Preserving Virtual Reality Artworks

Tom Ensom & Jack McConchie
Time-based Media Conservation, Conservation Department, Tate

Document Version: 01.00
Document Date: 13 August 2021

Abstract

This report introduces virtual reality (VR) technologies and identifies the challenges artists and the cultural heritage sector face in achieving the long-term preservation of artworks which make use of them. It was produced as part of Tate's Preserving Immersive Media Project, an ongoing research project developing strategies for the preservation of artworks which utilise immersive media such as 360 video, real-time 3D, virtual, augmented and mixed reality. This report is intended to inform those interested in the preservation of VR artworks, particularly time-based media conservators, as to the components they are likely to receive when acquiring VR artworks, their characteristics and dependencies, and their vulnerability in terms of long-term preservation. The report concludes with recommendations for artists and institutions who are dealing with the immediate problem of caring from VR artworks, and with recommendations for further research.

Acknowledgements

This research was supported by Tate's Conservation Department. Thanks to Louise Lawson for project management and to Deborah Potter for editing suggestions.

This research was also supported in part by Lumen Art Projects Ltd. Thanks to Carla Rapoport and Jack Addis for their project support and network introductions.

Thank you to the artists we interviewed during this research for their insight into their working practice: Lawrence Lek, Michael Takeo Magruder, Winslow Porter, Jakob Kudsk Steensen, James Stringer, Milica Zec. Thanks to Judy Willcocks and Dom Biddulph for instigating a case study from the Museum & Study Collection at Central Saint Martins, UAL.

Thank you to Dragan Espenschied and Claudia Roeck, with whom we organised the iPRES 2019 VR hackathon in Amsterdam, and to all those who participated in the event.

Thank you to the members of the Preserving Immersive Media Group[1] for their insight and feedback on our work during the course of this project, particularly those who participated in our first meeting in 2019: Edward Anderson, Rasa Bočytė, Nelson Crespo, Savannah Campbell, Phillipa Day, David Edge, Patricia Falcão, Mark Hellar, Stephen McConnachie, Claudia Roeck. Thanks also to Brandon Butler for his clarifications regarding fair use and copyright in the US on an early draft.

[1] The Preserving Immersive Media Group (PIMG) is a mailing list and meeting series established by the Preserving Immersive Media Project, based on Group.io: https://groups.io/g/pimg/

Contents

1. Introduction

Tate is a major arts institution that houses the United Kingdom's national collection of British, international and contemporary art. Tate's Conservation department works to ensure that this collection is appropriately cared for and remains displayable in the long-term. The Time-based Media Conservation team is concerned with the preservation of video, slide, film, audio, performance and, more recently, software-based artworks. As part of the department's research programme, Tate keeps abreast of new technologies and their use in contemporary art, to ensure preparedness as the collection grows and diversifies. This has led to a new area of research exploring the preservation of artworks using *immersive media*. We use this umbrella term to describe various related technologies, including virtual reality (VR), augmented reality (AR) and mixed reality (MR) technologies, all of which have been designed to immerse a user in a virtual space or combine virtual and physical spaces. This report discusses a strand of the Preserving Immersive Media project[2] which has focused on VR artworks. This report will henceforth use the term *VR artworks* to refer to works which use 360 video or real-time 3D media and are designed to be viewed through a headset with some form of motion tracking. We note a rapid expansion in the use of related technologies and an evolving set of associated terminology, of which this report offers only a snapshot.

Related research at Tate precedes this project. Firstly, over the past decade the Time-based Media Conservation teams have carried out extensive research on the conservation of software-based art[3][4][5][6], of which immersive media forms a subset. As part of Tate's Software-based Art Preservation Project[7], this research has recently informed the development of new standardised workflows for new software-based artwork acquisitions. Recent acquisitions of artworks using real-time 3D technologies, which are closely associated with VR, have presented further opportunities to refine these. These have included John Gerrard's *Sow Farm, near Libbey, Oklahoma 2009* (2009), acquired in 2015, and Ian Cheng's *Something Thinking of You* (2015), acquired in 2020. Research has also been carried out on the suitability of virtual reality (VR) technologies to document complex physical installations of time-based media artworks[8]. The insights, approaches and workflows developed in this body of research offer a starting point for developing a strategy for preserving VR artworks.

[2] More information about the Preserving Immersive Media project can be found on the Tate project webpage: https://www.tate.org.uk/about-us/projects/preserving-immersive-media

[3] Patricia Falcão, *Developing a Risk Assessment Tool for the Conservation of Software-Based Artworks*,MA thesis, Hochschule der Künste Bern, 2010.

[4] Pip Laurenson, 'Old Media, New Media? Significant Difference and the Conservation of Software-Based Art', in *Preserving and Exhibiting Media Art. Challenges and Perspectives*, Amsterdam University Press, Amsterdam, 2013.

[5] Klaus Rechert, Patricia Falcão and Tom Ensom, *Introduction to an Emulation-Based Preservation Strategy for Software-Based Artworks*, 2016, http://www.tate.org.uk/research/publications/emulation-based-preservation-strategy-for-software-based-artworks.

[6] Thomas Ensom, *Technical Narratives: Analysis, Description and Representation in the Conservation of Software-Based Art*, Ph.D thesis, King's College London, 2019, https://kclpure.kcl.ac.uk/portal/en/theses/technical-narratives(e01bff94-08bd-4b83-aeef-4e7d6d5b0dfc).html.

[7] Tate, *Software-based Art Preservation – Project*, 2021, https://www.tate.org.uk/about-us/projects/software-based-art-preservation.

[8] Jack McConchie, *VR tools as spatial documentation*, presented at the American Institute for Conservation Annual Meeting 2018, Houston, Texas, May 31, 2018.

Beyond work at Tate, we also build on earlier exploratory research around the preservation of VR. In her MA thesis, *A Rift in Our Practices?: Toward Preserving Virtual Reality*[9], Savannah Campbell explored the history and development of VR, and reflected on past and emerging challenges in its preservation; particularly the failures of historical collecting efforts and the need to consider VR systems holistically. She also points out that "there are currently no best practices in place for preserving virtual reality hardware and software in cultural heritage institutions" and proposes emulation and migration strategies as possible pathways. A report by Candice Cranmer, *Preserving the emerging: virtual reality and 360-degree video*, *an internship research report*[10], reports on research undertaken during a six-week placement at the Netherlands Institute of Sound and Vision in 2017. The report briefly discusses emulation and migration, identifying potential feasibility issues with both. Cranmer's research also points to a need for further work to develop pragmatic strategies, as, "if institutions collect an array of new technology before they are ready to preserve in a proactive manner, loss of files and the integrity of the work may be compromised".

As a starting point for our research, we determined that we would first need to identify the VR technologies being used by artists and better understand their concerns regarding long-term preservation. To do so, we consulted a group of artists and makers engaged in the production of VR artworks through a series of interviews, studio visits and questionnaires. Based on insights from this process, alongside topics identified in the prior research described above, we identified three primary research aims:

1) To understand the key risks to the longevity of VR artworks created by the interdependency, variability and obsolescence of VR hardware and software.

2) To explore the viability of available preservation strategies for time-based media artworks when applied to VR artworks.

3) To develop pragmatic recommendations for the immediate care of VR artworks, aimed at artists and collecting institutions.

The first aim is addressed primarily in Sections 2, 3 and 4. Based on insights from our consultation with artists and makers, we were able to identify a set of common key components of VR systems, including hardware, software and video formats. These sections report on the findings of in-depth research into these key components, identifying their production and role within the VR system, including variable characteristics and risk factors. This is organised into specific component groups. Section 2 explores VR systems, which are the sets of off-the-shelf VR hardware and software used to realise VR artworks. In Section 3 and 4 we examine the production and display of the two major media types we identified: real-time 3D software and 360 video. The second aim is addressed in Section 5. For this work, we considered acquisition workflows for similar media and widely used preservation strategies for software-based art, grouped under four categories: stockpiling, hardware migration, emulation and code migration.

[9] Savannah Campbell, *A Rift in Our Practices, Toward Preserving Virtual Reality*. MA thesis, New York University, 2017, https://miap.hosting.nyu.edu/program/student_work/2017spring/17s_thesis_Campbell_y.pdf.

[10] Candice Cranmer, *Preserving the Emerging: Virtual Reality and 360-Degree Video*, internship research report, Netherlands Institute for Sound and Vision, 2017, http://publications.beeldengeluid.nl/pub/584/NISV_final_Candice-Krenmer.pdf.

For each of these we examine the extent to which, based on insights from Sections 2-4, they may be applied to the long-term care of VR artworks. Finally, in Section 6 we reflect on our findings and address aim three by offering a set of pragmatic steps that can be taken to stabilise VR artworks in the short-term. We also propose sector-level goals for future research and advocacy.

2. VR Systems

VR *systems* are the sets of interconnected hardware and software components which enable access to VR content. The purpose of these systems is to transform inputs (such as 3D data and tracking information) into the outputs (such as moving images and haptics) that create a VR experience. This section presents an overview of the components of VR systems, which we divide into two parts: hardware and software. While we make reference to the content created by artists in this section (e.g. 360 video or real-time 3D software), this section focuses primarily on off-the-shelf components which artists may select and source but do not typically create themselves. These can be characterised as *dependencies*, on which the VR application is dependent in realising a VR experience. While some dependencies might be critical to the VR system functioning at all (e.g. an HMD and suitable driver), others may have more subtle effects (e.g. a VR runtime artificially increasing frame rate). As these components tend to be proprietary and closed source, there is limited information available about the way they function, limiting the conclusions that can be drawn. We focus on describing the purpose and characteristics of each component and discuss their impact on the overall VR experience. Artist-produced VR content is explored further in 3. Real-Time 3D VR and 4. 360 Video.

2.1. VR System Hardware

VR systems involve an array of interlinked hardware components, which will typically include a head-mounted display (HMD), tracking system and controller(s), all of which are connected to a host computer. The relationships between these components are described in Figure 1 below. These systems are ultimately centered on the user through the tracking of their movements and controller inputs, data from which is translated into interaction with virtual space and thus the generation of the frames sent to the HMD.

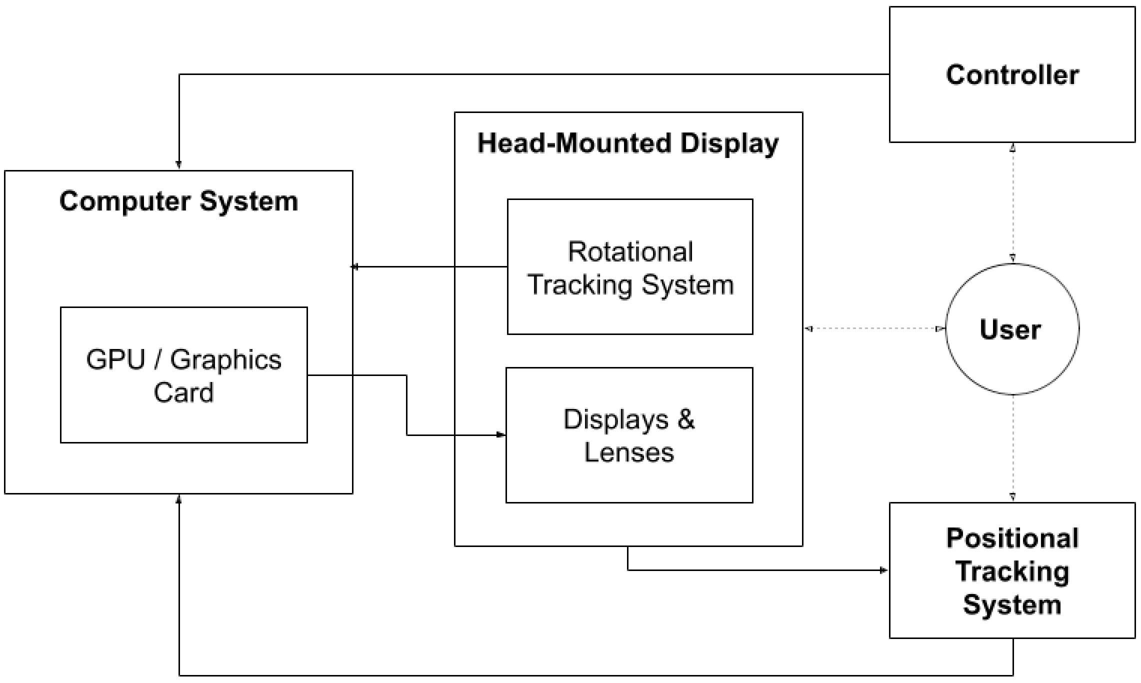

Figure 1. Diagram of a typical VR hardware system.

In the majority of cases, all of these components must be present in order for an VR artwork to be displayed, which creates immediate preservation risks through the potential for failure of the hardware used. Likelihood of hardware failure is elevated for some VR hardware, such as HMDs and controllers, as these are directly interacted with when installed. As for other specialised hardware encountered in time-based media conservation, as time passes obsolete components which stop functioning are likely to become impossible to replace with an equivalent device. The hardware systems used by artists tend to be consumer-oriented packages sold by a particular manufacturer and contain all the necessary hardware. The hardware systems sold and supported by VR companies currently move through rapid development and obsolescence cycles. For example, Oculus launched the consumer Rift in March 2016 and replaced it with the Rift S in March 2019, while HTC launched the Vive in April 2016 and replaced it with the Vive Pro in April 2018. In both cases, there were significant changes to the hardware and connectivity. This can be contrasted to, for example, Sony's 20" PVM CRT monitors which were in production for over 20 years and remained compatible with a consistent set of video standards.

2.1.1. Head-Mounted Displays

A head-mounted display (HMD) is a display device which is attached directly to a user's head. HMD devices contain either one or two screens, positioned in order to display a monoscopic or stereoscopic image at close proximity to the user's eyes. This close proximity creates a wide field of view image in order to increase immersion, while stereoscopy creates the illusion of depth perception. HMDs can be tethered (i.e. connected to a PC via physical cabling) or untethered (i.e. either standalone or connected to a PC wirelessly). At the time of writing, the latter is typically achieved by inserting a mobile device into an HMD (e.g. the Samsung Gear VR) or using mobile technology integrated into the HMD itself (e.g. the Oculus Quest).

As we found HMDs were used by all the artists we spoke to, for the purposes of this report we assume the use of these devices for viewing real-time 3D software and 360 video. There are, however, other approaches that could be taken, such as large scale projection or video monitor walls. It should be noted that in many cases an HMD could be substituted for a static screen (along with keyboard, mouse or other peripheral device input to control viewing direction).

We have identified the following variable characteristics among HMDs:

- **Screen panels.** There is variation in the type (LCD/OLED)[11], colour reproduction, pixel density, refresh rate, aspect ratio and latency of the panels embedded in HMDs.
- **Lenses.** In order to achieve clear visibility of the panels at close range and to maximise field of view, thick Fresnel lenses are placed between the eye and the panels. The image distortion they introduce must be corrected for in the frames sent to the HMD (see 2.1.1. Head Mounted-Displays) using a hardware-specific algorithm (see 2.2.1.

