement of confidentiality.

c) Hervey Declaration

Mr. Hervey is the author of Exhibit 1009 and has direct, personal knowledge

regarding his presentation of Exhibit 1009. He declares that he presented Hervey (“decodebin3: Modern playback use-cases”) at the 2015 GStreamer Conference. (EX1003, ¶4.) The Hervey presentation “was part of an open session of the conference and many industry participants attended.” (*Id.*, ¶5.) Copies of Hervey were “freely distributed without confidentiality restrictions to attendees at the conference in October of 2015.” (*Id.*, ¶6.) And, “[t]he presentation was also made available on the internet on GStreamer’s website shortly after the conference in 2015.” (*Id.*, ¶6.) The Hervey presentation was reported on contemporaneously and video recordings of the presentation were publicly available on the GStreamer website shortly after the conference. (*Id.*, ¶7.)

d) Contemporaneous Reporting

Shortly after the GStreamer 2015 Conference, the Hervey presentation was reported on by Linux Weekly News (LWN). On October 21, 2015, LWN published the article “*Upcoming work in and around GStreamer*” discussing Edward Hervey’s presentation on decodebin3. (EX1015.)



User: Password:

Upcoming work in and around Gstreamer

Benefits for LWN subscribers

The primary benefit from [subscribing to LWN](#) is helping to keep us publishing, but, beyond that, subscribers get immediate access to all site content and access to a number of extra site features. Please sign up today!

By **Nathan Willis**
October 21, 2015

[GStreamer Conference](#)

The 2015 GStreamer Conference included several talks that introduced new in-development features or ideas that are experimental in nature. They include a complete redesign of the automatic-decoding element, support for distributing broadcast television, and support for the WebRTC streaming protocol. Some of the features are being developed by members of the core GStreamer development team, while others originate in outside projects. As it frequently does, though, the conference allowed for these distinct teams to put their heads together.

decodebin3

Edward Hervey of Centricular presented his plans for a significant rewrite of the [decodebin](#) element, to get feedback from the GStreamer community before proceeding further. In GStreamer, the *-bin* suffix is used to designate an "auto-plugging" element that tries to recursively construct a processing pipeline to do the right thing for a common task. The decodebin element takes a media stream as input, figures out the stream type, and sets up the correct demultiplexing, parsing, and decoding elements for the content within the stream.

The trouble is that the current implementation, decodebin2, was written in 2006—when, Hervey said, "I didn't know the MPEG spec backward." So there are inefficiencies, such as the fact that decodebin2 decodes every media stream in the container file, even if only a subset of the streams are actually being played (say, one out of four available audio tracks). And switching between streams was convoluted. To continue the multiple-language-track example from above, switching from the English to the French track required setting up a new set of connections for the French audio *and* the video, emitting a signal to stop the current video and English audio streams, then restarting playback with the new setup. Worse still, if the user tried to do a seek event while this swap-over was being performed, they would jump to a position with no media available at all.

But these problems are magnified by more recent streaming protocols like HLS/DASH. A server might offer a selection of dozens of audio streams at a single URL; decodebin2 provides no way to inspect what each stream contains (even simple attributes like language tags), and decoding every stream would likely mean excessive overhead.

Hervey's plan is to write a new decodebin centered around a new, generic "stream" API. First, there will be a high-level GstStream object that makes important metadata (like video resolution or language tags) queryable. Next, multiple streams will be collected in an immutable GstStreamCollection

V. PERSON OF ORDINARY SKILL

A person of ordinary skill in the art as of the asserted priority date (“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. (EX1002, ¶¶34-36.)

Petitioner’s expert, Dr. Freedman, met this level by the priority date. (EX1002, ¶37.)

VI. CLAIM CONSTRUCTION

Per *Phillips*, Claim terms should be given their plain and ordinary meaning in view of the intrinsic evidence. (EX1002, ¶¶38-40.) The challenged claims have not been construed in other proceedings.

VII. GROUND 1: Orr In Combination With Hsu Renders Claims 1-20 Obvious²

A. Overview and Motivation to Combine

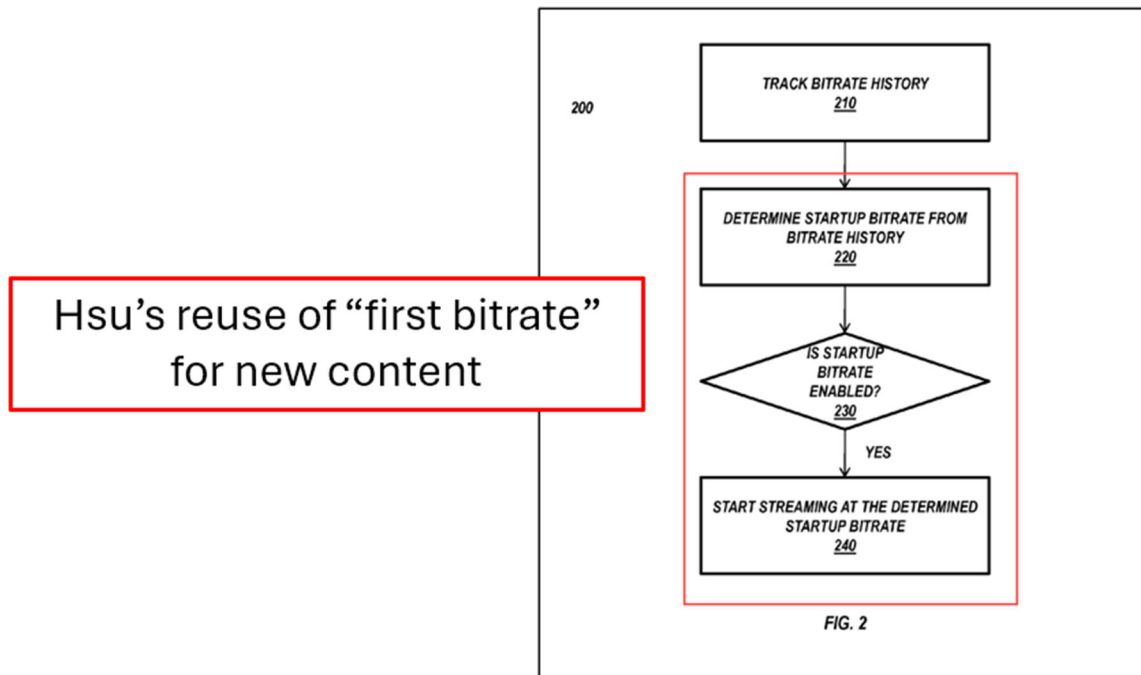
A POSITA would have been motivated to modify Orr to incorporate Hsu's teachings of bitrate initialization and adaptive bitrate streaming and would have had a reasonable expectation of success in doing so. (EX1002, ¶50.) A POSITA would have been motivated by the known benefits of reducing latency and 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. (*Id.*) A POSITA would have also been motivated by the increase in smooth stream playback from adaptive bitrate playback. (*Id.*)

The integration of Hsu's bitrate initialization component into Orr is

² Unless noted otherwise, all emphases in quotes and annotations to figures from prior art references are added.

technically straightforward and would have been well within the skill of the ordinary artisan. Orr's protocol stack manages 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 have implemented 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. (EX1002, ¶51.)

A POSITA would have been motivated to address the same fundamental user experience problem that Orr and Hsu identify: 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 changing to the default 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, by increasing streaming quality and reducing latency. (EX1002, ¶52.)



(EX1007, FIG. 2.)

A POSITA would also be motivated to incorporate Hsu's teachings of "Adaptive Bitrate Streaming," which Hsu explains is a known "technique used in streaming media" where "[t]he player switches between streaming the different bitrates depending on available resources." (EX1007, 3:66-4:5.) A POSITA would have been motivated by the known benefits of adaptive bitrate streaming, including "higher quality video using less manpower and fewer resources." (*Id.*)

These are 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. (EX1002, ¶53.)

A POSITA combining Orr and Hsu would implement Hsu's bitrate initialization teaching and adaptive bitrate streaming 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. (EX1002, ¶54.)

The combination of Orr and Hsu 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. (EX1002, ¶51.)

Orr and Hsu are analogous art. 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. (EX1002, ¶55.)