[11] A Real New World, *Understanding VR HMD Display Technology*, 2018, https://realnewworld.com/vr-hmd-display-technology/.

VR Runtimes) which can result in variation in the quality of the image across the field of view.

- **Field of view.** In the context of HMD design, field of view (FOV) is the extent to which the LCD panels completely cover the user's vision. This is a product of the panel and lense combination used.
- **Tethering.** HMDs can be tethered, meaning they attach to a host computer system (e.g. the Oculus Rift), or self-contained, meaning they contain a computer system (e.g. the Oculus Quest).
- **User experience.** The experience of wearing an HMD can vary considerably based on characteristics of its physical construction such as eye piece cushioning, strap design and overall weight.
- **User calibration.** HMD manufacturers provide a calibration procedure to customise the headset for the individual user e.g. interpupillary distance.

2.1.2. Tracking Systems

Tracking systems are used to capture the user's rotational and positional movement within physical space, so that this can be translated into movement in virtual space. The range of movements able to be interpreted and therefore represented in a virtual experience can be categorised as: three degrees of freedom (3DoF) and six degrees of freedom (6DoF). 3DoF refers to the pitch, yaw and roll of the head, rotational movements that can be interpreted by internal motion sensors in an HMD. 6DoF expands on this to include the positional movements of up/down, left/right and forwards/backwards. These positional movements are monitored by tracking systems requiring external components or reference to external surroundings. From a user's perspective, 3DoF allows the user to look around, whereas 6DoF allows the user to move and look around. The six types of movement possible are visualised in Figure 2 below.

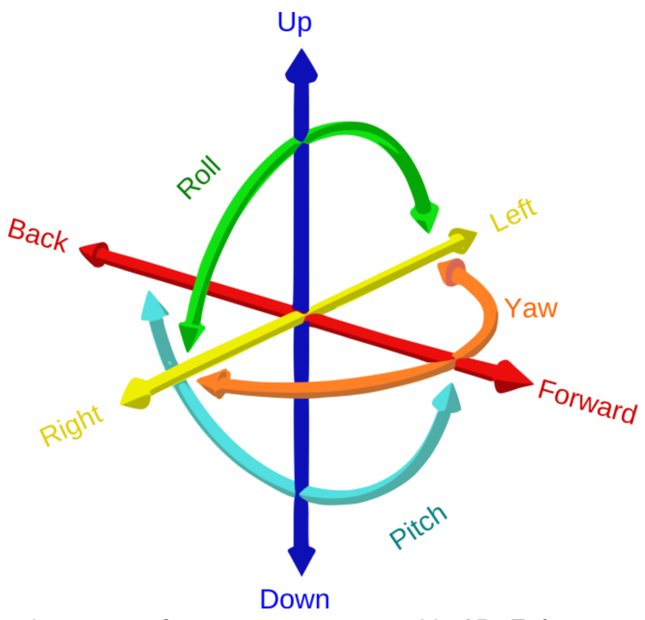

Figure 2. Diagram showing types of movement supported in 6DoF. Image credit: GregorDS, 2018, https://en.wikipedia.org/wiki/Six_degrees_of_freedom#/media/File:6DOF.svg, CC BY-SA 4.0.

There are two technical approaches to implementing tracking systems known as *inside-out* and *outside-in* tracking respectively. Inside-out tracking uses sensors placed on the HMD. Outside-in tracking uses sensors mounted in the external environment. Both approaches may utilise markers placed in the environment or on the HMD to improve tracking.

We have identified the following variable characteristics among tracking systems:

- **Area.** Flexibility in the tracked area can vary, including the maximum/minimum size and shape limitations (e.g. to accommodate physical spaces without four straight sides).
- **Resolution.** Systems can capture and transmit different densities of tracking data for software processing.
- **Latency.** The time taken for tracking data to be processed and translated into movement in virtual space. From the perspective of the user, this contributes to the level of responsiveness when interacting with the virtual environment.
- **Occlusion.** The ability of the system to tolerate occlusion (i.e. blocking of a tracking device or marker by a physical obstacle).
- **Hand/finger tracking.** The level of support for tracking movement in the hands or fingers of the user. Hand and finger movement can then be represented in virtual space as a means of increasing immersion within and interaction with the virtual environment.
- **Eye tracking.** Tracking systems can support tracking of the user's eye movement. This information can be used in rendering techniques which increase performance, like foveated rendering, or as a means of interacting with the virtual environment.
- **Virtual boundary representation.** Tracking systems use supporting software to create a representation of the physical boundaries of the interactive area in virtual space, to help to avoid problems like users colliding with their surroundings when wearing an HMD. The way in which the software represents the boundary limits of the interactive area to the user can vary (e.g. Oculus Guardian, Vive Chaperone).

2.1.3. Controllers

Controllers are physical devices which allow a user to interact with the virtual environment. A VR system can use a hand-held video game controller, combining joysticks, buttons and pads, and designed for screen-based gaming. VR hardware manufacturers have also experimented with various forms of custom controller. These tend to prioritise freedom of arm movement when compared to the typical form factor of a video game controller, which requires the two hands of the user to be held in a fixed, relative position. In some cases these controllers may be visibly represented inside the virtual environment to create a continuity between physical sensation and visual perception of virtual space.

We have identified the following variable characteristics among controllers:

- **Design.** The form factor and layout of the device, particularly the way it is gripped, creates a particular user experience (e.g. the Oculus Rift used trigger-type controllers, while the Vive used wand-type controllers).
- **Button/stick/pad configuration.** The number and the types of interactable elements on the controller allow for different levels of control to be configured.
- **Haptics.** Physical feedback in the controller (e.g. vibrations) can be created to respond to events in the virtual environment, with the aim of increasing immersion.
- **Virtual representation.** A representation of the controller and/or player hands can be created inside the virtual environment, with the aim of increasing immersion.

2.1.4. Computers

Computer systems orchestrate the execution of the software and its interaction with the VR hardware. The high performance requirements of VR have tended to require the use of powerful desktop PCs, including a dedicated graphics card (GPU). However, mobile computing technology is also being used and may become increasingly popular as it becomes more powerful. This has resulted in all-in-one HMD like the Oculus Quest (which relies on a small computer built into the HMD itself[12]) and the GearVR, which is a HMD designed for pairing with a Samsung mobile phone. While the exact computer hardware requirements of a VR application can vary considerably depending on the rendering techniques used, they tend to utilise a certain set of hardware components which are described in Table 1.

Name	**Description**
Central Processing Unit (CPU)	The CPU executes instructions contained in program code when an application is run. For VR applications, this includes managing the rendering of frames by sending jobs to the GPU and running physics simulations[13].
Random Access Memory (RAM)	RAM is volatile memory into which applications are loaded when they are executed by the CPU. This is as important for VR applications as any other software but is less likely to significantly impact performance compared to the CPU or GPU.
Graphics Processing Unit (GPU)	The GPU is a specialised piece of hardware used in the rendering of 3D graphics. GPUs can be found as dedicated expansion cards (as is frequently the case for VR systems) or integrated into some CPUs.
Storage Devices	Storage devices contain non-volatile memory used to store user programs and data. SSDs provide faster speeds so are often favoured over traditional magnetic HDDs for high performance applications such as VR.
Interfaces	VR systems can use a large number of peripheral devices which must be connected to the host machine, and so require the relevant interfaces to be present. This typically includes multiple USB ports for tracking and sensor

[12] Jad Meouchy, *Oculus Quest Teardown*, 2019, https://medium.com/badvr/oculus-quest-headset-disassembly-2f404b004a3c
[13] Google, *VR Performance best practices*, 2018, https://developers.google.com/vr/develop/best-practices/perf-best-practices.

	data and HDMI/DisplayPort connections for sending frames to the HMD and a monitor.

Table 1. Key components of a desktop computer suitable for running VR applications.

As a result of the large range of components available with which to construct computer systems and their complex interactions, understanding the variance they introduce to VR systems is beyond the scope of our research. However, we can make two broad generalisations about the significance of computer hardware. The first is that the primary purpose of hardware selection when creating a system to support a VR application is performance. Each component can be critical in lowering latency and increasing the speed with which frames are generated, both of which are important factors in creating a comfortable experience for a user. The second is that the GPU is of primary importance within this constellation of components, given its critical role in the processing of shaders and the creation of the frames sent to the HMD. Different GPU models, in combination with the specific driver versions installed, support different rendering features. For example, to use features of DirectX 11, the GPU and driver combination must support DirectX 11.

2.2. VR System Software

VR systems depend on several layers of software which operate in conjunction with hardware. These layers are illustrated in Figure 3 below.

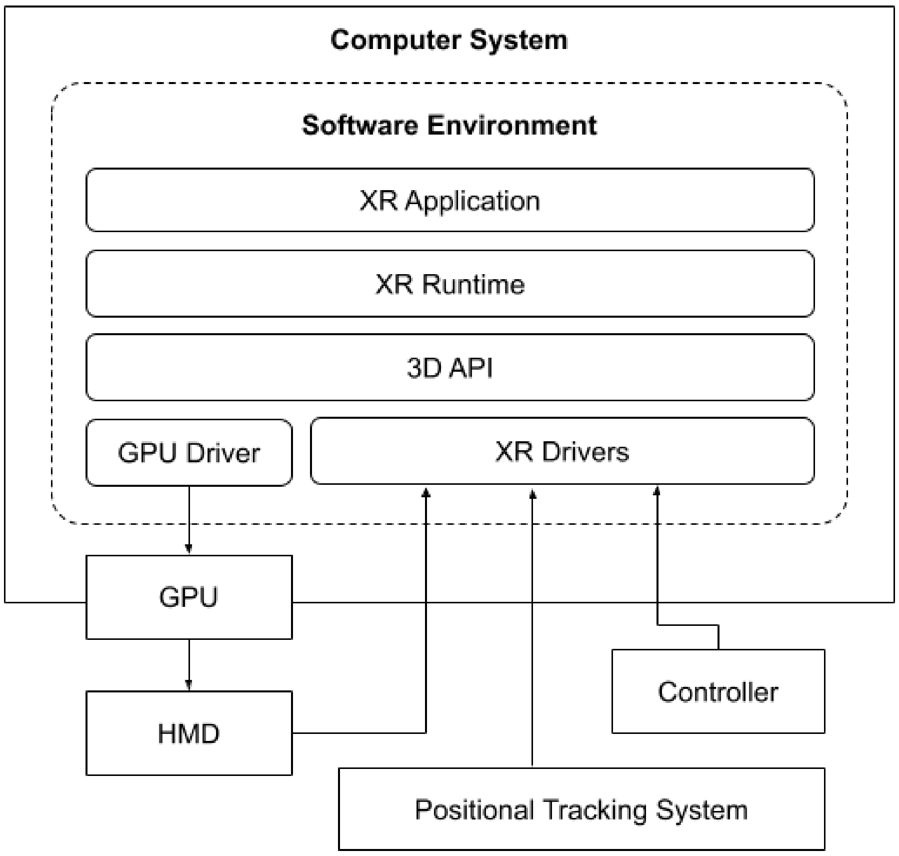

Figure 3. Diagram of components found in a generic VR software environment.

The software components of a VR system center on the VR application itself, the execution of which engages these other layers. VR applications are not discussed in this section but are covered in detail in Sections 3 (for Real-Time 3D) and 4 (for 360 Video). This section instead focuses on understanding the variance introduced by the other third-party software layers on which a VR application depends. Software dependencies arise through design decisions during the development process, usually because of the choice to target a specific set of hardware. The generic core components of software environments used to run VR applications are summarised in Table 2 below and discussed further in the subsequent sections.

Name	**Description**	**Examples**
Operating System	Supports a computer's general operation, including managing interactions between software and hardware.	Windows, Mac OS, Linux
3D API	Abstraction layer used by software to access the features of graphics hardware. Typically packaged with an OS e.g. DirectX is a core component of Windows, while Metal is a core component of MacOS.	Vulkan, DirectX, OpenGL, Metal
VR Runtime	Provides software with access to VR hardware (such as HMDs and tracking systems) and implements certain rendering features (e.g. lens distortion).	Oculus, OpenVR (Vive/SteamVR), Windows Mixed Reality
Device Drivers	Controls a connected hardware device. For consumer oriented VR products, device drivers are typically bundled with and managed by the VR runtime.	GPU/display, HMD, controllers, positional tracking system
Additional Libraries	Any additional libraries required to run the application which are not found in the operating system or VR runtime.	.NET/Mono runtimes, MSVC++ runtimes

Table 2. Description of the core components of a generic VR system with real-world examples of each technology.

Comments by the artists and makers we spoke to indicate that the management of software environment components is a significant challenge when working with VR. These environments are made particularly volatile by the tendency for VR runtime software to integrate automatic software updates. Several of the artists we spoke to commented that these updates made maintaining access to specific versions of their software challenging by breaking dependencies. In some cases, they had resorted to creating dedicated offline systems to ensure they remain static snapshots of the software environment. It is clear that if we hope to preserve and manage such an environment, particularly if we hope to effectively apply strategies such as emulation, we need to ensure we archive snapshots of working software environments. We recommend that the storage volumes containing software environments for artist-verified systems be captured using disk imaging for preservation purposes — a topic we discuss in more detail in 5.1. Acquiring VR Artworks.

2.2.1. VR Runtimes

A VR runtime is a software package which orchestrates communication between application software and VR hardware. VR runtimes are typically closely linked to the manufacturer of VR hardware. A brief summary of actively developed VR runtimes can be found in Table 3 below.

Name	**Description**	**Target Platforms**
SteamVR	Originally created for Valve's own Vive VR hardware, it has since expanded to include support for other VR systems. Implements Valve's own OpenVR[14] specification, but is not itself open source[15].	Windows, Linux, MacOS (beta only)
Oculus	Oculus's runtime for their Rift VR systems. Note: Oculus Go has the mobile version of the runtime installed on the HMDs embedded hardware.	Windows, MacOS, Linux
Open Source Virtual Reality (OSVR)	An open-source VR runtime intending to add support for all major VR hardware. It's future is uncertain, as commits to their GitHub repository have been infrequent since 2017. The main contributor, Ryan A. Pavlik, is now working on the OpenXR specification.	Windows, Android, Linux (partial support), Mac OS X
Daydream / Cardboard	Google's mobile-only platform for VR. Uses 'Google VR Services' on supported Android versions and phones.	Android
GearVR	A mobile-focused VR HMD kit developed by Samsung, which requires the use of a compatible Samsung Galaxy mobile phone.	Android
Windows Mixed Reality	Supports a variety of Windows Mixed Reality HMDs and the HoloLens HMDs. Has OpenXR support.	Windows
PSVR	A VR system for Sony's Playstation 4. Requires a licence to develop for — no public SDK/engine/tools.	Playstation 4
ARKit	Apple's augmented reality (AR) platform. The runtime is integrated into iOS 11 onward and supports hardware.	iOS
ARCore	Google's mobile-only augmented reality (AR) platform. Uses 'Google Play Services for AR' on supported Android versions and phones.	Android

[14] More information about OpenVR can be found in this article: Matias Nassi, *Introduction to OpenVR 101 Series: What is OpenVR and how to get started with its APIs*, 2018, https://skarredghost.com/2018/03/15/introduction-to-openvr-101-series-what-is-openvr-and-how-to-get-started-with-its-apis/
[15] See discussion on OpenVR's GitHub repository: https://github.com/ValveSoftware/openvr/issues/154

Monado	A Linux runtime implementing the OpenXR specification in development by Collabora.	Linux

Table 3. Description of VR runtimes still in active development at time of writing.

From a user perspective, using a VR runtime typically involves using an executable installer provided by the manufacturer of the VR hardware used. This installer carries out the download, installation and configuration of up-to-date versions of necessary libraries, drivers and applications (such as device management and calibration GUIs). From a preservation perspective, the behind-the-scenes nature of this installation process, in addition to the frequency of updates, makes collecting and archiving specific runtime versions challenging. The only way we have identified to ensure all elements of a runtime are captured is to carry out imaging of the entire software environment (see 5.1. Acquiring VR Artworks).

We identified a number of roles that the VR runtime plays in the process of running a VR application. One role is to prepare frames to be sent to the HMD. As HMDs contain thick lenses in front of the panels to maximise the field of view at close viewing distance, frames sent to the HMD must be pre-distorted to compensate for distortion these lenses introduce. Techniques can be used to improve the quality of the distorted image. Supersampling (i.e. rendering at a higher resolution and downscaling) can be used to further compensate for the loss of pixel data in the outer edges of the rendered frame as a result of the distortion correction process. Lens matched shading increases resolution consistency by rendering an image which more closely matches the distorted images sent to the HMD[16]. Lens distortion processes are typically handled by a VR runtime specific to the HMD manufacturer and will use a distortion algorithm specific to the lens shape in the HMD which may not be made public. This can create a strong dependency relationship between a VR application and a manufacturer's runtime. Techniques have been developed to derive distortion algorithms manually using photography[17]. The VR runtime also manages the tracking system and supports a visual representation of a boundary at the limits of safe physical space. This and other features can be configured and calibrated by the user via a management application.

Additional proprietary techniques can be implemented by the VR runtime to improve rendering performance. Reprojection techniques (sometimes known by terms such as timewarp and spacewarp) are used by the current generation of VR runtimes to artificially increase framerate when a host system is unable to generate frames at the rate required to avoid discomfort in VR[18]. These techniques generate extra frames by distorting the previously rendered frame based on the movement of the user and the scene, offering a very efficient way to generate additional frames. As these techniques are used simply to achieve higher framerates, and are not apparent to the user, they do not seem to be an important characteristic of VR runtimes from a preservation perspective. They could potentially be replaced by other software providing similar functionality or may simply become unnecessary over time due to advances in computer power. However, they could remain relevant in emulation, where an original

[16] NVIDIA, *VRWorks - Lens Matched Shading*, n.d., https://developer.nvidia.com/vrworks/graphics/lensmatchedshading.
[17] OpenHMD, *Universal Distortion Shader*, 2017, https://github.com/OpenHMD/OpenHMD/wiki/Universal-Distortion-Shader.
[18] David Heaney, *VR Timewarp, Spacewarp, Reprojection, And Motion Smoothing Explained*, 2019, https://uploadvr.com/reprojection-explained/.

software environment would still be used and artificially increasing frame rates may still be a concern.

VR runtimes are a source of concern in the preservation of VR artworks due to their proprietary, hardware-specific nature and critical role in providing applications with access to that hardware. Without the installation of an appropriate runtime on the host computer system, VR hardware may function in a limited way (e.g. without appropriate lens distortion correction) or not at all. Adding a further layer of complexity, the VR application must also have support for the runtime built in. Adding support for a specific VR runtime to a VR application requires adding support for the appropriate API during development. In the Unity and Unreal Engine 4 game engines, discussed further in 3.1. Real-Time 3D VR Production Materials, this involves installing or enabling certain plugins or SDKs. From the perspective of an artist who wants to utilise a specific set of VR hardware, the use of hardware-specific runtimes means that they must build this support into their application during development, or it will not be available to the packaged application. This may greatly restrict the use of other VR hardware in the future with that application, without modifying the source project.

The development and adoption of open VR runtime standards may help alleviate this problem. An open-specification runtime standard has been developed by the Khronos Group called OpenXR[19], which aims to standardise the connections between VR applications and VR runtimes, and VR runtimes and VR hardware (see Figure 4 below). These are considered distinct goals, thus allowing manufacturer-specific VR runtimes a continued place within the VR software ecosystem, providing they can speak the language of OpenXR.

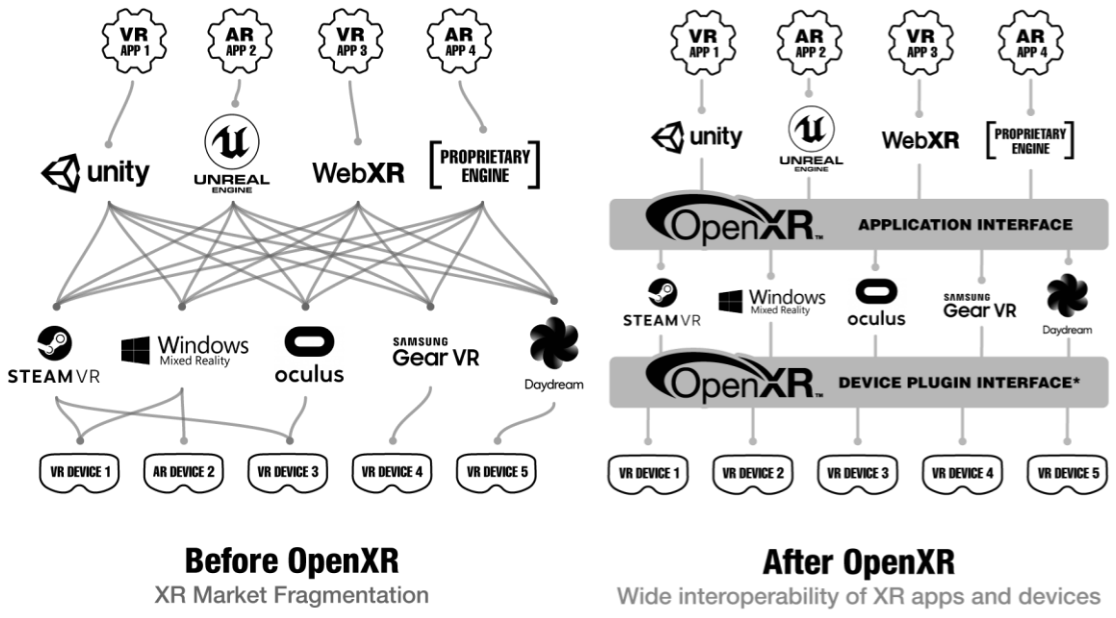

Figure 4. Diagram of current XR market fragmentation (left) and interoperability hoped to be created by OpenXR standard (right). Image credit: Khronos Group, 2019.

Adoption of such a standard by hardware manufacturers would be advantageous to artists, as it would allow them to support a range of hardware with just one plugin. It would also be advantageous from a preservation standpoint by opening up more options for hardware

[19] Khrono Group, *OpenXR Overview*, n.d., https://www.khronos.org/openxr.

migration and other interventions — topics we discuss further in 5.5. Code Migration and Related Approaches. The OpenXR 1.0 specification was released in July 2019, and currently has several public runtime implementations including Oculus[20] (runtime v19 or later), Monado[21] for Linux (which is open source) and Windows Mixed Reality[22]. On the engine side, support has been added to Unreal Engine since version 4.23 through an OpenXR plugin, while Unity has plans to support OpenXR[23].

2.2.2. Operating Systems

An operating system (OS), as for other kinds of software application, is responsible for managing the execution of a VR application and software access to hardware. It may also provide access to generic libraries and drivers (e.g. for audio and bluetooth devices) required by some VR hardware and software. As a result of the necessary use of such components, VR applications are typically built to support a particular OS and will not function on others unless recompiled from the source project. For VR applications we identified two particularly important components which are provided by the OS. The first is the implementation of a 3D API, which is called upon by a VR application to render 3D graphics. OS level support for 3D APIs is summarised in 2.2.3. 3D APIs. The second is supporting direct mode access to VR hardware, which considerably improves user experience when setting up an HMD. This is discussed further in 2.2.4. VR Device Drivers.

2.2.3. 3D APIs

3D Application Programming Interfaces (APIs) are an abstraction layer used to provide applications with access to functionality related to rendering 3D graphics. They are typically implemented in part by the operating system and in part by the GPU driver software, both of which must be present for an application to be able to use that particular API with that GPU hardware. 3D APIs are a ubiquitous component of modern VR development and real-time 3D development more generally. An application must be built to support a particular 3D API. A list of currently maintained 3D APIs can be found in Table 4.

3D API	Operating System Implementations	Description
Direct3D	Windows 10, Linux (limited support via Wine)	Proprietary API that has been part of Windows OS's since Windows 95, as part of DirectX. Support on other platforms is limited to Wine's implementation for Linux which is not complete.
OpenGL	Windows 10, MacOS (deprecated in	Open standard for 3D APIs implemented on many platforms (and on mobile through the

[20] Oculus, *OpenXR Support for PC Development*, web page, n.d. https://developer.oculus.com/documentation/native/pc/dg-openxr/.
[21] Monado, *Monado - XR Runtime (XRT)*, n.d., https://monado.freedesktop.org/.
[22] Sean Endicott, *OpenXR now available on the Microsoft Store for Windows Mixed Reality*, 2019, https://www.windowscentral.com/openxr-now-available-microsoft-store-windows-mixed-reality.
[23] mfuad, *Unity's plans for OpenXR*, 2018, https://forum.unity.com/threads/unitys-plans-for-openxr.993225/.

	10.14), Linux	OpenGL ES variant). Developed and maintained by Khronos Group.
OpenGL for Embedded Systems / OpenGL ES	Windows, Linux, Android	Open standard for 3D APIs on embedded devices and portables such as mobile phones. Developed and maintained by Khronos Group.
Metal	MacOS	Apple's low-level 3D API intended as a competitor to Direct3D 12 and Vulkan.
Vulkan	Windows 10, Linux	Successor to OpenGL developed by the Khronos Group, this open standard is a low-level API designed to compete with Direct3D 12 and Metal.

Table 4. List of currently maintained 3D APIs and the operating systems they have been implemented on.

In their core features, the current generation of 3D APIs described in Table 4 are relatively similar[24]. This is reflected in the ability of modern real-time 3D engines to target a variety of APIs when building applications. Indeed, in current versions of Unreal Engine and Unity, applications can be built to support any of the APIs listed in Table 4. For Unity, the separation of the custom application content and the Unity player component means that an application can be executed using any of the APIs supported by the standard player component — the choice of API to use is then automatically chosen based on availability or controlled using launch options.

Despite similarity in core features, there remains the potential for the choice of 3D API to introduce variability in rendering characteristics. For example, the real-time ray tracing rendering technique is currently supported only in Microsoft's DirectX version 12[25] and Vulkan[26]. The choice of API is also significant in terms of the potential loss of API support from future GPU driver and operating system updates. There are indications that changes of this kind happen in practice, a notable example being Apple's choice to deprecate the use of OpenGL 3D rendering API on MacOS in favour of their own Metal API[27]. There are also historical examples of APIs becoming obsolete, such as the Glide API created by the graphics card manufacturer 3dfx and widely used in the late 1990s, but eventually being superseded by Direct3D and OpenGL[28]. Ensuring support for open standards (such as OpenGL and Vulkan) as opposed to proprietary, platform-specific APIs (such as Direct3D and Metal) when building VR applications may lower the risk of losing access through obsolescence, as an open specification makes writing compatibility software more practical. 3D APIs, being a combined product of operating system and GPU drivers, are made available to an application by ensuring compatible versions of these two components are present.

[24] Alain Galvan, *A Comparison of Modern Graphics APIs*, 2020, https://alain.xyz/blog/comparison-of-modern-graphics-apis.
[25] Shawn Hargreaves, *Announcing DirectX 12 Ultimate*, 2020, https://devblogs.microsoft.com/directx/announcing-directx-12-ultimate/.
[26] Khronos Group, *Ray Tracing In Vulkan*, 2020, https://www.khronos.org/blog/ray-tracing-in-vulkan.
[27] Samuel Axon, *The end of OpenGL support, plus other updates Apple didn't share at the keynote*, 2018, https://arstechnica.com/gadgets/2018/06/the-end-of-opengl-support-other-updates-apple-didnt-share-at-the-keynote/.
[28] Tony Smith, *3dfx open sources Glide, Voodoo 2 and 3 specs*, 1999, https://www.theregister.com/1999/12/07/3dfx_open_sources_glide_voodoo/.

2.2.4. VR Device Drivers

VR device drivers are specialised software programs which communicate with hardware devices. All VR hardware devices require a suitable driver to be present in order to function, whether it be manufacturer-supplied or a generic driver packaged with the OS. There is limited information available about the role of these drivers, which for most consumer-oriented VR hardware (as typically used by artists) are proprietary and closed-source. What we do know is largely a result of the efforts of the open source community to create open-source drivers, such as OpenHMD[29], libsurvive[30] and Monado[31]. These projects have demonstrated that many core elements of manufacturer supplied driver functionality can be reimplemented in open-source software and their documentation is a useful resource for learning more about VR hardware drivers. One important feature of device drivers is management of access to the HMD. Since the arrival of consumer-oriented VR HMDs (namely the Oculus Rift CV1 and HTC Vive), HMD drivers implement a feature known as *direct mode*. This allows an application to treat an HMD as a dedicated and distinct display device, rather than as an additional monitor which would extend the desktop[32]. The latter is known as *extended mode*, and is now rarely used due to the amount of configuration required and the improved user experience provided by direct mode. At time of writing, extended mode remains accessible in current versions of SteamVR but has been removed from the Oculus runtime.

From a user perspective, there is little distinction in the installation and configuration process between VR device drivers and the VR runtime, which carries out this process behind-the-scenes. From examination of SteamVR on Windows, we observed that it uses both installed system drivers, managed by the OS, and DLL drivers which are loaded and managed by the runtime. These factors make extracting drivers for storage independent of a runtime environment impractical. As for other components of the software environment, they are likely to be best preserved as part of a captured disk image (see 5.1. Acquiring VR Artworks).

2.2.5. GPU Drivers

GPU drivers allow software to interact with specialised graphics processing hardware known as a Graphics Processing Unit (GPU) or graphics card. Beyond supporting 3D rendering capabilities, GPU drivers implement device-specific support for direct mode VR (for example, NVIDIA added support for Oculus direct mode in 355.83, HTC Vive direct mode in 361.75 and Windows Mixed Reality in 387). Drivers also form a part of the implementation of different 3D APIs. The importance of the GPU driver is a concern for long-term preservation, as making changes to GPU hardware will likely necessitate a change to the GPU driver, potentially resulting in loss of access to rendering features, VR direct mode and 3D APIs.