B. Limitation-By-Limitation Analysis

1. Claim 1

a) 1[pre]: A method comprising:

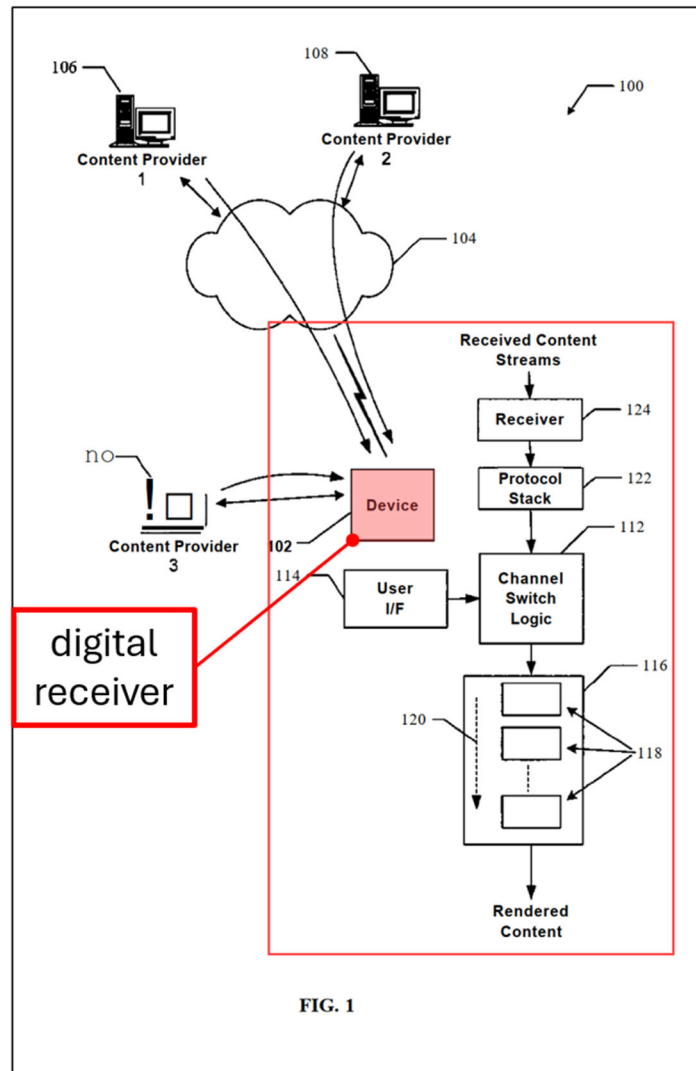
To the extent the preamble is limiting, Orr in combination with Hsu renders obvious this limitation. (EX1002, ¶¶56-57.)

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, by a digital receiver, of a first piece of content at a first bitrate, the first bitrate being a bitrate adapted for playback of the first piece of content;

“digital receiver”: Content is delivered to Orr’s device via a data network 104 as a *digital* signal. (EX1005, ¶[0016]; ¶[0017] (“The content providers 106, 108 and 110 operate to provide content to the device 102.”); FIG. 1.) Orr describes that the “circuits described in [Orr] may be implemented or performed with ... a digital signal processor (DSP). (*Id.*, ¶[0053].)

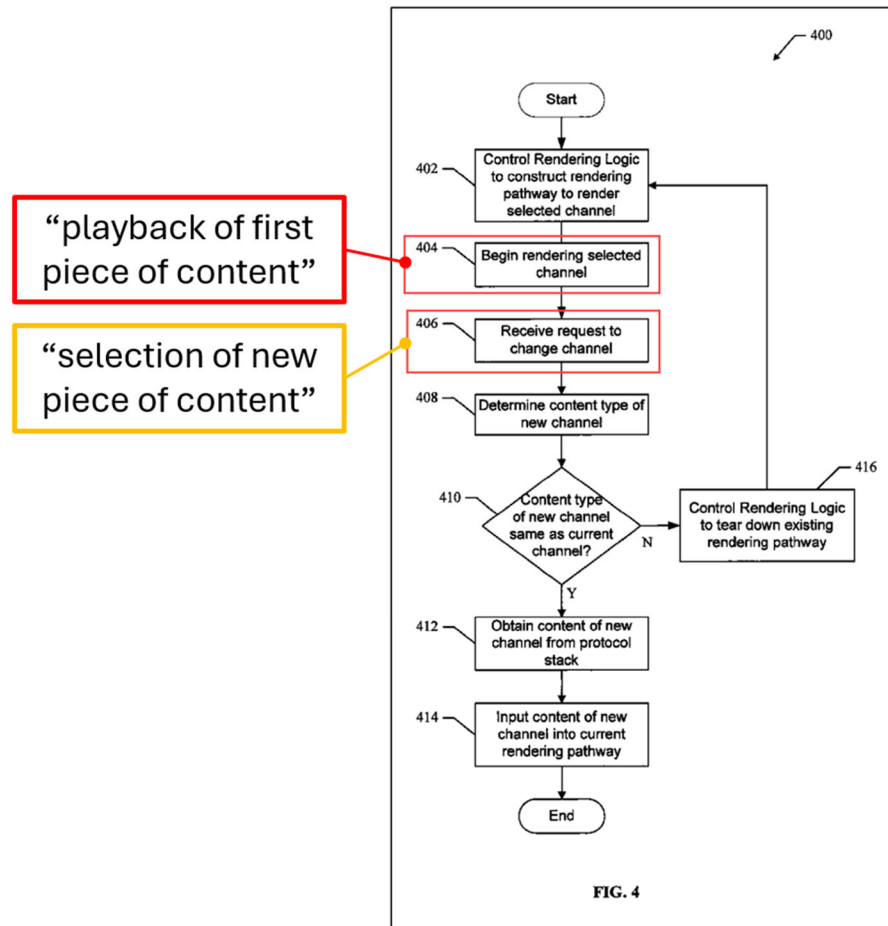
Orr describes its device 102 as including a receiver 124, channel switch logic 112, a user interface 114, a protocol stack 122, and rendering logic 116. (*Id.*, ¶[0018]; FIG. 1.) Orr’s “device 120” is an express disclosure of claim 1’s **“digital receiver.”** (EX1002, ¶¶58-60.)



(EX1005, FIG. 1.) (Annotated)

“causing playback ... of a first piece of content”: Orr describes that its device (i.e. ***digital receiver***) constructs a rendering pathway to ***cause playback of a first piece of content***. (EX1005, ¶ [0040-41]; FIG. 4, blocks 402–404.) FIG. 4 of Orr begins with “Control Rendering Logic to construct rendering pathway to render selected channel” (block 402) followed by “Begin rendering selected channel”

(block 404). (EX1002, ¶61.)



(EX1005, FIG. 4.) (Annotated)

“a first bitrate, the first bitrate being a bitrate adapted for playback of the first piece of content”: In the proposed combination, Orr renders media content using the adaptive bit rate methods described in Hsu, including Hsu’s disclosure of ***playback using a bitrate adapted for playback of the content***. Specifically, Hsu explains that “Adaptive Bitrate Streaming ... can detect a playback device bandwidth and other factors in real time and adjust the quality of a video stream

accordingly.” (EX1007, 3:62-4:2.) Hsu explains that “during playback of media content on the adaptive bitrate streaming media player,” adaptive bitrate streaming “determine[s] *bitrate values* used at a plurality of different times for providing the streaming media content.” (EX1007, claim 1.) The bitrate at which a first piece of content is played therefore reflects a bitrate that is adapted in response to real-time bandwidth conditions during playback. The current bitrate being used for playback, *i.e.* “the most recent bitrate,” is the claimed *first bitrate*. (EX1007, 6:2-3.) (EX1002, ¶62.)

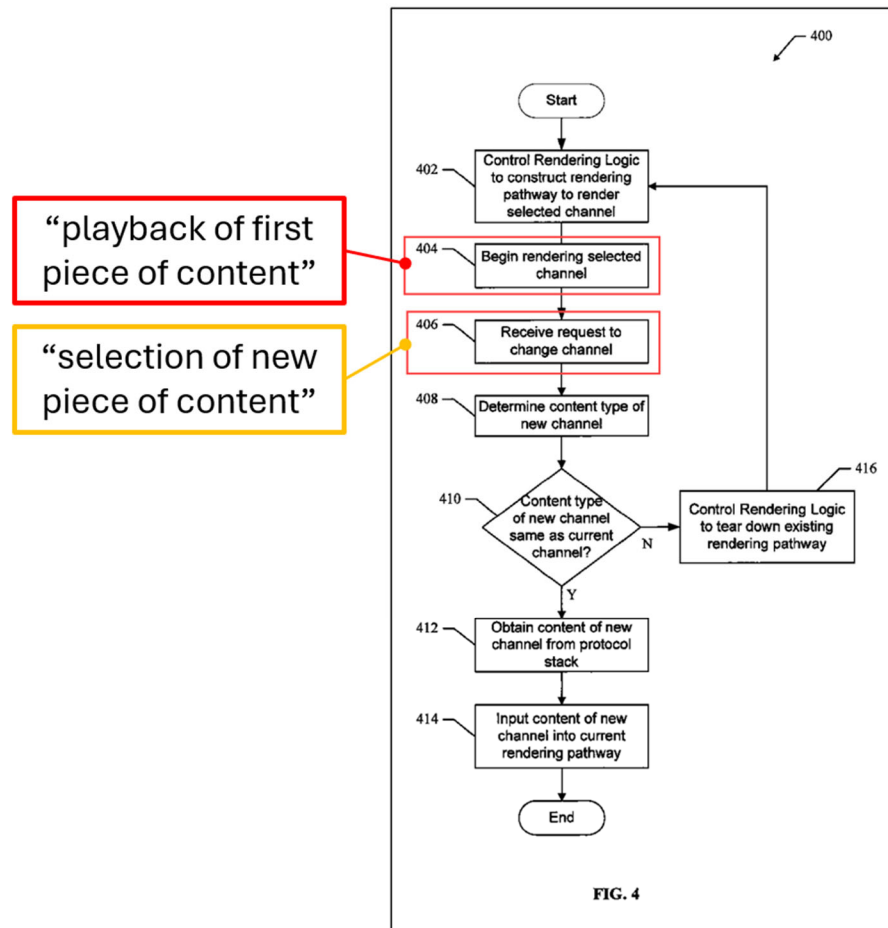
- c) **1[b]: receiving, by the digital receiver, a selection of a new piece of content for playback during the playback of the first piece of content;**

Orr describes that device 102 (the claimed “digital receiver”) *receives a selection of new piece of content*. 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].) “The 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].) (EX1002, ¶¶63-65.)