[29] OpenHMD, web page, n.d., http://www.openhmd.net/.
[30] libsurvive, GitHub repository, 2020, https://github.com/cntools/libsurvive.
[31] Monado, *Monado - XR Runtime (XRT)*, n.d., https://monado.freedesktop.org/.
[32] Monado, *What is Direct Mode*, n.d.,https://monado.freedesktop.org/direct-mode.html.

3. Real-Time 3D VR

Real-time 3D (RT3D) is the use of software to dynamically generate a moving image from 3D data. In contrast to pre-rendered 3D, which uses similar data sources, the real-time nature of the process means that the moving image sequence can be dynamic and respond to user interaction. This technology has been used extensively in the video game industry and has found use in contemporary art — there are five artworks in the Tate collection which use RT3D. Alongside 360 video, RT3D is one of two main approaches that can be taken to producing VR content. RT3D VR can be considered an extension of real-time 3D rendering, as applications are built using the same tools and underlying principles. RT3D VR content produced by artists is ultimately compiled as a software application, a package of code and data which is executable on a suitable computer system.

In order to understand how to preserve RT3D VR applications, we need to understand how they are created and how they function. In this section, we will start by introducing the production process for real-time 3D applications, with a focus on the engine-based approach taken by the artists we interviewed. Given evidence supporting the value of source materials in the conservation of software-based art[33,34], we will also consider the practical implications of collecting production materials associated with VR applications. In the next section, we will examine the compiled software applications (sometimes known as *builds*) which are the primary output of the production process. In the final section, we will explore approaches to the creation of conservation documentation for real-time 3D VR applications.

3.1. Real-Time 3D VR Production Materials

The process of producing RT3D VR artworks involves bringing together an array of *assets* — various data types including 3D models, textures and audio — in a *3D engine*. A 3D engine is a development environment for creating real-time 3D content, which can then be exported (or *built*) as an executable application. This section examines the production process in further detail, and identifies the key software tools and data sources involved.

3.1.1. Engines and Project Files

Real-time 3D engines are development environments for creating real-time 3D software. They can integrate a broad range of features useful in constructing 3D environments and typically have a graphical user interface (GUI) built around the manipulation of 3D scenes through a viewport (see Figure 5). Features included in a contemporary game engine may include:

[33] Deena Engel and Glenn Wharton, *Reading between the Lines: Source Code Documentation as a Conservation Strategy for Software-Based Art*, *Studies in Conservation* 59 (6): 404–15, 2014, https://doi.org/10.1179/2047058413Y.0000000115.
[34] Mark Hellar and Deena Engel, Computational Provenance and Computational Reproducibility: What Can We Learn about the Conservation of Software Art from Current Research in the Sciences?, Electronic Media Review 4, 2015, https://resources.culturalheritage.org/emg-review/volume-4-2015-2016/engel-hellar/.

- **Renderer:** Generates animated 3D graphics in real-time from the assembled data sources (geometry, materials, lighting and particle systems etc.).
- **Shader management:** Enables authoring of shaders (small rendering programs which run on the GPU) and translation[35] into 3D API specific languages such as HLSL (Direct3D), GLSL (OpenGL) and SPIR-V (Vulkan)
- **Scripting:** Allows creation of dynamic or interactive behaviours using programming languages or equivalent interfaces (e.g. the Blueprints graph editor in Unreal Engine 4).
- **Physics simulation:** Simulates an approximation of physical systems and the resulting interaction between geometry components.
- **Audio engine:** Allows playback and manipulation of audio, including positional and spatial components.
- **Asset management:** System for the import, export and organisation of assets such as textures, 3D models and audio files.
- **Cross-platform build support:** Allows compilation of projects to binaries for different platforms, including various operating systems, 3D APIs and VR runtimes.

Much like VR systems, real-time 3D engines have been developed primarily in the context of video game production. The use of third-party engines is advantageous because many core features of an engine can be reused: there is little value to engaging in the lengthy process of implementing a 3D renderer or physics engine when an existing implementation can meet your requirements and allow focus on the more creative elements of the production process.

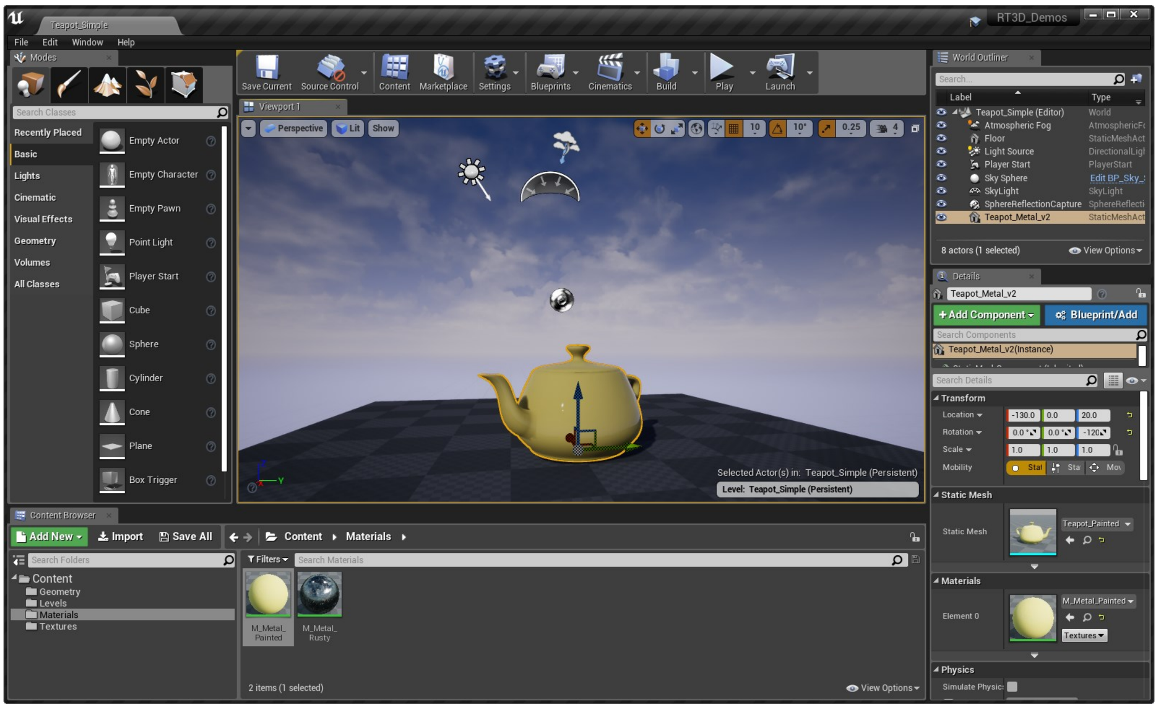

Figure 5. A screenshot of a simple 3D scene in the Unreal Engine 4.22 editor software.

[35] Where compilation or interpretation of shader code is handled by the GPU at runtime.

The GUI-based editors (see Figure 5) distributed with engines lower the barriers to entry, allowing those without a background in 3D rendering techniques and programming to develop real-time 3D software. These factors make third-party engines particularly appealing to artists — the six that we spoke during this research worked exclusively with the third-party engines Unity and Unreal Engine 4.

While larger game studios might develop their own engine, many will licence a third-party engine such as the aforementioned Unity and Unreal Engine 4 (UE4)[36]. Unity and UE4 are both free to use software packages which employ revenue-based licensing models. Unity uses a tiered subscription model which requires users with revenue or funding of over $100,000 over the past 12 months to buy a paid plan, which rises in cost in several further tiers based on higher revenue levels[37]. UE4 uses a licencing model which requires users to pay a 5% royalty to Epic Games if their product is monetised and revenues exceed $1,000,000 USD[38]. Although they weren't used by the artists we spoke to, there are alternative RT3D development tools available under open-source, permissive licenses, including the engine Godot[39] and web frameworks A-Frame[40] and Three.js[41].

Applications are developed in engines as projects: collections of data and code arranged in a well-defined folder structure. Encouraging the preservation of the source projects of real-time 3D applications seems a logical approach to their long-term preservation, as these are analogous to source code. Retaining these opens up options of modification and migration in order to keep the software running in future technical environments (see 5.5. Code Migration and Related Approaches). The Unity and Unreal Engine 4 engine project formats exist within a single directory, which can be archived as-is to capture the hierarchy of assets, levels and other materials. However, they remain contingent on the appropriate version of the engine binaries for access — opening them in another version may cause damage to the project, failure to compile or other compatibility issues.

Reliance on engine binaries presents several challenges for ongoing access to engine projects:

- Engines are typically managed by installer applications which conceal details of the installation and configuration process.
- Engine projects may also have additional dependencies on third-party pieces of software in order to build for certain platforms (e.g. Google's Android SDK must be installed to create Android builds) or on plugins.
- Engine updates are relatively frequent and there is a tendency to remove access to old engine binaries over time. For example, an average of four major versions of Unreal

[36] Limited data is available to make meaningful estimates as to how many, but this survey of data about games on the Steam distribution platform provides some indication: https://www.reddit.com/r/gamedev/comments/8s20qp/i_researched_the_market_share_of_game_engines_on/.
[37] Unity, *Compare Unity plans*, 2021, https://store.unity.com/compare-plans.
[38] Unreal Engine, *Frequently Asked Questions*, 2021, https://www.unrealengine.com/en-US/faq.
[39] Godot Engine, web page, 2021, https://godotengine.org/.
[40] A-Frame, web page, 2021, https://aframe.io/.
[41] Three.js, web page, 2021, https://threejs.org/.

Engine have been released every year since 2014. At the time of writing versions of Unreal Engine binaries prior to version 4.0.2 (released 28 March 2014) are no longer publicly accessible, while accessible Unity binaries extend back as far as version 3.4.0 (released 26 July 2011).

It is worth noting that even if a source project is archived with suitable engine binaries, without archiving complete engine source code we have only a partial representation of the software. Both UE4 and Unity have accessible source code repositories but in neither case is the full source code made available under an open-source licence. A partial C# component of the Unity source code is publicly accessible on GitHub but does not allow modification or redistribution[42]. Full source access and modification rights are only available with a higher tier paid subscription and an individually negotiated source access agreement. The full UE4 source code can be read and modified by anyone agreeing to the Unreal Engine EULA[43] but not redistributed in source code form.

3.1.2. Scenes and Assets

The term asset is used to describe a unit of data which is imported into or created in a game engine, and used in the construction of a virtual scene (also known as a level, map or scenegraph). Assets can include a huge range of file types, including meshes (3D geometry), textures, materials, particle systems and audio. The creation of assets may involve various specialised workflows and tools outside the engine, from which a suitable interchange format is exported and used to import them into the engine. Based on examination of source projects during this research, files appear to retain their original format when imported into Unity but are converted to a native format when imported into Unreal Engine 4. We have not been able to locate public documentation of the latter format. The variety of asset types is such that we limited the scope of our research and the following discussion to examining the most unique of these to 3D rendering: the 3D model. In this section we will discuss what constitutes an engine-ready 3D model, which we define as a virtual representation of an object, and identify common file formats used.

At its most basic, a 3D model is a set of point coordinates or *vertices*[44], which describe the structure of the surface of a 3D object, known as a mesh. Each vertex can have properties, such as a *normal*, which describes the direction it is facing (i.e. the outside of the surface), and also include UV map coordinates which describe how textures are projected over the model's surface. A 3D model can also be associated with descriptions of its surface properties, known as *materials*, which are used by the engine to light and render it. Historically this might have been as simple as a square raster image tiled over the surface of the model to describe its colour, but in modern 3D rendering can include many layers of information that are used to achieve physically accurate results when the surface is lit. This information is typically represented using texture *maps*, raster images which contain information describing a particular property such as how rough or smooth a surface is. This approach to creating

[42] Aras Pranckevičius, *Releasing the Unity C# source code*, 2018, https://blogs.unity3d.com/2018/03/26/releasing-the-unity-c-source-code/.
[43] Unreal Engine, *Unreal Engine End User License Agreement For Creators*, 2021, https://www.unrealengine.com/en-US/eula/creators.
[44] The singular form of vertices is *vertex*.

materials is known as *physically-based rendering* (PBR) and uses texture maps describing color (also known as a diffuse or albedo map), surface detail (known as normal maps), shadowing (ambient occlusion maps), roughness (or in some workflows glossiness), metallicness and specularity.

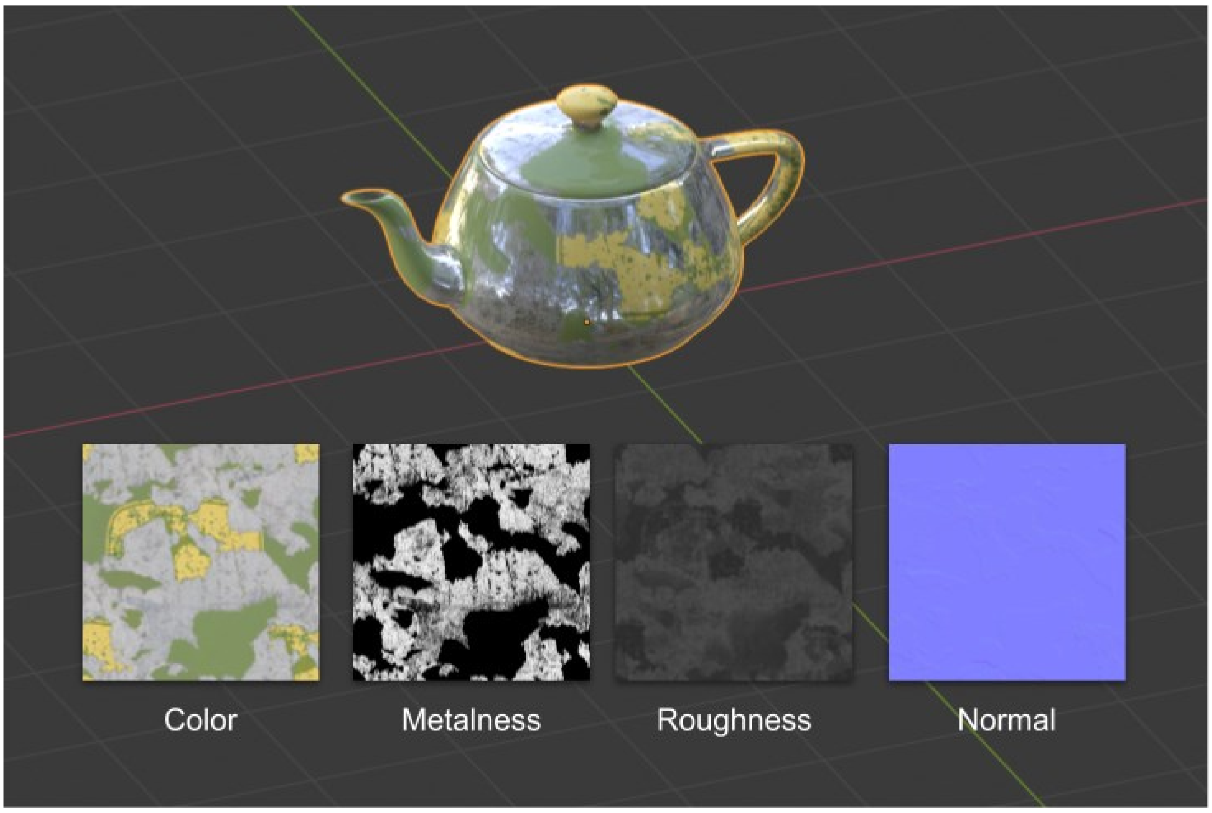

Figure 6. These four texture maps have been used to apply a PBR material to a mesh in the 3D modelling software Blender 2.8. Material textures downloaded from freepbr.com.

In combination, these maps allow the renderer to infer how light would bounce off that surface, and thus determine how to colour the pixels that represent that surface for a particular frame. Depending on the file format and workflow adopted, materials might be packaged with the geometry data or included as a sidecar file.

There is an array of tools available to create engine-ready 3D models, with popular tools that we encountered including 3D Studio Max, Blender (the primary free and open source option in this domain), Houdini (which is specialised in procedural animation), Maya and ZBrush (which is specialised in sculpting). In addition to their own native formats, these tools are capable of importing and exporting 3D models in a variety of file formats in order to accommodate varied production workflows. Additional tools may be used in texturing workflows beyond widely used raster graphics editing tools like Photoshop and GIMP, such as the Substance toolset which is used for the creation of PBR materials. A 3D model suitable for usage in real-time 3D applications, particularly in an VR context, may be heavily *optimised* in order to improve performance. This typically involves reducing the complexity of geometry and lowering the resolution of texture maps.

Given the relative novelty of 3D model file formats as a digital preservation research topic, we are not yet at a stage where community consensus of preservation suitable formats has been reached, and a detailed comparison in the context of our research was beyond the scope of this project. The best resource at time of writing is the Library of Congress' 'Sustainability of Digital Formats' website[45], which includes a number of in-depth 3D file formats as part of an ongoing programme of reviews. Fragmentation in requirements from different user groups, and formats favoured by particular tools, means that 3D model formats are heterogenous and often domain specific. The more generic formats with support across many applications — of which FBX, a proprietary format owned by Autodesk, is particularly favoured in real-time 3D production — are so-called interchange formats, which often contain only partial representations of the content created in the authoring tool and require a degree of processing and configuration when imported into another tool. While there have been efforts to introduce open standards for 3D model file formats (for example, the COLLADA, U3D and X3D formats), adoption of these has been poor, perhaps due to the inherent challenge of keeping up with the fast-moving nature of the 3D graphics industry.

In Table 5 below we offer a brief summary of selected file formats, which we identified as noteworthy during our research by their matching one or more of two criteria: 1) those that are frequently used in real-time 3D rendering applications and 2) those which are open standards. For each we give a brief description of the format, some preliminary notes on issues relating to their suitability for long-term preservation and, where applicable, point to their review in the Library of Congress' 'Sustainability of Digital Formats' resource.

Format	**Description**	**Preservation Notes**
Wavefront OBJ/MTL	OBJ is a simple interchange format for 3D geometry. An OBJ file can be accompanied by an MTL file describing material properties.	While the most feature limited of the formats listed here (e.g. no support for animation), OBJ has a track record of long-term support across many applications. While the specification is public the licence it is released under is unclear and may be proprietary. **LoC sustainability review: https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml**
FBX	FBX is a proprietary format maintained by Autodesk and widely used as a 3D model interchange format in real-time 3D rendering applications.	Widely adopted in real-time 3D engine workflows. The binary format has been partially reverse engineered by Blender Foundation for their own import/export tools, but its proprietary nature remains a concern for long-term preservation.
X3D	X3D is a royalty-free open standard which can represent 3D objects and scenes. The format is an ISO standard and has been in development by the Web3D consortium since the early 2000s.	The format is not widely used in RT3D applications (neither Unity nor Unreal Engine 4 include X3D importers). Release of new versions of the specification is infrequent (the last ratified version, 3.3, was released in 2015) which means its features are out of step with those of other formats such

[45] Library of Congress, *Sustainability of Digital Formats: Planning for Library of Congress Collections*, 2021, https://www.loc.gov/preservation/digital/formats/.

		as FBX and glTF[46]. **LoC sustainability review:** **https://www.loc.gov/preservation/digital/formats/fdd/fdd000490.shtml**
glTF	glTF is a royalty-free open standard designed for the efficient transmission of 3D models and scenes. It is maintained by the Khronos Group.	Primarily aimed at the web, adoption for real-time 3D applications seems likely if web-based RT3D continues to gain popularity. Tools supporting import/export are currently limited however. Monitoring will be required given priorities regarding transmission efficiency and the potential for introduction of lossy compression features (e.g. Draco[47]). **LoC sustainability review:** **http://www.loc.gov/preservation/digital/formats/fdd/fdd000498.shtml**
Alembic	The Alembic (.abc) format is a royalty-free open standard used to store complex animated 3D geometry. It is maintained by Sony Pictures Imageworks and Industrial Light & Magic.	This format is widely used in RT3D rendering and importing is natively supported in both Unity and Unreal Engine 4. As it only includes only the 'baked' animation, it is not lossless with the respect to complex rendergraph present in source projects (e.g. in Houdini).
Universal Scene Description (USD)	A royalty-free open-standard designed as an interchange format for complete 3D scenes composed of many elements. Maintained by Pixar.	Extent of adoption in RT3D is not known, but both Unreal Engine 4 and Unity support import of USD scenes.

Table 5. Summary of RT3D relevant 3D file formats identified during our research.

3.2. Real-Time 3D VR Applications

RT3D VR applications (or builds) are the software that results from the development process described above, and is then used in the display of RT3D VR artworks. These applications are dependent on the software and hardware layers of a VR system, as described in 2. VR Systems. In this section, we discuss what constitutes a RT3D VR application and consider how they might be acquired.

3.2.1. Application Packages

An application is produced from a 3D engine project (see 3.1.1. Engines and Project Files) and usually contains a combination of executable code and packaged data arranged in a well-defined directory structure. In most cases this will be the primary media used in the display of

[46] Leonard Daly, *glTF/X3D Comparison*, n.d., https://realism.com/blog/gltf-x3d-comparison.
[47] Draco, GitHub repository, 2021, https://github.com/google/draco.

the work and can be seen as analogous to a master. A RT3D VR application (excluding one targeting WebGL — see notes below) will generally consist of:

- One or more executable files (e.g. Windows Portable Executable on Windows, Mach-O on MacOS).
- Additional runtime libraries (e.g. Mono Runtime for Unity or the Ogg Vorbis audio decoder library).
- Data files containing packaged assets (e.g. .pak files in Unreal Engine 4 and .resource files in Unity)

The code component of the executable files is machine code compiled for the target platform, in order to maximise performance. Despite commonalities among CPU architectures on contemporary computing platforms (Windows, MacOS and Linux all utilise the x86-64 architecture), this low-level code remains platform specific due to the use of system calls and other OS-specific references. Modern game engines, including Unity and Unreal Engine 4, allow cross-compiling for the creation of builds for different target platforms (e.g. Windows, MacOS, Linux, Android) from the same source project. While a build can only support one platform, it can support multiple 3D APIs and VR runtimes, providing these are supported by that host platform. Support for these will depend on the engine version and plugins in use when the application is built.

RT3D applications built for the web use a different set of technologies from the standalone application builds described above, so that they can be run in a web browser. The artists we worked with during our research were not using these technologies, so our coverage of this area of RT3D VR is limited to a very broad overview. Web RT3D applications are built as code which can be executed by a web browser at runtime, such as JavaScript or WebAssembly, and use the WebGL 3D API instead of desktop APIs such as OpenGL and DirectX. The application package itself is also divergent from native RT3D builds in that its structure more closely resembles a website. Assets are typically not packaged and are instead stored in web-friendly file formats such as OBJ and glTF, while code is stored in a human-readable format rather than a binary format (excluding WebAssembly code).

3.2.2. Executing an Application

Executing a RT3D VR application engages the layers of hardware and software described in 2. VR Systems. The specification and features of these components can impact the characteristics of the VR experience in various ways such as frame rate, latency and image quality. In Table 6 below we briefly describe some of the key points of variability we have identified alongside notes on approaches to recording them. Monitoring these characteristics particularly could be useful in identifying problems with VR application performance or when assessing the impact of changes made to a VR system in the course of applying preservation techniques (e.g. migration or emulation). Thi summary is intended to offer a starting point for analysis of RT3D VR application execution; fully understanding the significance, documentation, and management of these characteristics will require further research.

Characteristic	Description	Measurement Approach
Resolution	Resolution describes the number of pixels rendered in each direction per frame (e.g. 1080x1920 or 2160×1200).	This usually matches the native resolution of the target display device (i.e. HMD). Can be verified using capture software or hardware.
Bit-depth	Bit-depth describes the possible range of values in which color/luminance can be expressed. Real-time 3D applications can support 8-bit or 10-bit per channel output, and can use the full dynamic range of 0-255 (or 0-1023 for 10-bit) when connected to a suitable display device.	Usually controlled with GPU driver utilities (e.g. NVIDIA Control Panel) but involves compatible OS components and suitable display equipment to be applied. Capture cards may provide an independent means of verification.
Colour Space	The colour gamut, gamma and white point for output frames. For real-time 3D applications this is usually sRGB, which can be converted by TV equipment to the similar Rec. 709.	Usually controlled with GPU driver utilities (e.g. NVIDIA Control Panel) but involves compatible OS components and suitable display equipment to be applied. Capture cards may provide an independent means of verification.
Frame rate / Frame time	Measurements of the speed with which frames are generated. Frame rate describes the number of frames created per second, while frame time is the amount of time taken to generate a frame.	It is not clear how existing RT3D monitoring tools such as MSI Afterburner / RivaTuner Statistics Server interact with VR runtimes. Runtime specific tools such as Oculus Profiler and SteamVR Frame Timing Tool should be used instead.
Latency	Measurement of the time taken for physical input to be translated into output frames.	We are not aware of any tools for measuring this for an arbitrary RT3D VR application, at time of writing.
3D API Feature Level	The set of shading related features supported by the 3D API, GPU and driver set, and targeted by a RT3D application.	We are not aware of any tools for identifying feature level support for an arbitrary RT3D VR application. Must be identified by examining the source project.
Internal Timing	The clock according to which events unfold within the simulation of the virtual environment.	Can only be identified from examining program code. This may cause issues where events are not coded to be framerate independent and thus become locked to processing speed[48].

Table 6. List of performance and rendering characteristics and approaches to their analysis and documentation.

[48] Construct, *Delta-time and framerate independence*, 2017, https://www.construct.net/en/tutorials/delta-time-and-framerate-independence-2

When installed on a system or run for the first time, additional local files may be created by a VR application to store configuration and saved state information. These may be stored within the application directory or elsewhere on the host system; identifying locations may require the use of system call tracing to identify write operations made by the software during execution[49].

3.3. Real-Time 3D VR Documentation

Given the large number of hardware and software components, the separation between user experience and external environment, and the interactive nature of the medium, documentation is a potentially expansive topic in relation to real-time 3D VR. Our research has only really scratched the surface in terms of identifying how we might approach these novel documentation requirements. It seems likely that in the short term many existing tools and approaches from the fields of art conservation and digital preservation will be suitable to guide aspects of the documentation process. In particular, there is a well-established precedent for documenting technically complex, installation-based artworks in time-based media conservation[50,51,52]. We feel that approaches which have already been adapted for software-based artworks[53,54] are particularly likely to be suitable with relatively little modification. This is due to the similarity of the core components encountered in VR artworks — such as computer systems, display hardware and custom software — and their manifestation in artwork characteristics. In this section we will briefly consider extensions required to existing methods that we identified during this research. This is not exhaustive and should be considered a starting point for further research.

3.3.1. Acquisition Information Template

In the first phase of our research, we identified that, as for any other artwork acquisition, we would need to gather sufficient information about VR artworks to make effective decisions about bringing them into a collection, the archiving of relevant components and planning for its long-term preservation. Given the number of medium specific questions which arose in scoping this, we identified a need for a tool to guide the information gathering process at the early stages of acquisition and so developed a document template for this purpose, partly informed by Tate's acquisition template. This is designed to be completed by or in close collaboration with an artist prior to receiving media from the artist. We tested the template by inviting the artists we interviewed to complete it for a specific artwork. Version 1.00 of the

[49] An example of this approach can be found in: Tom Ensom, 'Revealing Hidden Processes: Instrumentation and Reverse Engineering in the Conservation of Software-Based Art', *Electronic Media Review* 5, 2018, https://resources.culturalheritage.org/emg-review/volume-5-2017-2018/ensom/.

[50] Matters in Media Art, web page, 2015, http://mattersinmediaart.org/

[51] Guggenheim, *Time-Based Media*, n.d., https://www.guggenheim.org/conservation/time-based-media.

[52] Smithsonian, *Time-based Media & Digital Art*, n.d., https://www.si.edu/tbma/.

[53] Tate, *Software-based Art Preservation – Project*, 2021, https://www.tate.org.uk/about-us/projects/software-based-art-preservation.

[54] Guggenheim, *The Conserving Computer-Based Art Initiative*, n.d., https://www.guggenheim.org/conservation/the-conserving-computer-based-art-initiative.

template is available on Tate's Preserving Immersive Media project page[55]. Further enhancement of the template has been carried out by Savannah Campbell and Mark Hellar and was presented at AIC 2019[56].

3.3.2. User-Perspective Video Capture

An effective way of capturing the experience of interacting with a real-time 3D VR artwork in a software-independent way is to capture the perspective of the user as video. User-perspective video capture then, is the recording of moving image frames as perceived by the user of the VR system through, for example, an HMD. We identified two approaches that can be taken to this kind of capture: fixed-view and 360 degree. Fixed-view video is captured from the perspective of the virtual camera as the user moves. Both rotational and positional movements become fixed to those that were carried out during the period of recording. This kind of video can be captured before or after the compositor carries out VR runtime specific processing. Pre-compositor capture is undistorted and without frame interpolation, while post-compositor capture will include the effects of these runtime processes.

An alternative approach is to capture 360-degree video from the perspective of the user, which captures the entirety of the users surroundings and so allows rotational tracking to be maintained in the resulting media. This is interesting not only as a documentation technique, but as a preservation approach, as it can offer a video-based surrogate for RT3D VR experiences which do not use positional tracking (also known as 'on-rails'). This 360 video version does not have the fixed dependency on a specific real-time 3D rendering technology, as the output of the capture process is simply video data which can be played back in a variety of players (see 4. 360 Video). Features for capturing 360 video from real-time 3D environments are present in both the Unreal Engine 4 and Unity editor software. Claudia Roeck has identified and tested a third-party tool for capturing 360 video in compiled UE4 and Unity real-time 3D applications called Surreal Capture[57]. The primary limitation of the 360 video capture approach is that positional tracking and other forms of interactivity beyond rotational tracking are lost. Additionally, a by-product of bypassing the normal fixed field-of-view player camera is that certain 'screen space' visual effects will not be captured (e.g. vignetting, light shafts, motion blur)[58].