Orr likewise describes that the receiver receives this selection *during*

playback of the first piece of content: “[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 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.) (EX1002, ¶66.)

In the following annotated figure, FIG. 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, blocks 404, 406.) (EX1002, ¶67.)

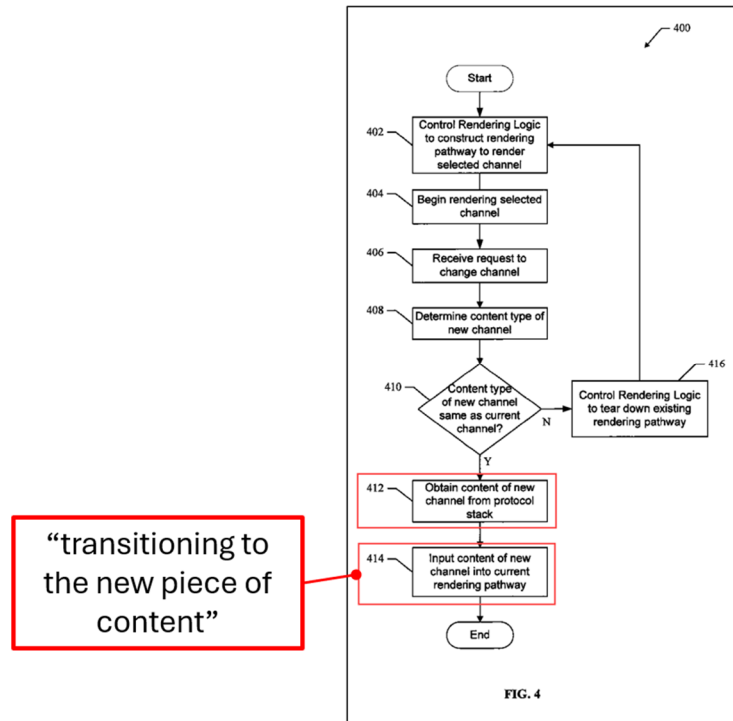


(EX1005, FIG. 4.) (Annotated)

- d) 1[c]: in response to the receiving of the selection, transitioning from ongoing playback of the first piece of content to the new piece of content by initiating playback of the new piece of content using the first bitrate used for playback of the first piece of content;

Orr describes *transitioning to the new channel's content in response to the channel selection*. In response to receiving a new channel selection indicator, the channel switch logic performs a series of operations that eventually results in obtaining the new content from the protocol stack and either inputs it into the

existing rendering pathway or constructs a new pathway, both of which are the device *transitioning to the new content*. (EX1005, ¶[0022]; ¶[0046-47]; FIG. 4, blocks 412–414.) The *transition to the new content* requires the selection of new content, and is therefore *in response to* the selection. 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,” which indicates that the transition is *in response to* a new channel selection. As shown in the annotated figure below, FIG. 4, in response to receiving a “request to change channel” at step 406, Orr’s device performs several steps (described *infra* at 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].) (EX1002, ¶¶68-69.)



(EX1005, FIG. 4.) (Annotated)

“playback of the new piece of content using the first bitrate used for playback of the first piece of content”: In the proposed combination, Orr transitions to new content using bitrate methods described in Hsu where new content is played using the previous content’s bitrate. Specifically, Hsu discloses that, upon initiating playback of new content, the player “determine[s] a starting bitrate” for the new content. (EX1007, 6:1-5.) Hsu further discloses that “[t]he technology can use the most recent bitrate [*i.e.* ***first bitrate***] as the startup bitrate.” (*Id.*) Hsu confirms that the new content’s “streaming begins at the determined startup bitrate.” (*Id.*, claim 1.) (EX1002, ¶70.)

- e) **1[d]: during playback of the new piece of content using the first bitrate, detecting whether the new piece of content can maintain playback at the first bitrate; and**
- f) **1[e]: in response to the detecting that the new piece of content cannot continue playback at the first bitrate, adjusting the first bitrate.**

In the proposed combination, Orr renders media content using the adaptive bitrate methods described in Hsu, including Hsu's continuous monitoring of *whether the current bitrate can be maintained during playback* and *adjusting the bitrate if it cannot*. Hsu's adaptive bitrate methods "can detect a playback device bandwidth and other factors in real time and adjust the quality of a video stream accordingly," where "[t]he player switch[ing] between streaming the different bitrates depending on available resources." (EX1007, 3:66-4:5; *see also id.*, claim 1 ("determining bitrate values used at a plurality of different times for providing the streaming media content").) Hsu also teaches that "it is important to start playback devices with consistently lower communications bandwidths with a starting video bitrate that will stream properly for those devices." (*Id.*, 5:46-49.) In other words, Hsu teaches that its adaptive bitrate methods checks "available resources" to see if *playback of the new content can be maintained at the current bitrate*. (*Id.*) If available resources cannot maintain the current bitrate, then "[t]he player switch[es]" to a "different bitrate," *i.e. adjust the first bitrate*. (*Id.*) (EX1002, ¶¶71-72.)

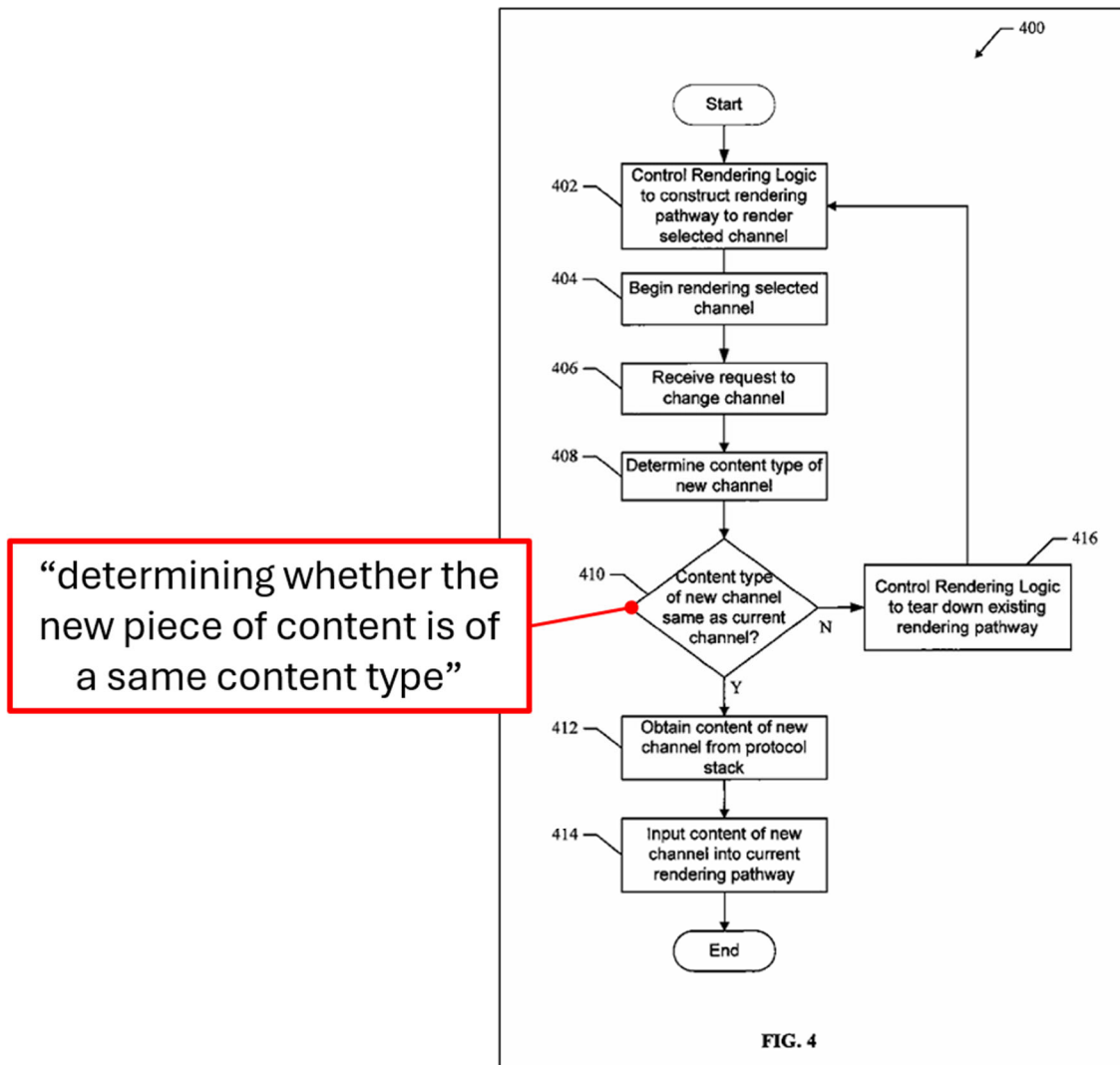
Hsu's disclosures are further supported by its teaching that for "users who change location between a higher bandwidth environment and a lower bandwidth environment," the "lower" bitrate is selected. (EX1007, 6:16-26.) Although Hsu describes this in the context of selecting an initial bit rate for the new content, in view of the other disclosures in Hsu cited above, a POSITA would understand that this principle applies to adapting the bitrate while the content is playing, especially to give effect to Hsu's teaching that the bitrate of the video is adjusted based on bandwidth. (*Id.*, 3:66-4:5.) (EX1002, ¶73.)

2. Claim 2

- a) **2[pre]: The method of claim 1, further comprising:**
- b) **2[a]: determining whether the new piece of content is of a same content type as the first piece of content; and**

Orr's device *determines whether the new piece of content is the same content type as the first piece of content*. Specifically, Orr's 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].) FIG. 4 depicts this determination explicitly at block 410: "Content type of

new channel *same as current channel?*” (EX1005, FIG. 4, block 410.) (EX1002, ¶¶74-75.)



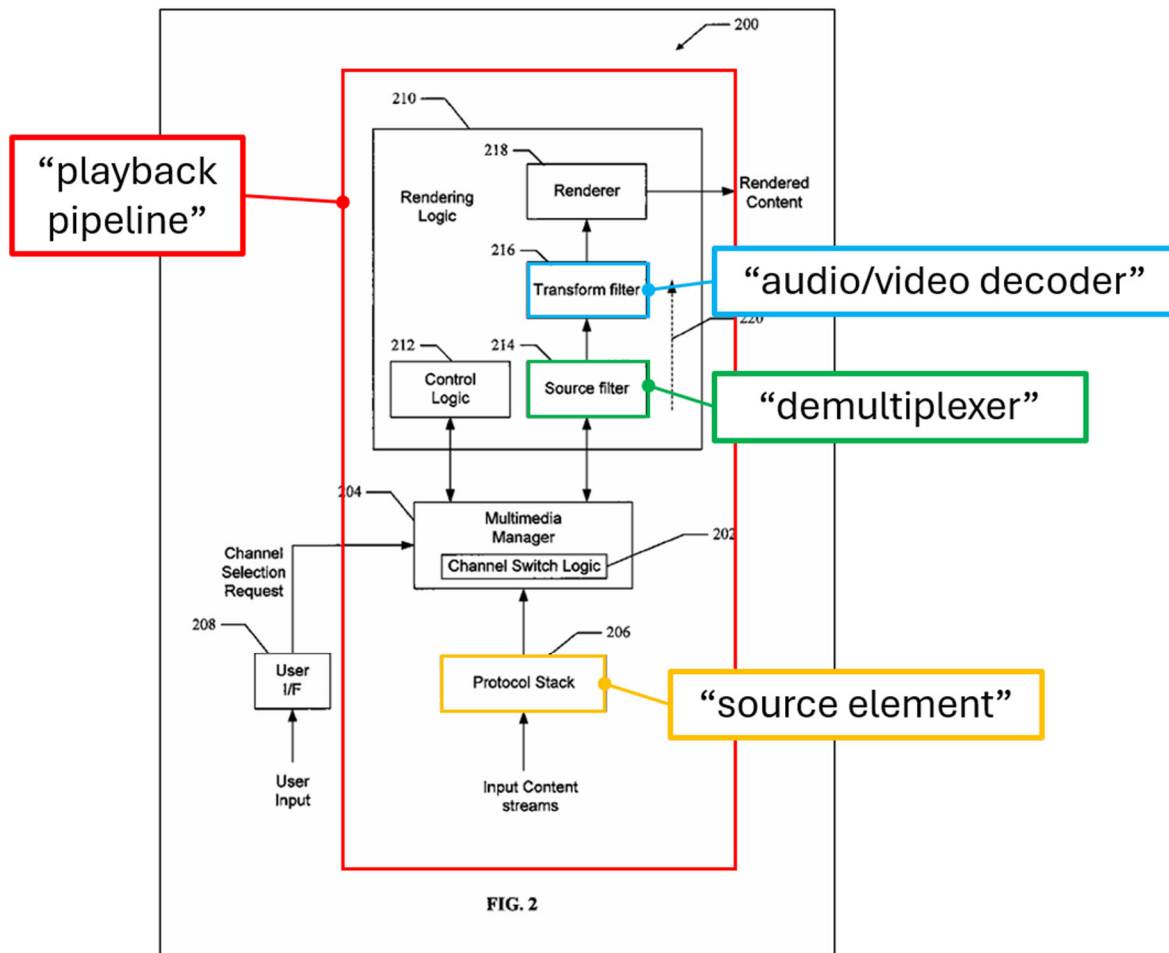
(EX1005, FIG. 4.) (Annotated)

- c) 2[b]: based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content, the preserving of at least the portion of the playback

pipeline comprising preserving one or more of a source element, a demultiplexer, an audio decoder, or a video decoder.

Orr discloses this limitation. This limitation requires (1) a “playback pipeline” for the first piece of content that has a source element, a demultiplexer, an audio decoder, and a video decoder, (2) based on determining the new content has the same content type, determining whether to preserve “at least a portion of the playback pipeline,” and (3) preserving one or more of the source element, demultiplexer, audio decoder, or video decoder. (EX1002, ¶¶76-77.)

“playback pipeline comprising ... a source element, a demultiplexer, an audio decoder, and a video decoder”: 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 the claimed source element, demultiplexer, audio decoder, and video decoder. (EX1002, ¶78.)



(EX1005, Fig. 2) (Annotated)

“Source element.” Orr discloses that its playback pipeline includes a *source element*. Specifically, Orr discloses a “protocol stack 206” and a “protocol stack interface 310,” which together constitute a *source element*. (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, it is the *source element* because it receives content from content sources. (EX1002, ¶79.)

“A demultiplexer.” Orr discloses that its playback pipeline includes a *demultiplexer*, *i.e.* a component that separates the video and audio signal. (EX1001, 5:34-36: “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].) (EX1002, ¶80.)

In addition, it would have been obvious to include a demultiplexer in Orr’s playback pipeline. 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. (EX1002, ¶81.) 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.

“An audio decoder.” 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,

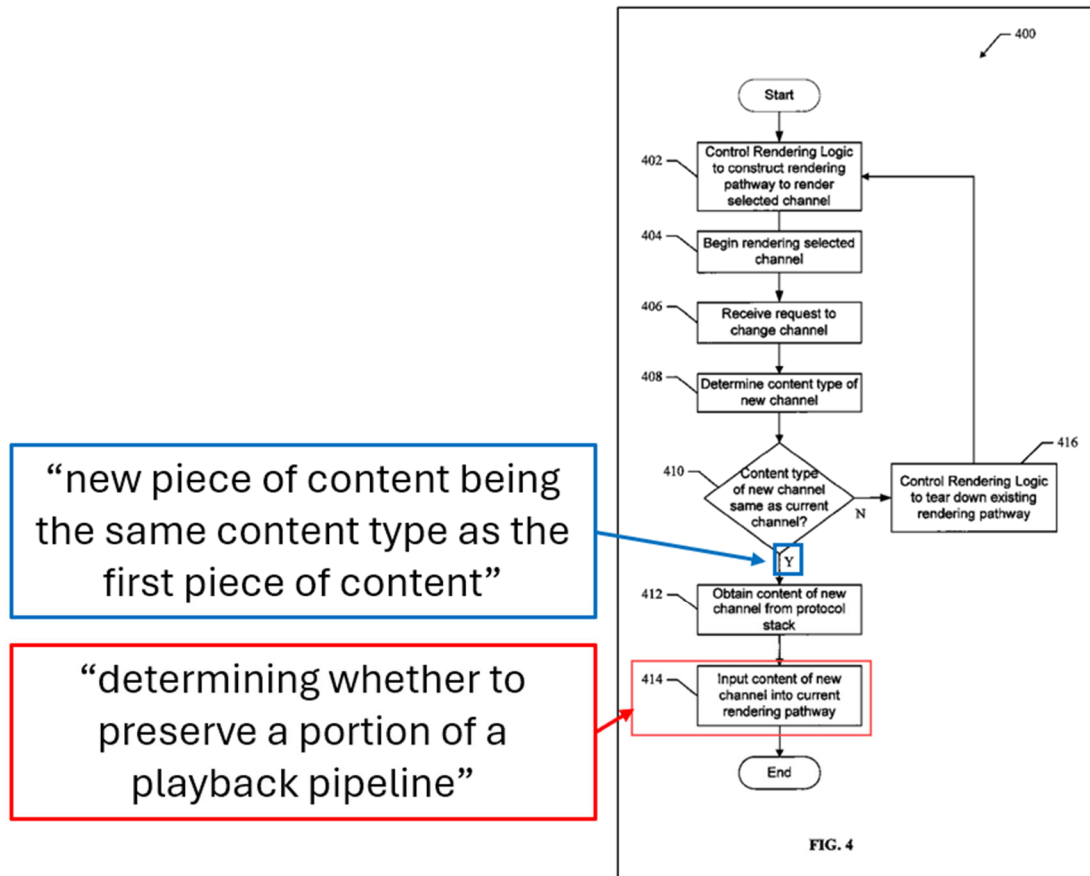
¶¶[0018]-[0019].) In addition, Orr discloses that the rendering logic includes a “transform filter 216,” which “is a pluggable filter” that “decompresses” (*i.e. decodes*) media content using a “decoder.” (EX1005, ¶[0024].) The pluggable filter sends the decoded signals to the “renderer” which “render[s] ... audio.” (EX1005, ¶[0024].) A POSITA would therefore understand that the transform filter of the rendering logic includes an *audio decoder*. (EX1002, ¶82.)