Capturing both fixed-view and 360 video in a real-time 3D engine is resource intensive and can generate very large volumes of data if captured in an uncompressed form. Strain on the host system can be eased by utilising dedicated encoding hardware (e.g. capture cards or NVIDIA's NVENC feature available on some GPUs), freeing up the GPU to focus on rendering. Decisions over appropriate video encoding are similar to those when working in other video

[55] Tate, *Preserving Immersive Media – Project*, 2021, https://www.tate.org.uk/about-us/projects/preserving-immersive-media.
[56] Savannah Campbell and Mark Hellar, 'From Immersion to Acquisition: An Overview Of Virtual Reality For Time Based Media Conservators', *Electronic Media Review* 6, 2019, https://resources.culturalheritage.org/emg-review/volume-6-2019-2020/campbell/.
[57] Claudia Roeck, *Capturing a VR-executable as a 360-degree video*, forthcoming report.
[58] Gavin Costello, *Capturing Stereoscopic 360 Screenshots and Movies from Unreal Engine 4*, 2016, https://www.unrealengine.com/en-US/tech-blog/capturing-stereoscopic-360-screenshots-videos-movies-unreal-engine-4

production contexts (e.g. chroma subsampling, bitrate, compression), but care should be given to ensure encoding matches the frames output by the GPU as closely as possible for an accurate capture (e.g. colour space, bit-depth, framerate). Currently, the value of capture is impeded by the lack of transparency over where software tools actually capture frames in the rendering pipeline. Using a dedicated video capture card, capable of intercepting the GPU output, could improve this situation, but it is unclear whether such hardware could capture HMD-targeted output in this way. Further research is required to better understand how we might effectively carry out this kind work and whether available tools are fit for purpose.

Beyond capturing what a user sees, we have also identified a need to capture interaction as it occurs within the physical installation space. Understanding how interaction has occurred in the past offers contextual insight into how that work was presented and received. While photographic methods of documentation will be successful in capturing VR artwork installations to some extent, the dynamic and interactive nature of VR points towards the importance of video as an extension of this. Captured simultaneously with user-perspective documentation, this provides one way of connecting the users virtual experience with their movements in physical space. Tests during a hackathon at iPRES 2019[59] demonstrated that capture tools such as Brekel OpenVR Recorder provide one way of documenting this kind of interaction, although further research is required to understand how the outputs could be used.

[59] Tom Ensom, Jack McConchie, Dragan Espenschied and Claudia Roeck, *Understanding the Variability of Virtual Reality Artworks*, hackathon at iPres 2019, https://ipres2019.org/static/pdf/iPres2019_paper_154.pdf.

4. 360 Video

360 video is a video format in which every direction of view is available to the viewer. Though the direction of view is unlimited, in most circumstances the viewing position in space is either fixed or on a predetermined “on rails” path. In this section, we will explore how 360 video is generated from camera capture, the methods employed to generate a three dimensional experience from the resulting files, and how spatial audio can be represented.

4.1. 360 Video Production Materials

4.1.1. Camera Capture

360 video can be produced from two distinct workflows, as a capture from a real-time 3D engine (see 3.3.2. User-Perspective Video Capture) or captured from a camera or array of camera lenses. Camera capture can result in monoscopic or stereoscopic footage. Monoscopic 360 video is captured from a single point of view, creating an identical view in each eye. It is typically captured with dual fisheye lenses, arranged back-to-back, each capturing half of the 360 degree field of view. Stereoscopic 360 video employs an array of lenses to capture material from multiple viewpoints which, after processing, provides a slightly different point of view for each eye. This creates an illusion of depth and a potentially more realistic experience.

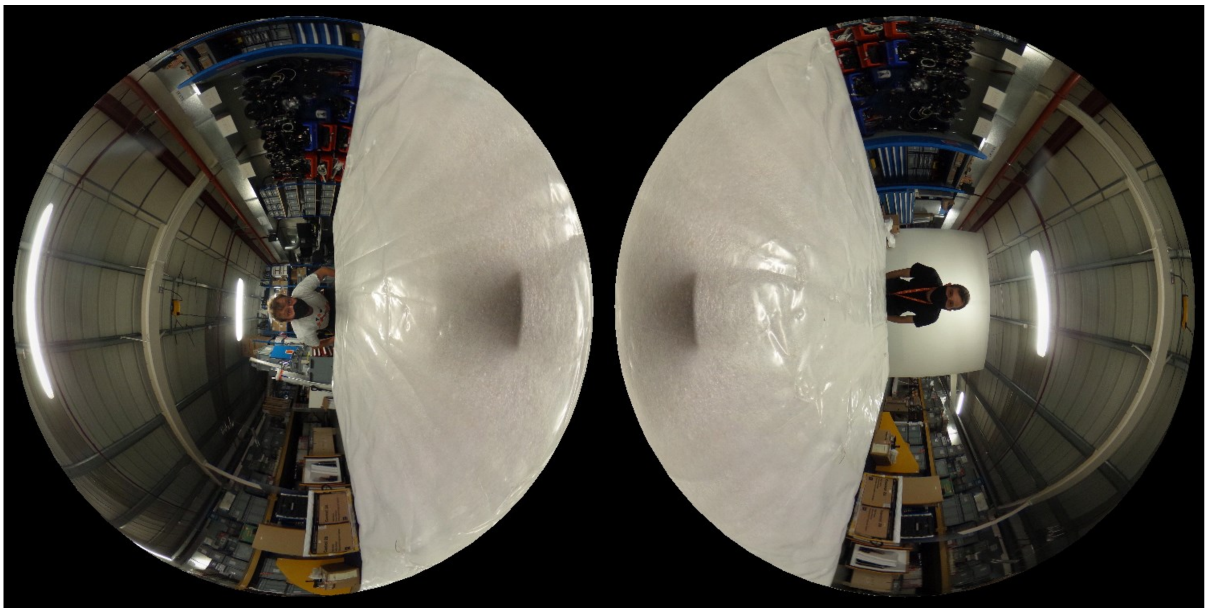

Figure 7. Image taken with a monoscopic 360 camera showing fisheye images corresponding to each lens.

4.1.2. Stitching

In both of these examples, raw video data from the camera captures have video streams that must be stitched together to produce viewable video content. There are several software programs that undertake the process of stitching, and a camera manufacturer might bundle their own proprietary software in order to undertake this. There are also third-party options available, which contain templates of commonly used lens arrays and in some cases can automatically detect camera layouts. The process of stitching involves detecting the overlapping edges of footage in order to combine them into a single file (monoscopic) or a single file for each eye (stereoscopic). Given that overlapping pixels are blended and discarded, the process of stitching is an inherently lossy one and is subject to inaccuracies that can cause artefacts. There is potential for increased accuracy of stitching algorithms as technology develops, therefore there may be a case for archiving raw camera footage alongside stitched footage for the purposes of stitching in the future. The storage requirements for this are potentially vast however, particularly in the case of multi camera stereoscopic capture due to large resolution files with high percentage of pixel overlap.

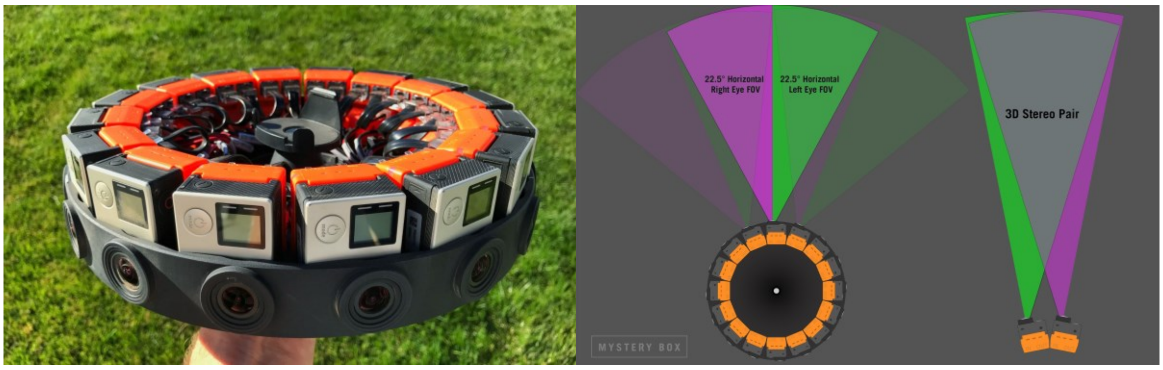

Figure 8. GoPro Odyssey multi camera rig and diagram illustrating a section of the image from each lens generating a stereoscopic image. Image credit: Mystery Box, 2017, https://www.mysterybox.us/blog/2017/1/31/shooting-360-vr-with-gopro-odyssey-and-google-jump-vr.

4.1.3. Projection Format

The video files that result from the stitching process are stored in a 2D video format (planar), with a rectangular aspect ratio. The method employed to store the 3D video information in a planar video file is referred to as the *projection format*.

360 video files are played back using specialised pieces of software known as 360 video players. These players are real-time 3D applications (see 3.2. Real-Time 3D VR Applications) capable of decoding video and mapping it onto the interior of a 3D object, the shape of which corresponds to the chosen projection format (e.g. a sphere for equirectangular or a cube for cubemap). A UV map (see 3.1.2. Scenes and Assets) or equivalent information, which is specific to this projection format, tells the 360 video player how to correctly distribute the video pixels over the interior surface of the 3D object. During playback the viewer of the 360 video is positioned at the centre of the 3D object and the mapped video is viewed from this position. Players may be standalone pieces of 3D software or may be authored using RT3D engines such as Unity or Unreal Engine.

The most common projection format is called *equirectangular* projection, familiar to us as the type of geometric distortion used to represent the surface of the earth in a 2D form. The inherent distortion of the pixels in their planar equirectangular form is referred to as curvilinear. One downside to this projection type is that the top and bottom of the image use a disproportionately large area of pixels in comparison to that of the centre of the image —

varying *pixel density.* Note in Figure 9 below, how a large proportion of the pixels are used on the ceiling and table which have relatively low detail in comparison to the central section of the image where most of the detail is. This is particularly inefficient given that the ceiling and table areas are mostly in the user's periphery vision where detail perception is minimal. This inefficiency has led to the use of *cubemap* projection, where the 360 Video file is projected onto the inside surface of a cube. This projection type has more even pixel density than equirectangular projection, though density still varies to a lesser degree over the cube faces.

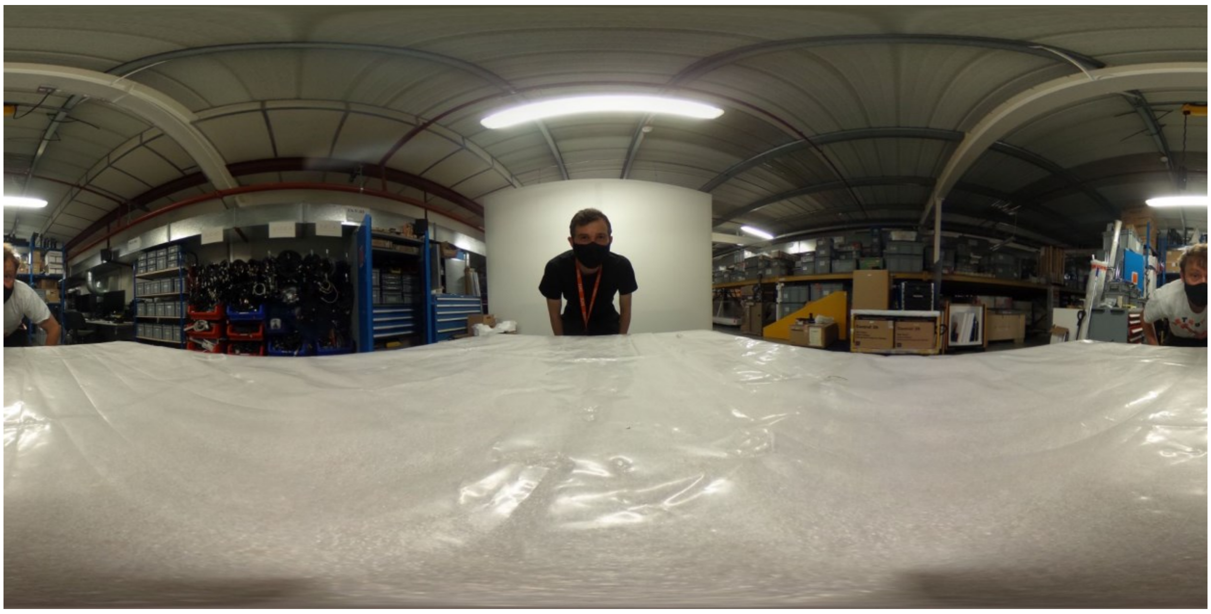
Figure 9. The image from Figure 7 in equirectangular format.

Cubemap projection types allow for more efficient use of compression codecs than equirectangular, since the inherent curvilinear distortion in equirectangular video introduces distortion into the representation of the path of a moving object. One of the ways in which compression attempts to minimise file sizes and data rates is by applying temporal compression, which is based upon the principle that from one video frame to the next most pixels remain the same, though the point of view may have changed slightly, or an object may have moved within the frame. Instead of redrawing the entirety of the pixels, motion compensation is used to represent the difference between each frame instead, using less data to do so. This process is at its most efficient when movement of perspective or object is in a straight line, and hence is hindered by curvilinear distortion[60].

A further development in projection format is the equi-angular cubemap (EAC) — in this projection type, a distortion mesh (see 3.1.2. Scenes and Assets) is introduced onto each face of the cube, resulting in each degree of viewing angle being assigned an equal number of pixels, creating more even pixel density[61]. Other projection types are being developed, such

[60] Rabia Shafi, Wan Shuai and Muhammad Usman Younus, '360-Degree Video Streaming: A Survey of the State of the Art', *Symmetry* 12(9), 2020, https://doi.org/10.3390/sym12091491.
[61] Chip Brown, *Bringing pixels front and center in VR video*, 2017, https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/.

as Facebook's pyramid projection format[62], a development of the cubemap but a pyramid shape is used in place of the cube.