In addition, it 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. (EX1002, ¶83.) A POSITA would have had a reasonable expectation of success because audio decoders were ubiquitous at the time of the ’652 patent and it was well-within the skill of an ordinary artisan to implement an audio decoder in a playback pipeline.

“A video decoder.” Orr discloses that its playback pipeline includes a *video decoder*. Specifically, Orr discloses that its “transform filter 216 is a pluggable filter, such as an H264 decoder,” a well-known video decoder. (EX1005, ¶[0024].) (EX1002, ¶84.)

Based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content: When the content-type comparison at block 410 of FIG. 4 indicates that the new content type is “same as” the current content type, Orr’s channel switch logic does not reinitialize the rendering logic but instead feeds the new content directly into the existing pathway: “If the channel switch logic 112 determines that the current rendering pathway 120 can process and render the content on the new channel, then 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 depicts this at block 414: “Input content of new channel into current rendering pathway.” (EX1005, FIG. 4, block 414.) The processing logic makes this determination based on content-type compatibility. (EX1005, ¶[0037]; claim 1.) Orr therefore describes determining whether the type of content of the new piece of content is the same as the type of content of the first piece of content (at step 410), and preserving at least a portion of the current playback pipeline. (EX1002, ¶85.)

As shown below in FIG. 4, as annotated, step 410 compares the content types, and based on that comparison, step 414 is the *determination to preserve a portion of the playback pipeline*.



(EX1005, FIG. 4.)

The preserving of at least the portion of the playback pipeline comprising preserving one or more of a source element, a demultiplexer, an audio decoder, or a video: When the content types match, Orr’s existing source filter (*i.e. demultiplexer*) and transform filter (*i.e. audio and video decoders*) are preserved in

place and used to process the new channel's content for display. (EX1005, ¶[0022]; ¶[0028]; ¶[0047].) (EX1002, ¶86.)

3. Claim 3

- a) **3[pre]: The method of claim 2, wherein the determining whether to preserve comprises:**
- b) **3[a]: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and**

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]), which a POSITA would understand requires ascertaining the codec or format type of the new channel's content. A POSITA would understand that the format and codec requirements of content are included in metadata of that content. (EX1002, ¶¶87-88.) 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.

Orr further states: “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 decoder types the new content requires. Orr’s claim 4 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.” 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 therefore involves detecting the format and codec information of the new content. (EX1005, claim 4.) (EX1002, ¶89.)

c) 3[b]: determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.

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].) This is a direct comparison between the existing decoder types and the codec requirements of the new content (including audio and video decoders). (*See, e.g.*, EX1005, ¶[0024] (Orr describes the transform filter as “a pluggable filter, such as an H264 decoder that takes in data

from the source filter 214 and decompresses it.”) (EX1002, ¶¶90-91.)

4. Claim 4

- a) **The method of claim 3, further comprising preserving the entire playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.**

Orr teaches preserving the entirety of the existing rendering pathway when the content types match. As discussed above in claim 1, 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.) (EX1002, ¶¶92-93.) 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. (EX1002, ¶93.) Therefore, the entire playback pipeline is preserved.

5. Claim 5

- a) **5[pre]: The method of claim 3, further comprising:**
- b) **5[a]: releasing the audio decoder and the video decoder in the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information; and**

Orr teaches releasing the existing rendering pathway, which contain the audio and video decoders, when the content types differ. When “the content types are different,” Orr’s 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” when “the second content can not be rendered using the first rendering pathway.” (EX1005, claim 6.) “Tearing down” the rendering pathway releases the transform filter and decoder elements (including the audio and video decoders) that comprise it. For example, 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.) The teardown releases all elements of the rendering pathway including the source filter, the transform filter (corresponding to the demultiplexer, audio decoder, and video decoder), and the renderer. (EX1005, ¶¶[0045]; FIG. 2.) (EX1002, ¶¶[94-95].)

c) 5[b]: obtaining a new audio decoder and a new video decoder based on the codec information.

Orr teaches constructing a new rendering pathway with new functional elements appropriate to the new content's codec requirements after tearing down the old pathway. (EX1005, ¶¶[0045]–¶¶[0046]; FIG. 4, block 416 (returning to block 402 to “construct rendering pathway to render selected channel”).) Orr’s claim 6 recites “constructing a second rendering pathway to render the second content” following teardown, which involves acquiring new functional elements (including new transform filters and new audio and video decoders based on the codec information) suited to the new content’s format. (EX1005, claim 6.) These audio and video decoders are new because the new content was “formatted differently,” thereby necessitating “a new rendering pathway 220 to *decode* the content on the new channel.” (EX1005, ¶¶[0027].) In order to decode the new content that has a different format, a new video and new audio codec would be needed—as the previous codecs

had to be removed as they were incapable of decoding the new content. (EX1002, ¶¶96-97.)

6. Claim 6

Claim 6 is the “system” claim analog to claim 1 and largely mirrors the same claim limitations from claim 1. The differences between the claims are identified and analyzed below. The claim 6 analysis incorporates in its entirety the analysis for claim 1.

a) 6[pre]: A system comprising:

For the reasons discussed above in limitation 1[pre], Orr discloses this limitation. (EX1002, ¶98.)

b) 6[a]: one or more hardware processors; and

Orr discloses a device comprising hardware processors. Orr’s processing logic 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.) (EX1002, ¶99.)

c) 6[b]: a storage device storing instructions that configure the one or more hardware processors to perform operations comprising:

Orr discloses a storage device storing executable instructions. 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.) (EX1002, ¶100.)

- d) **6[c]: causing playback, by a digital receiver, of a first piece of content at a first bitrate, the first bitrate being a bitrate adapted for playback of the first piece of content;**

For the reasons discussed above in limitation 1[a], Orr discloses this limitation. (EX1002, ¶101.)

- e) **6[d]: receiving, by the digital receiver, a selection of a new piece of content for playback during the playback of the first piece of content;**

For the reasons discussed above in limitation 1[b], Orr discloses this limitation. (EX1002, ¶101.)

- f) **6[e]: in response to the receiving of the selection, transitioning from ongoing playback of the first piece of content to the new piece of content by initiating playback of the new piece of content using the first bitrate used for playback of the first piece of content;**

For the reasons discussed above in limitation 1[c], Orr in combination with Hsu renders obvious this limitation. (EX1002, ¶102.)

- g) **6[f]: during playback of the new piece of content using the first bitrate, detecting whether the new piece of content can maintain playback at the first bitrate; and**

For the reasons discussed above in limitation 1[d], Orr in combination with Hsu renders obvious this limitation. (EX1002, ¶103.)

- h) **6[g]: in response to the detecting that the new piece of content cannot continue playback at the first bitrate, adjusting the first bitrate.**

For the reasons discussed above in limitation 1[e], Orr in combination with Hsu renders obvious this limitation. (EX1002, ¶104.)

7. Claim 7

- a) **7[pre]: The system of claim 6, wherein the operations further comprise:**

For the reasons discussed above in limitation 2[pre], Orr discloses this limitation. (EX1002, ¶105.)

- b) 7[a]: determining whether the new piece of content is of a same content type as the first piece of content; and:**

For the reasons discussed above in limitation 2[a], Orr discloses this limitation. (EX1002, ¶106.)

- c) 7[b]: based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content, the preserving of at least the portion of the playback pipeline comprising preserving one or more of a source element, a demultiplexer, an audio decoder, or a video decoder.**

For the reasons discussed above in limitation 2[b], Orr discloses this limitation. (EX1002, ¶107.)

8. Claim 8

- a) 8[pre]: The system of claim 7, wherein the determining whether to preserve comprises:**

For the reasons discussed above in limitation 3[pre], Orr discloses this limitation. (EX1002, ¶108.)

- b) 8[a]: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content;**

For the reasons discussed above in limitation 3[a], Orr discloses this limitation. (EX1002, ¶109.)

- c) 8[b]: determining whether the audio decoder and the video decoder in the playback pipeline are of the**

types indicated by the codec information.

For the reasons discussed above in limitation 3[b], Orr discloses this limitation. (EX1002, ¶110.)

9. Claim 9

- a) **The system of claim 8, wherein the operations further comprise preserving the entire playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.**

For the reasons discussed above in claim 4, Orr discloses this limitation. (EX1002, ¶111.)

10. Claim 10

- a) **10[pre]: The system of claim 8, wherein the operations further comprise:**
- b) **10[a]: releasing the audio decoder and the video decoder in the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information; and**

For the reasons discussed above in limitation 5[a], Orr discloses this limitation. (EX1002, ¶¶112-13.)

- c) **10[b]: obtaining a new audio decoder and a new video decoder based on the codec information.**

For the reasons discussed above in limitation 5[b], Orr discloses this limitation. (EX1002, ¶114.)

11. Claim 11

- a) 11[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:**

For the reasons discussed above in limitations 6[a] and 6[b], Orr discloses this limitation. (EX1002, ¶115.)

- b) 11[a]: causing playback, by a digital receiver, of a first piece of content, the first bitrate being a bitrate adapted for playback of the first piece of content;**

For the reasons discussed above in limitation 6[c], Orr discloses this limitation. (EX1002, ¶116.)

- c) 11[b]: receiving, by the digital receiver, a selection of a new piece of content for playback during the playback of the first piece of content; and**

For the reasons discussed above in limitation 6[d], Orr discloses this limitation. (EX1002, ¶117.)

- d) 11[c]: in response to the receiving of the selection, transitioning from ongoing playback of the first piece of content to the new piece of content by initiating playback of the new piece of content using the first bitrate used for playback of the first piece of content;**

For the reasons discussed above in limitation 6[e], Orr in combination with Hsu renders obvious this limitation. (EX1002, ¶118.)

- e) 11[d]: during playback of the new piece of content using the first bitrate, detecting whether the new piece of content can maintain playback at the first**

bitrate; and

For the reasons discussed above in limitation 6[f], Orr in combination with Hsu renders obvious this limitation. (EX1002, ¶119.)

- f) 11[e]: in response to the detecting that the new piece of content cannot continue playback at the first bitrate, adjusting the first bitrate.**

For the reasons discussed above in limitation 6[g], Orr in combination with Hsu renders obvious this limitation. (EX1002, ¶120.)

12. Claim 12

- a) 12[pre]: The machine storage medium of claim 11, wherein the operations further comprise:**

For the reasons discussed above in limitation 7[pre], Orr discloses this limitation. (EX1002, ¶121.)

- b) 12[a]: determining whether the new piece of content is of a same content type as the first piece of content; and**

For the reasons discussed above in limitation 7[a], Orr discloses this limitation. (EX1002, ¶122.)

- c) 12[b]: based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to playback the first piece of content, the preserving of at least the portion of the playback pipeline comprising preserving one or more of a source element, a demultiplexer, an audio decoder, or a video decoder.**

For the reasons discussed above in limitation 7[b], Orr discloses this limitation. (EX1002, ¶123.)

13. Claim 13

- a) **13[pre]: The machine storage medium of claim 12, wherein the determining whether to preserve comprises:**

For the reasons discussed above in limitation 8[pre], Orr discloses this limitation. (EX1002, ¶124.)

- b) **13[a]: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and**

For the reasons discussed above in limitation 8[a], Orr discloses this limitation. (EX1002, ¶125.)

- c) **13[b]: determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.**

For the reasons discussed above in limitation 8[b], Orr discloses this limitation. (EX1002, ¶126.)

14. Claim 14

- a) **The machine storage medium of claim 13, wherein the operations further comprise preserving the entire playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being the types indicated by the codec information.**

For the reasons discussed above in claim 9, Orr discloses this limitation.

(EX1002, ¶127.)

15. Claim 15

- a) 15[pre]: The machine storage medium of claim 13, wherein the operations further comprise:**

For the reasons discussed above in limitation 10[pre], Orr discloses this limitation. (EX1002, ¶128.)

- b) 15[a]: releasing the audio decoder and the video decoder in the playback pipeline in response to the audio decoder and the video decoder in the playback pipeline being different than the types indicated by the codec information; and**

For the reasons discussed above in limitation 10[a], Orr discloses this limitation. (EX1002, ¶129.)

- c) 15[b]: obtaining a new audio decoder and a new video decoder based on the codec information.**

For the reasons discussed above in limitation 10[b], Orr discloses this limitation. (EX1002, ¶130.)

16. Claim 16: The method of claim 1, wherein: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated; and the adjusting comprises increasing the first bitrate.

Hsu teaches that adaptive bitrate streaming involves continuous monitoring of download conditions during playback and bidirectional adjustment of the bitrate in response. Hsu describes that adaptive streaming technology “can detect a playback

device bandwidth and other factors in real time and adjust the quality of a video stream accordingly,” and that “[t]he player switches between streaming the different bitrates depending on available resources.” (EX1007, 3:62-4:5.) Hsu further teaches that “[t]he playback device ... *switches to a higher bitrate* as conditions allow,” such as “where the bandwidth to support a higher bitrate is available.” (*Id.*, 5:33-49.) (EX1002, ¶131.)

A POSITA would understand that detecting that bandwidth is available involves determining that *downloading is taking less time than anticipated*. For example, U.S. Pat. Pub. No. 2008/0195745A1 (published August 14, 2008) (“Bowra”), explains that with “modern communication protocols, such as TCP,” if there is available bandwidth, “more and more packets are successfully received and acknowledged,” and “TCP ... increases the amount of data sent in each transmission. (EX1019, ¶[0002].) Since more data is sent, the download takes less time. (EX1002, ¶132.)

17. Claim 17: The method of claim 1, wherein: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking longer than anticipated; and the adjusting comprises decreasing the first bitrate.

Hsu teaches continuous monitoring of whether the current bitrate can be maintained during playback. Hsu describes that adaptive streaming technology “can detect a playback device bandwidth and other factors in real time and adjust the

quality of a video stream accordingly,” and that “[t]he player switches between streaming the different bitrates depending on available resources.” (EX1007, 3:62-4:5.) Hsu further teaches that for “users who change location between a higher bandwidth environment and a lower bandwidth environment,” the “lower” bitrate is selected. (EX1007, 6:16-26.) (EX1002, ¶133.)

A POSITA would understand that detecting lower bandwidth involves determining that *downloading is taking more time than anticipated*. For example, Bowra explains that “[w]hen the amount of data transmitted exceeds the available network bandwidth,” the TCP protocol will “respond to this by reducing the amount of data sent during each transmission. (EX1019, ¶[0003].) Since less data is sent, the download takes more time. (EX1002, ¶134.)

- 18. Claim 18: The system of claim 6, wherein: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated; and the adjusting comprises increasing the first bitrate.**

For the reasons discussed above in claim 16, Orr in view of Hsu renders obvious this limitation. (EX1002, ¶135.)

- 19. Claim 19: The system of claim 6, wherein the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download is taking longer than anticipated; and the adjusting comprises decreasing the first bitrate.**

For the reasons discussed above in claim 17, Orr in view of Hsu renders

obvious this limitation. (EX1002, ¶136.)

20. Claim 20

- a) **20[pre]: The machine storage medium of claim 11 wherein:**
- b) **20[a]: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download of the new piece of content is taking less time than anticipated and the adjusting comprises increasing the first bitrate; or**

For the reasons discussed above in claim 18, Orr in view of Hsu renders obvious this limitation. (EX1002, ¶137.)

- c) **20[b]: the detecting whether the new piece of content can maintain playback at the first bitrate comprises determining that download is taking longer than anticipated and the adjusting comprises decreasing the first bitrate.**

For the reasons discussed above in claim 19, Orr in view of Hsu renders obvious this limitation. (EX1002, ¶138.)

VIII. GROUND 2: Hervey In Combination With Hsu Renders Claims 1-20 Obvious

A. Overview and Motivation to Combine

A POSITA would have been motivated to integrate Hsu's bitrate initialization into Hervey and would have had a reasonable expectation of success in doing so. (EX1002, ¶139.) A POSITA would have been motivated by the known benefits of reducing latency and degradation of the stream quality by reusing the current bitrate

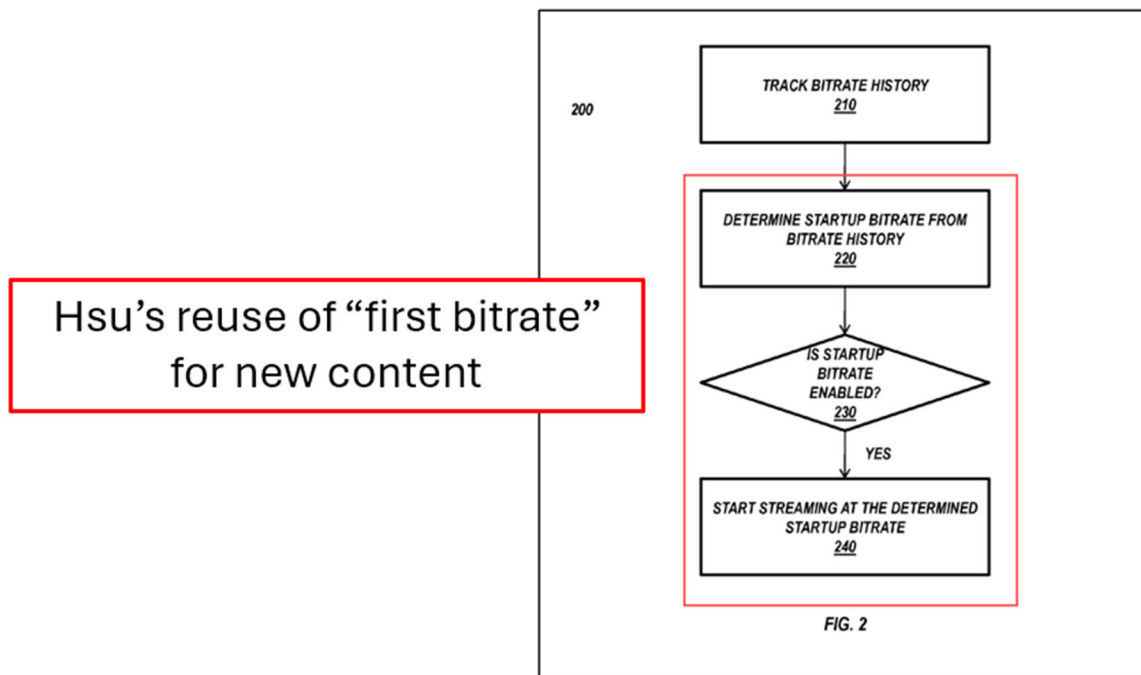
for new content, as changing to the default bitrate for new content requires time and resources, and also causes the new content to initially stream at a lower quality. (*Id.*) A POSITA would have also been motivated by the increase in smooth stream playback from adaptive bitrate playback. (*Id.*)

A POSITA would implement Hsu's bitrate initialization teaching in Hervey's source element, configuring uridecodebin to begin downloading new content at the previously achieved bitrate rather than the system default lowest bitrate. (EX1002, ¶140.)