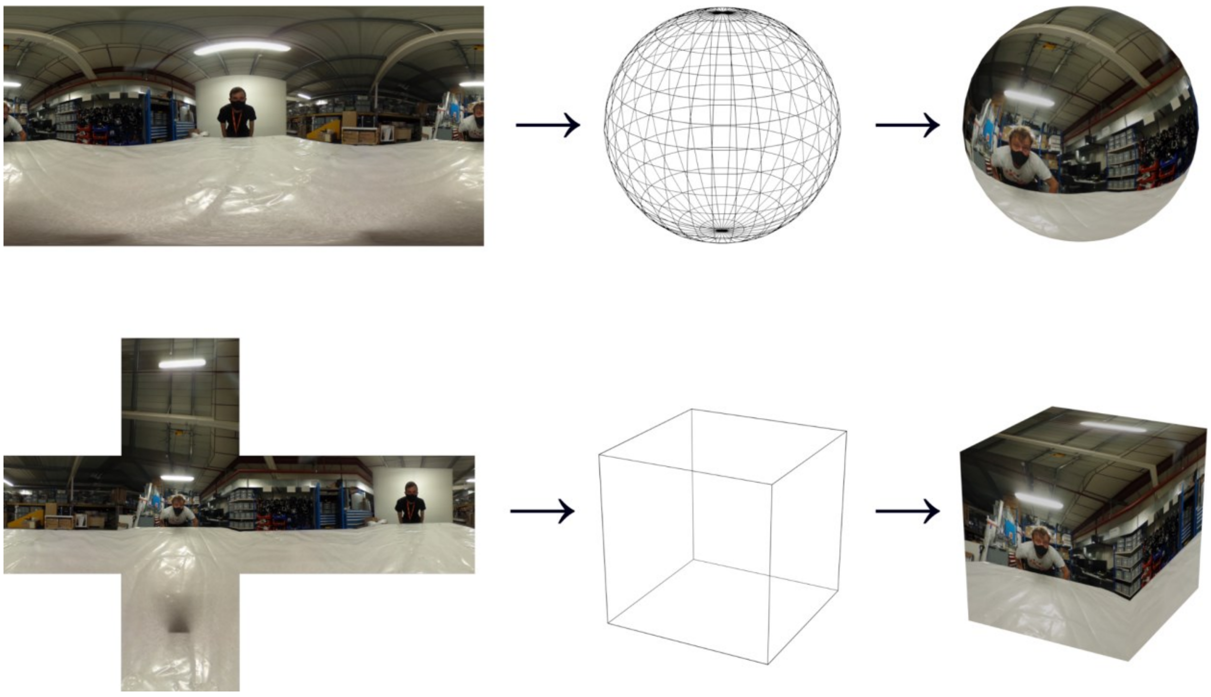

Figure 10. Representation of the rendering of equirectangular (top) and cubemap (bottom) projection types. In this process, a stored 360 image (left) is mapped onto a suitable 3D geometry primitive (centre), resulting in a projected image viewable with 3DoF from the centre of the primitive (right).

4.2. 360 Video Audio

Audio accompanying 360 video can be either be of the fixed mono/stereo type, where audio channels are played through a user's headphones irrespective of the position of the head, or it can use the position of the HMD to calculate a *binaural* mix from a multitrack spatial recording, in real time, relative to the position of the head. This gives the impression of the sound sources changing position relative to the video, potentially creating a more immersive soundscape.

4.2.1. Order of Ambisonics

In the case of 360 video captured from a camera with accompanying spatial audio, it is likely to be recorded by an ambisonic microphone — which in its simplest form (first order, or 4-channel) uses an array of four microphone capsules to record 360 degrees of audio. Second order (or 9-channel) ambisonics employs the same techniques but achieves improved spatial resolution with 9 microphones, while third order (or 16-channel) ambisonics uses 16 microphones for further resolution.

[62] Evgeny Kuzyakov and David Pio, *Next-generation video encoding techniques for 360 video and VR,* 2016, https://engineering.fb.com/2016/01/21/virtual-reality/next-generation-video-encoding-techniques-for-360-video-and-vr/.

4.2.2. Ambisonic Formats

This audio in its raw, recorded form is referred to as ambisonic *A format*. For this audio to be embedded into a video file, it undergoes processing to generate *B format*, where the raw audio from the capsules is converted to four audio files that represent the X, Y and Z spatial axes and W, overall amplitude. B format represents the captured space or *sound field* in an intermediate form for decoding- a fundamental principle is that it is *speaker agnostic* and can be decoded for a variety of speaker arrays. The conversion from A to B format is unique for each microphone, based on microphone make, model and individual capsule calibration. Software to undertake this process (often a VST plugin for a digital audio workstation) is therefore provided by the manufacturer. Some microphone arrays are available that record natively into B format, though these are less common.

4.2.3. Head-related Transfer Functions

In VR, B format ambisonics are decoded by a *head-related transfer function* (HRTF) to generate a real time binaural mix. An HRTF is an algorithmic process that models the human head and ears within a sound field recording, and approximates what a pair of human ears would hear within that space at a given orientation. It undertakes several key calculations to do this, such as: calculating the time difference taken for a sound to reach each ear; calculating the diffraction of sound waves caused by the head; and sound waves heard through being absorbed into the head and chest. This is performed in real time by the player, taking positional input from the HMD. Several data sets for the HRTF algorithm exist, based on averaged characteristics of human head and ear sizes and shapes. Whilst some data sets allow for adjustments of head size and ear spacing to accommodate differences in head shape and sizes, the calculations do not achieve perfect accuracy.

4.2.4. Format Conventions

Care must be taken in the preservation of ambisonic audio to note the specific conventions used to generate the B format files, as these impact how it is played back. B format files can be placed in different orders according to various conventions such as Furse-Malham, ACN and SID. Furthermore, to achieve the correct spatialisation the files are normalised in relation to each other according to various conventions such as maxN or SN3D. Two prominent exchange formats exist, FuMA (**Fu**rse-**Ma**lham) prescribes the channel ordering WXYZ and maxN normalisation, whilst AmbiX (**Ambi**sonic e**X**change) prescribes WYZX (ACN) ordering SN3D normalization. Video containers are agnostic to these conventions and we rely on file metadata to accurately reflect the scheme used to ensure that a playback device is able to correctly interpret the files.

4.3. 360 Video File Types & Metadata

4.3.1. Aspect Ratio and Resolution

In storage, 360 video file formats show similar characteristics to those of planar video. The aspect ratio of monoscopic video is commonly 2:1, which relates to 180 degrees of vertical vision and 360 degrees horizontal. The resolution of the stored file is often higher than that planar video — considering the viewable area of the video is 90 degrees and hence around 25% of the entire image, a 360 video would have to have four times the resolution of a planar video to achieve comparable levels of pixel density. Pixel resolution is often higher still in stereoscopic, as images for each eye are stored either in a side by side (SBS) or over/under (OU) format. This requires a further doubling of the image resolution, meaning that to view a stereoscopic image with comparable quality to a 1920x1080 planar video file, a horizontal resolution of approximately 16000 pixels would be required in an SBS file.

4.3.2. Frame Rates

The frame rate experienced by a VR user is a combination of the video frame rate and the refresh rate of the HMD. Smooth playback is potentially impacted by a low frame rate, latency in the VR tracking and rendering system, and to some extent the content of the video — high contrast vertical lines will create more visible motion artefacts such as ghosting than a scene without fine detail. Because of the wide virtual viewing area and the ability for the user to quickly rotate the head over large virtual areas of video, motion artefacts and latency can be experienced either consciously or subconsciously at higher frame rates than the equivalent frame rate of planar video. The study of VR sickness is complex and is thought to depend on many factors, but latency has been identified as one contributing factor[63]. Whilst Oculus suggests a minimum frame rate of 30 frames per second (FPS) for 360 video in its guidelines for content[64], it has been argued that higher frame rates will bring an improvement in quality should other factors such as storage or bandwidth allow.

4.3.3. File Sizes

High frame rates, large resolutions and stereoscopic images all combine to cause a significant increase in file sizes compared to planar video in similar formats. As a result, compression is often employed in playback to address issues of streaming bandwidth and storage — especially in the case of untethered devices such as the Oculus Go which is limited to 32GB or 64GB of onboard storage. At the time of writing, there are no specific codecs for 360 video — H.264 and H.265 are commonly employed. It is likely that a compromise between compression level, frame rate and resolution will have to be reached, optimised for the visible characteristics of the content and minimising the corresponding artefacts.

As online streaming becomes more commonplace, Scalable Video Coding is being utilised to maximise visible quality whilst minimising data bandwidth[65]. In this technique, each area of

[63] Eunhee Chang, Hyun Taek Kim and Byounghyun Yoo, ‘Virtual Reality Sickness: A Review of Causes and Measurements’. *International Journal of Human–Computer Interaction* 36 (17), 2020, https://doi.org/10.1080/10447318.2020.1778351.
[64] Marcos Carranza, *Introduction to 360 Video for Gear VR*, 2017, https://developer.oculus.com/blog/introduction-to-360-video-for-gear-vr/.
[65] Rabia Shafi, Wan Shuai and Muhammad Usman Younus, ‘360-Degree Video Streaming: A Survey of the State of the Art’, *Symmetry* 12(9), 2020, https://doi.org/10.3390/sym12091491.

video (in the example of cubemap projection, each face of the cube) has a subset of videos with greater compression and hence smaller file sizes/bandwidth. The area being viewed is streamed in maximum possible quality whilst the video in the periphery areas is streamed from a file of lesser quality.

4.3.4. Metadata

Due to the increased number of variable parameters in 360 video over planar video, there are increasing demands on the file metadata to accurately describe projection type, distortion map, and audio convention. At the time of writing, standards are being expanded to include capture information from the camera such as GPS position, direction of motion and other sensor data. Dominant capture standards at this time are the GoPro metadata format (now renamed to General Purpose Metadata Framework or GPMF) and the Camera Motion Metadata Spec. The GPS information can be used to situate the footage within databases such as Google street view, and the motion information can be used to enhance post processing procedures such as stitching and image stabilization. For playback, there is a spatial media metadata standard put forward by Google, and a corresponding spatial media metadata injection tool[66]. One particularly promising aspect for preservation is that it proposes to describe the relationship between video file and projection type mathematically within the metadata, adding a degree of self description. At the time of writing it is too early to comment on how widely it has been adopted. Apple now offers the ability to inject spatial metadata information with the Compressor tool. In absence of a metadata standard, some players had previously interpreted a string of characters from the file name- such as "_LR.mp4" for Left/Right stereoscopic panoramic video[67].

4.4. 6DOF & Volumetric Video

One of the primary differences between 360 video VR and RT3D VR is that in 360 video playback interactivity is limited to rotation left and right, rotation up and down, tilt left and right (3DoF). Moving the head from side to side or up and down positionally does not result in any change in view, given that the camera's position in a space is fixed as the video is pre-recorded from a fixed perspective.

There are several attempts underway in the VR industry to make limited 6DoF possible for 360 video, such as Adobe's project Sidewinder or HTC's 6DOF Lite. Some of these methods make use of a depth map which can be generated in some cases by stitching programs or capture hardware, whilst others, such as HTC's 6DOF Lite are able to generate a limited 6DoF experience from existing stereoscopic video by generating depth information in real-time. Tools to perform these functions are blurring the boundaries between 360 video players and real-time 3D engines, as more complex features of RT3D rendering such as vertex displacement are employed.

[66] Google, *Spatial Media*, 2018, https://github.com/google/spatial-media.
[67] Oculus, *Oculus Android VR Media Overview*, n.d., https://developer.oculus.com/documentation/native/android/mobile-media-overview/.

The similarities with RT3D rendering grow further still with volumetric capture, a technique that uses an array of cameras to capture a scene from every angle, hence allowing a subject or *volume* to be viewed from any angle. It differs from common 360 video formats in that it is often filmed from the outside and might be used to capture the performance of a human for example. It has similarities to the techniques used in photogrammetry, a capture technique which can derive a 3D representation of an object using photographs taken from multiple perspectives, but is carrying out this process of conversion from still images frames to point cloud 3D data in real time. At the time of writing, the amount of storage and computational power required put it outside the scope of this report.

5. Suitability of Existing Preservation Strategies

In this section we will use the insights gained in the preceding sections to explore approaches to the acquisition and long-term preservation of VR artworks. Ideally, these approaches would allow us to prevent loss of access to a VR artwork due to the failure and obsolescence of components. While these processes are something time-based media conservators contend with for any technology-based artwork, our research indicates that their effects may be particularly severe for VR artworks because of the following factors:

- **Frequency of failure**: VR hardware is handled frequently when a work is installed (i.e. HMDs, controllers) making it more likely to be damaged.
- **Rate of change:** Commercially available VR hardware changes very rapidly, with devices being removed from sale within 5 years of release.
- **Dependency on manufacturer-specific software**: In addition to the complex software and hardware interdependencies familiar from software-based artworks, VR applications typically require manufacturer-specific software to be present in order to access that manufacturer's VR hardware devices. Support for this software layer has to be built into the application when it is created if it is to support that hardware.

In the following sections we will explore factors we have found which are likely to complicate the process of acquiring and preserving VR artworks when compared to the more familiar forms of video and software-based art. We consider processes at point of acquisition and preservation strategies, the latter drawing on established concepts from conservation including stockpiling, hardware migration, emulation and code migration as starting points for discussion. We do not propose that these approaches are mutually exclusive, but rather that could be used in combination to maximise chances of providing long-term access to VR artworks.

5.1. Acquiring VR Artworks

5.1.1. Acquiring Real-Time 3D VR Artworks

For real-time 3D VR artworks, the point of acquisition is the time to gather together materials and documentation that will serve to support the work's life in the collection. Compiled real-time 3D VR applications are similar to those associated with other forms of software-based art, and many of the approaches we have developed for software-based art can be applied. Based on procedures in use at Tate, we would expect materials to be delivered by the artist (or generated at point of acquisition) to include: a computer system and other hardware suitable for running the artwork (or specification for sourcing an equivalent); executable software; source materials; and documentation.

As a starting point for acquiring a RT3D VR software application, we would typically test the software received using either an artist-supplied computer system or a new computer system created from artist-supplied specifications. Due to the complexity of VR systems and the many variables involved in correctly setting up and running an application, it will be beneficial for the artist (or someone else familiar with the artwork) to be present for this. This is an opportunity to begin to learn how the work functions, how it should be installed, and how it might be preserved. This is also the point at which a backup system would be created. Tate's existing best practice guidelines for software-based artworks is to ensure there are two functional reference systems available for a specific artwork. We therefore recommend that those aiming to preserve VR artworks source and configure two sets of reference hardware when an artwork is acquired. This is based on the principle of having a backup system for display in case of failure, and a reference system for monitoring the impact of changes. The process of creating another version of the system is a useful tool in understanding the artwork and validating dependencies.

As an additional backup of the software environment — and again, a standard part of software-based artwork preservation at museums such as Tate[68] and the Guggenheim[69] — disk images can be created from storage media of the computer system. When captured from a computer system, we can consider the disk image a representation of a complete *software environment*, incorporating an operating system, installed programs and user data. This image is useful not only because it encapsulates all the data contained, but because we can use it as a basis for emulation or virtualisation as a means of accessing these environments in the long-term (see 5.4. Emulation and Related Approaches). This is a well-established approach to preserving software environments in digital preservation and time-based media conservation, and resources and guidance can be found elsewhere[70,71,72]. We propose that this need be met by creating raw disk images of the storage media contained in the reference systems acquired or created. The resulting disk images ensure the complete software environment is captured and can then be used as a means of cloning the content to new storage media and as the basis of future emulation and virtualization work.

Supplementing this, it is beneficial to acquire builds of the software for as many different platforms as possible, to open up further options for keeping the software accessible. Building for Windows, MacOS and Linux operating systems increases the chances that a suitable software environment can be recreated in the future. Target VR runtimes and 3D APIs also need to be considered at this time, as support for a specific runtime or API must be included when the application is built. As noted in 2.2.1. VR Runtimes and 2.2.3. 3D APIs, targeting open standards such as OpenXR and Vulkan is likely to open up the largest number of options for hardware migration (see 5.5. Code Migration and Related Approaches for further discussion). However, changing the runtime or API can have significant effects on the characteristics of the software, so any new builds should be carefully tested, ideally in collaboration with the artist or studio.