A POSITA would have been motivated to address the same fundamental user experience problem that Hervey and Hsu identify: the latency experienced when switching between pieces of content on a streaming playback 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. (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]educ[e]s buffering.”); EX1002, ¶141.)

As shown in FIG. 2 below, Hsu addresses the bitrate initialization

component: eliminating the quality degradation and latency caused by changing to the lowest bitrate at the start of each new session. (EX1007, 5:33-39; FIG. 2.) Instead, Hsu 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. (EX1002, ¶142.)



A POSITA would also be motivated to incorporate Hsu's teachings of "Adaptive Bitrate Streaming," which Hsu explains is a known "technique used in streaming media" where "[t]he player switches between streaming the different bitrates depending on available resources." (EX1007, 3:66-4:5.) A POSITA would

have been motivated by the known benefits of adaptive bitrate streaming, including “higher quality video using less manpower and fewer resources.” (*Id.*)

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. (EX1002, ¶143.)

A POSITA combining Hervey and Hsu would implement Hsu’s bitrate initialization teaching and adaptive bitrate streaming 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. (EX1002, ¶144.)

The combination of Hervey and Hsu requires no architectural changes to either reference and produces the predictable result of a faster, higher-quality content transition. 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. (EX1002, ¶140.)

In addition, both Hervey and Hsu are analogous art, as they 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. (EX1002, ¶145.)

B. Limitation-By-Limitation Analysis

1. Claim 1

a) 1[pre]

The presentation 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.) (EX1002, ¶¶146-47.)

Playback in GStreamer

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

(EX1009, slide 4.)

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. (EX1002, ¶148.)

b) 1[a]

Hervey describes *causing playback of a first piece of content* on the digital receiver device. 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. (EX1009, slide 4.) Hervey further describes the pipeline as actively playing content during which stream switching events may be received: “Compare available, active, pending and requested streams.” (EX1009, slide 28.) The reference to “active” streams confirms that content is actively being played at

the time. (EX1002, ¶¶149-151.)

Playback in GStreamer

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

(EX1009, slide 4.)

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

(EX1009, slide 28.)

In the proposed combination, Hervey renders media content using the

adaptive bit rate methods described in Hsu. As discussed above in Ground 1, limitation 1[a], Hsu discloses *a bitrate adapted for playback of the first content*. (EX1002, ¶152.)

c) 1[b]

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, slide 28.) (EX1002, ¶153.)

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

(EX1009, slide 28.)

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, slide 32.) A POSITA would understand that the “about-to-finish” flag indicates that the current playing content is about to finish. (EX1002, ¶154.)

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

(EX1009, slide 32.)

d) 1[c]

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, slide 28.) (EX1002,

¶¶155-56.) This *transition to the new content* requires selection of the new content, and is therefore *in response to* that selection. (*Id.*)

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

(EX1009, slide 28.)

The gapless playback slide similarly describes switching to new content upon the “about-to-finish” signal. (EX1009, slide 32.) (EX1002, ¶157.)

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

(EX1009, slide 32.)

As discussed above in Ground 1, limitation 1[c], Hsu discloses *initiating playback of the new piece of content using the first bitrate used for playback of the first piece of content*. Thus, when combined with Hervey, this limitation is rendered obvious. (EX1002, ¶158.)

e) 1[d] / 1[e]

As discussed above in Ground 1, Hsu teaches these limitations. When combined with Hervey, the combination renders obvious this limitation. (EX1002, ¶159.)

2. Claim 2

a) 2[pre]

For the reasons discussed above in limitation 1[pre], Hervey discloses this

limitation. (EX1002, ¶160.)

b) 2[a]

Hervey renders obvious this limitation. (EX1002, ¶160.)

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, slide 28.) (EX1002, ¶161.)

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

(EX1009, slide 28.)

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, slide 29.) (EX1002, ¶162.)

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

(EX1009, slide 29.)

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. (EX1002, ¶163.) This is because a decoder of one particular type can only decode audio or video streams of that same type. (*Id.*) 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*.

A POSITA would also consider it obvious to determine whether the content is of the same content type when “[c]hecking if decoder can accept CAPS.” As

discussed, Hervey teaches linking the stream to the current rendering pipeline “if the decoder can accept” the media type of the new content. A POSITA would be motivated by the fact that a simple way of checking if the decoder can accept the media type of the new content is to compare the media type of the new content with the media type of the old content—because if the capabilities of the old and new content are the same, then the current decoder can “accept CAPS” of the new content. (EX1002, ¶164.) A POSITA would have had a reasonable expectation of success in implementing such a functionality as it would be a simple programming change.

c) 2[b]

“Playback pipeline comprising ... video decoder”: Hevey’s decodebin3 architecture explicitly discloses a ***playback pipeline*** comprising the claimed ***source element, demultiplexer, audio decoder, and video decoder***. (EX1002, ¶¶165-67.)

“Source element.” Hervey teaches that decodebin3 has a “single input sink pad,” *i.e.* a single element for receiving media input from media ***sources***. (EX1009, slide 25.) Hervey further teaches that decodebin3 contains the functionalities of “uridecodebin,” which was part of a previous version of decodebin. (EX1009, slide 33.) Hervey teaches that “uridecodebin” performs “buffering/queueing” of “network streams,” which is manages the sourcing of the media content. (EX1002, ¶168.)

“A demultiplexer.” Hervey discloses that decodebin3’s ***playback pipeline*** has

a “Demuxer,” which is a demultiplexer. (EX1009, slide 24.) The demuxer outputs a “GstStreamCollection,” which is a “collection” of streams, indicating that it demultiplexes the audio and video signals. (EX1009, slide 20.) (EX1002, ¶169.)

“An audio decoder.” Hervey discloses that decodebin3’s *playback pipeline* has an “ADec2,” *i.e.* an **audio decoder**. (EX1009, slide 24.) (EX1002, ¶170.)

“A video decoder.” Hervey discloses that decodebin3’s *playback pipeline* has a “VDec,” *i.e.* a **video decoder**. (EX1009, slide 24.) (EX1002, ¶171.)

These elements are express disclosure of the ’652 patent’s playback pipeline in terms of a “source element,” a “demultiplexer,” an “audio decoder,” and a “video decoder.” (EX1001, 5:11-41; FIG. 2.) (EX1002, ¶172.)

“Based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline”: Hervey teaches the pipeline preservation determination. The “Re-using decoders” slide states: “Check if decoder can accept CAPS. If so, just link. If not, unlink, insert new decoder, link.” (EX1009, slide 29.) (EX1002, ¶173.)

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

(EX1009, slide 29.)

This is a determination of whether to preserve the existing decoder based on whether the new content type is the same as the old content type. If the decoder can accept the CAPS of the new content, *i.e.* the CAPS of the new content matches the CAPS of the old content, the decoder is preserved (“just link”); if not, it is replaced (“unlink, insert new decoder”). (EX1002, ¶174.)

“Preserving one or more of the source element, the demultiplexer, the audio decoder, or the video decoder used to obtain the first piece of content for the new piece of content”: Hervey’s “Demuxer handling” slide teaches that the decoders, the demuxer and source elements may be 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, slide 30.) (EX1002, ¶175.)

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

(EX1009, slide 30.)

3. Claim 3

a) 3[pre]/3[a]

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, slide 29.) (EX1002, ¶¶176-77.)

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

(EX1009, slide 29.)

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. (EX1002, ¶178.)

b) 3[b]

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, slide 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. (EX1002, ¶¶179-80.)

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

(EX1009, slide 29.)

4. Claim 4

Hervey expressly teaches full pipeline preservation when the CAPS check succeeds. When “decoder can accept CAPS,” the result is “just link.” (EX1009, slide 29.) In other words, the existing decoder is preserved in place and connected directly to the new stream. 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, slide 30.) (EX1002, ¶¶181-82.)

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

(EX1009, slide 30.)

Together, these passages teach full pipeline preservation when the codec types match, including preserving the source element, demuxer, and all decoders.

(EX1002, ¶183.)

5. Claim 5

a) 5[a]

Hervey teaches releasing incompatible decoders. When the CAPS check fails, *e.g.*, when the existing decoder cannot accept the new stream's CAPS, Hervey teaches: “unlink, insert new decoder, link.” (EX1002, ¶¶184-85.)

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

(EX1009, slide 29.)

The “unlink” step is the release of the existing decoder: it is disconnected from the pipeline. A POSITA would understand “unlink” in this context as equivalent to “releasing” the decoder, as the effect is that the decoder is removed from the active pipeline and its resources are freed. (EX1002, ¶186.)

b) 5[b]

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. (EX1002, ¶¶187-88.)

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

(EX1009, slide 29.)

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, slides 20-21, 31.) A POSITA would understand that selecting and instantiating a new decoder based on the stream's CAPS is equivalent to “obtaining a new audio decoder and a new video decoder based on the codec information” as recited in this limitation. (EX1002, ¶189.)

6. Claim 6

a) 6[pre]

For the reasons discussed above in limitation 1[pre], Hervey discloses this limitation. (EX1002, ¶190.)

b) 6[a]

This limitation is obvious in view of Hervey. (EX1002, ¶¶191-92.)

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, slide 34.) (EX1002, ¶192.)

Summary

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

(EX1009, slide 34.)

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, slide 7.) (EX1002, ¶193.)

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

(EX1009, slide 7.)

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. (EX1002, ¶194.)

c) 6[b]

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, slide 34.) (EX1002, ¶195-96.)

Summary

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

(EX1009, slide 34.)

The “Decodebin2: Stream Switching” slide (EX1009, slide 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, slide 11.) (EX1002, ¶197.)

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

(EX1009, slide 11.)

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. (EX1002, ¶198.) In addition, storing CPU instructions in memory is expressly disclosed in Orr, as discussed in Ground 1. This establishes that well-known fact CPU instructions are stored in memory and loaded into the CPU registers for execution. (EX1002, ¶198.)

d) 6[c]

For the reasons discussed above in limitation 1[a], Hervey discloses this limitation. (EX1002, ¶199.)

e) 6[d]

For the reasons discussed above in limitation 1[b], Hervey discloses this limitation. (EX1002, ¶200.)

f) 6[e]

For the reasons discussed above in limitation 1[c], Hervey/Hsu renders obvious this limitation. (EX1002, ¶201.)

g) 6[f]

For the reasons discussed above in limitation 1[d], Hervey/Hsu renders obvious this limitation. (EX1002, ¶202.)

h) 6[g]

For the reasons discussed above in limitation 1[e], Hervey/Hsu renders obvious this limitation. (EX1002, ¶203.)

7. Claim 7

a) 7[pre]

For the reasons discussed above in limitation 2[pre], Hervey discloses this limitation. (EX1002, ¶204.)

b) 7[a]

For the reasons discussed above in limitation 2[a], Hervey discloses this limitation. (EX1002, ¶205.)

c) 7[b]

For the reasons discussed above in limitation 2[b], Hervey discloses this limitation. (EX1002, ¶206.)

8. Claim 8

a) 8[pre]

For the reasons discussed above in limitation 3[pre], Hervey discloses this limitation. (EX1002, ¶207.)

b) 8[a]

For the reasons discussed above in limitation 3[a], Hervey discloses this limitation. (EX1002, ¶208.)

c) 8[b]

For the reasons discussed above in limitation 3[b], Hervey discloses this limitation. (EX1002, ¶209.)

9. Claim 9

For the reasons discussed above in claim 4, Hervey discloses this limitation. (EX1002, ¶210.)

10. Claim 10

a) 10[pre]

b) 10[a]

For the reasons discussed above in limitation 5[a], Hervey discloses this limitation. (EX1002, ¶211.)

c) 10[b]

For the reasons discussed above in limitation 5[b], Hervey discloses this limitation. (EX1002, ¶212.)

11. Claim 11

a) 11[pre]

For the reasons discussed above in limitations 6[a] and 6[b], Hervey discloses this limitation. (EX1002, ¶¶213.)

b) 11[a]

For the reasons discussed above in limitation 6[c], Hervey discloses this limitation. (EX1002, ¶214.)

c) 11[b]

For the reasons discussed above in limitation 6[d], Hervey discloses this limitation. (EX1002, ¶215.)

d) 11[c]

For the reasons discussed above in limitation 6[e], Hervey/Hsu renders obvious this limitation. (EX1002, ¶216.)

e) 11[d]

For the reasons discussed above in limitation 6[f], Hervey/Hsu renders obvious this limitation. (EX1002, ¶217.)

f) 11[e]

For the reasons discussed above in limitation 6[g], Hervey/Hsu renders obvious this limitation. (EX1002, ¶218.)

12. Claim 12

a) 12[pre]

For the reasons discussed above in limitation 7[pre], Hervey discloses this limitation. (EX1002, ¶219.)

b) 12[a]

For the reasons discussed above in limitation 7[a], Hervey discloses this limitation. (EX1002, ¶220.)

c) 12[b]

For the reasons discussed above in limitation 7[b], Hervey discloses this limitation. (EX1002, ¶221.)

13. Claim 13

a) 13[pre]

For the reasons discussed above in limitation 8[pre], Hervey discloses this limitation. (EX1002, ¶222.)

b) 13[a]

For the reasons discussed above in limitation 8[a], Hervey discloses this limitation. (EX1002, ¶223.)

c) 13[b]

For the reasons discussed above in limitation 8[b], Hervey discloses this limitation. (EX1002, ¶224.)

14. Claim 14

For the reasons discussed above in claim 9, Hervey discloses this limitation. (EX1002, ¶225.)

15. Claim 15

a) 15[pre]

For the reasons discussed above in limitation 10[pre], Hervey discloses this limitation. (EX1002, ¶226.)

b) 15[a]

For the reasons discussed above in limitation 10[a], Hervey discloses this limitation. (EX1002, ¶227.)

c) 15[b]

For the reasons discussed above in limitation 10[b], Hervey discloses this limitation. (EX1002, ¶228.)

16. Claim 16

a) 16[pre]

b) 16[a]

For the reasons discussed above for Ground 1, Claim 16, Hsu teaches that adaptive bitrate streaming involves continuous monitoring of download conditions during playback and bidirectional adjustment of the bitrate in response, and that a POSITA would have understood Hsu's system to encompass increasing the bitrate in response to detecting that download is taking less time than anticipated. When combined with Hervey, the combination renders obvious this limitation. (EX1002, ¶229.)

17. Claim 17

a) 17[pre]

b) 17[a]

For the reasons discussed above for Ground 1, limitations 17[pre] and 17[a], Hsu teaches that its real-time bandwidth detection encompasses detecting that download is taking longer than anticipated when available bandwidth is insufficient to sustain delivery at the current bitrate, and that a POSITA would have understood Hsu's system to encompass decreasing the bitrate in response to detecting that download is taking more time than anticipated. When combined with Hervey, the combination renders obvious this limitation. (EX1002, ¶230.)

1. Claim 18

For the reasons discussed above in claim 16, Hervey in view of Hsu renders obvious this limitation. (EX1002, ¶231.)

2. Claim 19

For the reasons discussed above in claim 17, Hervey in view of Hsu renders obvious this limitation. (EX1002, ¶232.)

3. Claim 20

a) 20[pre]

b) 20[a]

For the reasons discussed above in claim 18, Hervey in view of Hsu renders obvious this limitation. (EX1002, ¶233.)

c) 20[b]

For the reasons discussed above in claim 19, Hervey in view of Hsu renders obvious this limitation. (EX1002, ¶234.)

IX. CONCLUSION

For at least the foregoing reasons, this Petition should be instituted.

Respectfully submitted,

Date: May 22, 2026

/s/ James M. Glass

James M. Glass (Reg. No. 46729)
QUINN EMANUEL URQUHART &
SULLIVAN LLP
295 5th Avenue, 9th Floor
New York, NY 10016
Email: jimglass@quinnemanuel.com
Phone: 212-849-7142
Fax: 212-849-7100

CERTIFICATION UNDER 37 C.F.R. § 42.24

Under the provisions of 37 C.F.R. § 42.24, the undersigned hereby certifies that the word count for the foregoing Petition for *inter partes* review (excluding the table of contents, table of authorities, mandatory notices, certificate of service or word count, and appendix of exhibits or claim listing) totals 13,157 words, which is within the word limit allowed under 37 C.F.R. § 42.24(a)(i.)

Date: May 22, 2026

/s/ James M. Glass

James M. Glass (Reg. No. 46729)
QUINN EMANUEL URQUHART &
SULLIVAN LLP
295 5th Avenue, 9th Floor
New York, NY 10016
Email: jimglass@quinnemanuel.com
Phone: 212-849-7142
Fax: 212-849-7100

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. §§ 42.6(e), 42.105(a), the undersigned hereby certifies service on the Patent Owner of a copy of this Petition and its respective exhibits at the official correspondence address for the attorneys of record for the '652 patent as shown in USPTO PAIR via FedEx:

Kudelski Group & MAIER & MAIER, PLLC
345 SOUTH PATRICK STREET
ALEXANDRIA, VA 22314
UNITED STATES

Courtesy copies were also sent via FedEx and electronic mail to Patent Owner's counsel of record in *OpenTv, Inc. v Pinterest, Inc.*, Case No. 1:25-cv-01404-JCG (D. Del.):

RKOpenTVPinterest@RobinsKaplan.com
TWaller@RobinsKaplan.com
CMorton@RobinsKaplan.com
BLinden@RobinsKaplan.com
akipp@robinskaplan.com

Date: May 22, 2026

/s/ James M. Glass

James M. Glass (Reg. No. 46,729)
QUINN EMANUEL URQUHART &
SULLIVAN LLP
295 5th Avenue, 9th Floor
New York, NY 10016
Email: jimglass@quinnemanuel.com
Phone: 212-849-7142
Fax: 212-849-7100