[68] Tate, *Software-based Art Preservation – Project*, 2021, https://www.tate.org.uk/about-us/projects/software-based-art-preservation.
[69] Guggenheim, *The Conserving Computer-Based Art Initiative*, n.d., https://www.guggenheim.org/conservation/the-conserving-computer-based-art-initiative.
[70] MoMA, Disk Imaging, resources from *Peer Forum I: Disk Imaging*, 2017, https://www.mediaconservation.io/disk-imaging.
[71] Eddy Colloton, Jonathan Farbowitz, Flaminia Fortunato and Caroline Gil, 'Towards Best Practices In Disk Imaging: A Cross-Institutional Approach', *Electronic Media Review* 6, 2019, https://resources.culturalheritage.org/emg-review/volume-6-2019-2020/colloton/.
[72] Tom Ensom, *Disk Imaging Guide*, 2021, https://www.tate.org.uk/file/disk-imaging-guide-pdf.

Acquiring source materials for software-based art is important for preservation purposes[73], allowing the migration or modification of the software in order to maintain access in a changing technical environment. Source materials associated with VR applications pose challenges due to the complex processes and toolchains involved in production. This may be work carried out by a team, may involve the use of many distinct software tools and may involve multiple format conversions before assets are used in an engine. The capture and archiving of production environments as disk images may offer a pragmatic solution. This could be approached in two ways. The first is to disk image computers used in production directly. While providing the best way to ensure a complete capture, the indiscriminate nature of this approach results in the capture of all programs and data installed on the target machine, which may include unnecessary and/or personal data that would not be suitable for archiving. The second approach is to acquire the engine project (essentially a structured directory) and recreate the production environment on a physical or virtual machine by installing the appropriate engine binaries and testing the configuration. This is less invasive than the first approach and would reduce the amount of data that would have to be stored, but risks incomplete capture if not thoroughly tested (e.g. by recompiling the software and comparing it to the artist-supplied software). Whether it is useful to acquire asset production materials (i.e. projects and files associated with modelling, texture painting, material definition animation and other workflows) remains an open question. While having access to these would allow for more export options in the future and avoid problems with limited export options in engines, gathering and caring for them would have significant resource implications for the institution taking them on.

Documentation of real-time 3D VR artworks is an area we have not thoroughly explored in this report. However, the Acquisition Information Template and capture approaches discussed in 3.3. Real-Time 3D VR Documentation can be used as a starting point for RT3D VR-specific documentation work. Further research is required to understand additional forms of documentation which might find use in the conservation of software-based artworks. For the time being, we recommend that institutions work closely with the artists throughout the acquisition process to ensure that the materials retained are complete and well documented.

5.1.2. Acquiring 360 Video VR Artworks

360 video utilises the same container formats and compression codecs as planar video. File types will be familiar to those working in video preservation and well-established preservation strategies[74] will broadly apply. During the acquisition process it is common to collect a compressed playback copy of an artwork alongside a primary copy[75]. The primary copy should ideally be uncompressed or have minimal compression and this principle applies directly to video captured from RT3D engines. When collecting 360 video from camera capture, we are faced with the extra potential choice of collecting the raw camera output for restitching at higher quality in the future. At the time of writing, many consumer grade 360 video cameras

[73] Deena Engel and Glenn Wharton, *Reading between the Lines: Source Code Documentation as a Conservation Strategy for Software-Based Art*, *Studies in Conservation* 59 (6): 404–15, 2014, https://doi.org/10.1179/2047058413Y.0000000115.
[74] Matters in Media Art, *Sustaining Media Art*, n.d., http://mattersinmediaart.org/sustaining-your-collection.html.
[75] For digital video, the *primary* copy is typically a file in a high quality, stable format which is used for creating additional preservation exhibition copies.

will compress in real time during recording and store files in mp4 format with h264 codec. Post-stitching, the primary file may be edited and provided for acquisition in an intermediary format such as ProRes, and hence the primary material may be in a more lossy codec than the intermediary. An important point here is that because the stitching and editing process is potentially lossy and can create artefacts, the compressed camera files could still contain more accurate visual information than the intermediary, potentially making the case for them as a preservation format.

Extra care should be taken to ensure that the metadata accurately describes the projection format and spatial audio convention. As with other video types, checking for consistent playback across a variety of players and HMDs is a useful tool in checking for anomalies or potential areas for error. Because of the relatively fast paced development of headsets, a regular maintenance activity of playing the material on the newest generation of headset is advisable to provide early warning of obsolescence or need for migration. An HMD might be experienced slightly differently by different users, and in combination with their inherent distortion due to the use of wide-angle lenses, some degree of quality control is likely to be better undertaken on a calibrated monitor in combination with a headset.

At the time of writing, it is unclear how depth maps may be integrated into 360 video though software from Kandao is able to generate depth information at the stitching phase[76] and it is reasonable to expect that players may offer the ability to generate real time depth maps in the future[77]. The impact this has on the experience of the work should be considered and documented.

5.2. Hardware Stockpiling

An intuitive first response to the risks of hardware failure and obsolescence is to securely store digital objects[78] and stockpile suitable hardware. Stockpiling has well-established precedent in time-based media conservation, and alongside ongoing collaboration with specialised communities outside the museum, is one of the primary strategies for ensuring long-term access to CRT monitors for Tate's large collection of video art. However, applying the same logic to VR system hardware raises many difficult to answer questions. Given the short lifespans of interactive equipment used in public galleries and Tate's mandate to care for artworks in perpetuity, how many pieces of hardware is enough? Given the relatively small number of VR artworks likely to be acquired by any one institution (this remains an emerging medium), how can we justify the considerable financial outlay of acquiring such a quantity of hardware? Given that VR-specific hardware like HMDs and tracking systems are contingent

[76] Mic Ty, *Kandao Obsidian 3D 360 cameras can now export a depth map*, 2016, https://360rumors.com/kandao-obsidian-3d-360-cameras-can-now-export-depth-map/.

[77] Benjamin Attal, Selena Ling, Aaron Gokaslan, Christian Richardt, and James Tompkin, 'MatryODShka: Real-time 6DoF Video View Synthesis using Multi-Sphere Images', at *European Conference on Computer Vision*, 2020, http://visual.cs.brown.edu/projects/matryodshka-webpage/.

[78] As we have not identified any issues relating to the effective archival storage of digital materials that are unique to VR, this topic will not be addressed further in this report. It is worth noting however that virtual reality artwork binaries and project files, as well as disk images, can present very large volumes of data.

on specific computer hardware, what would be required to stockpile suitable computers as well?

The model of stockpiling and specialist repair that museums have employed to maintain access to CRT television technology has worked thus far because these technologies were widely used during the time of their production and sale. The same could not be said for VR hardware at this time, which despite recent interest remains relatively niche. It is currently not clear how easy it would be to source spare parts or service manuals in the long term, or what specialist support might be available. Based on the challenges identified here, the stockpiling and maintenance of a pool of VR hardware does not seem to be a practical option for maintaining long term access to VR artworks. However, ensuring access to at least two sets of hardware for any one artwork, as described in the preceding section, does seem a sensible short-term measure. Whilst stockpiling larger quantities of hardware might seem to solve a problem, we expect that all hardware will fail eventually. In combination with budget and space restrictions, other preservation strategies are necessitated.

5.3. Hardware Migration

If replacement of failed hardware is not possible with like-for-like hardware, we can instead consider replacement with newer hardware — a process known as hardware migration. For example, we might hope to run an old VR application with a contemporary HMD and tracking system. Based on our research, we feel that there is a high likelihood of these kinds of changes resulting in loss of the characteristics that define an artwork, or the loss of access to a particular VR artwork altogether. We can look at the impact of changing HMD as an example. At best, this might result in the loss of the distinctive character of the original HMD (e.g. the lower resolution panels and restricted field-of-view found in earlier models), which may or may not be significant for that particular artwork. At worst, it might result in the loss of functionality altogether, due to the replacement HMD lacking software support for communication with the VR application — a newer HMD is likely to require the use of new drivers, APIs and/or runtimes, support for which may not have been built into the application.

As noted earlier in this report, these risks can to some extent be lowered by creating multiple builds of the application which support different platforms. This should include building support for a variety of VR runtimes into the application and creating multiple versions of the application for different operating systems and 3D APIs. Carrying out this kind of preparatory work allows us much more flexibility in recreating a suitable execution environment in the future by maximising the ways in which a functional system could be pieced together. If an open standard for VR runtimes such as OpenXR is widely adopted, building support for such a standard into VR applications may greatly ease the process of hardware migration. Theoretically, it might be possible to run older VR applications built with OpenXR support on any contemporary VR hardware supporting the standard, providing backwards compatibility with earlier versions of the standard is maintained. Whether backwards compatibility will be maintained is uncertain, as we do not know if the maintainers of OpenXR, the Khronos Group, see this as a priority.

3D engines and VR hardware are both unlikely to be created by artists themselves, therefore control over the availability of OpenXR to artists lies with the companies that create these technologies. Although signs of industry investment and adoption are promising, we cannot be certain as to the extent to which this standard will be adopted or supported by software and hardware. There are also good reasons to treat industry investment in open standards with some skepticism, as priorities are known to change. As noted in [2.2.3. 3D APIs](), Apple have recently decided to deprecate their OpenGL implementation on MacOS in favour of their own proprietary Metal API. OpenXR is a consortium largely made up of industry partners with paid for representation, though tiers of membership are available, and representation of cultural heritage institutions has been discussed in outline.

While use of open standards such as OpenXR has clear advantages for maintaining access to VR artworks in the short term, their longer-term benefits may be limited by other factors affecting the feasibility of hardware migration. For example, significant changes in prevalent CPU architectures, operating systems and 3D APIs will also affect the viability of running older VR applications on contemporary computers. We must therefore consider the possibility of more significant interventions in the future, such as emulation and code intervention, which we consider in turn in the following sections.

5.4. Emulation and Related Approaches

Emulation and related approaches are preservation strategies which make use of software to allow one computer system to behave as if it were a different computer system. This allows us to run unmodified software (e.g. a VR application or 360 video player) on computer hardware which it was not designed to run on. Within this category of approach we include not only emulation and virtualisation, but also related tools like compatibility layers and wrappers. These can be contrasted with code migration and related approaches, which are preservation strategies that make changes to the software itself through the modification or rewriting of code. These are discussed further in the subsequent section.

Emulation relies on the use of tools which recreate a particular set of hardware (the most important of which is the CPU) in software. For long-term preservation purposes, this allows a software environment (e.g. a disk image) to be separated from obsolete physical hardware and accessed through emulated hardware on a contemporary computer system. Related to emulation is virtualization, which involves many of the same principles but additionally allows the guest (i.e. the emulated machine) access to some physical hardware on the host machine, which improves performance. Emulation and virtualization have demonstrated uses in the conservation[79,80] of software-based art and use of emulators is common in the care of software-based artworks at Tate. We might therefore expect these approaches to be applicable to VR artworks too. They can be seen as a first logical step for treatment, as in comparison to the more involved work of code migration, they can be carried out without source code access or developer time.

[79] Klaus Rechert, Patricia Falcão and Tom Ensom, *Introduction to an Emulation-Based Preservation Strategy for Software-Based Artworks*, 2016, http://www.tate.org.uk/research/publications/emulation-based-preservation-strategy-for-software-based-artworks.

[80] Patricia Falcão, Ashe Alistair, and Brian Jones, *Virtualisation as a Tool for the Conservation of Software-Based Artworks*, at iPRES 2014, https://www.academia.edu/12462584/Virtualisation_as_a_Tool_for_the_Conservation_of_Software-Based_Artworks.

At time of writing, full system emulation does not appear to be a feasible strategy for preserving virtual reality artworks. This stems from limitations in emulated hardware performance and functionality. Emulators rely on virtual CPU and GPU hardware, which recreates or approximates physical graphics hardware in software which can execute the necessary code. However, due to the overhead of executing this code in software, virtual hardware tends to be much less powerful than its physical counterpart, resulting in performance problems when running applications with high performance requirements — such as VR applications. At time of writing, the emulated GPUs we examined (including those packaged with QEMU, VirtualBox and VMware) do not offer sufficient 3D rendering capabilities to run virtual reality artworks at required speeds (or at all, in some cases). Furthermore, the current generation of VR runtimes require access to a physical GPU in order to access an HMD in direct mode. A workaround for this is to allow the emulated environment a level of access to a physical graphics card via techniques called paravirtualization and passthrough. However, these techniques still rely on physical hardware and associated drivers, thus limiting the value of the resulting emulated machine for preservation purposes. Time will tell whether or not emulation will become suitable for running the VR applications of our current time. Given trends in computing, it seems likely that this may eventually be so, but remains contingent on sufficient interest from enthusiast communities or industry to create suitable emulators.

The use of compatibility layers and similar tools offers an alternative to full system emulation with short-term relevance to the preservation of VR artworks. Compatibility layers and wrappers translate system or API calls made by a program to another system or API. By virtue of shared processor architectures (most contemporary desktop computing software is compiled for x86-64 processors) a program built for one operating system can be executed on another operating with minimal performance overhead, as any native code can be run on the CPU as-is. However, system and API calls are usually specific to the operating system on which they were built to run. For example, Wine[81] allows the execution of software designed for modern Windows operating systems on modern Linux operating systems (including applications which require the use of the Direct3D API) by translating system calls from one platform to another. While the dependency on a specific processor architecture remains, tools such as Wine do have the potential to at least slow obsolescence by allowing software to be run on a wider variety of software environments.

Compatibility layers have also been developed to allow the use of applications designed for one VR hardware set with another. For example, Revive[82] allows Oculus-targeting applications to be used with HTC Vive hardware, while OpenOVR[83] allows the inverse for SteamVR targeting applications. Whether similar tools will emerge to support legacy VR applications on future platforms is unknown, but given the level of interest in VR from the gaming community it’s easy to imagine retrogaming enthusiasts taking on this kind of challenge — particularly with the financial support of cultural heritage institutions. The use of proprietary technologies will slow progress in this area, as they require time-consuming reverse engineering work to make sense of. This gives further support to the notion that we should favour open standards where possible. Again, use of OpenXR would provide a significant advantage here as an open

[81] Wine, *WineHQ* web page, n.d., https://www.winehq.org/.
[82] LibreVR, *Revive*, GitHub repository, 2020, https://github.com/LibreVR/Revive.
[83] Campbell Suter, *OpenComposite* GitLab repository, 2020, https://gitlab.com/znixian/OpenOVR.

standard: a specification of the standard is publicly available, which theoretically allows compatibility software to be written in compliance with the standard that can then communicate with any other software that is compliant.

While it is impossible to predict whether or not the emulation approaches discussed in this section will be available to us in the future to apply to VR applications, there are important steps we can take now to prepare for the eventuality that they are. One of these is to, as described in 5.1.1. Acquiring Real-Time 3D VR Artworks, create raw disk images of the storage media of artist-verified and tested computer systems. Disk images can then be used as the basis of emulation efforts, without the need to reconstruct a suitable software environment from scratch in the future — something which is likely to be very challenging due to the large number of components involved and their delivery through internet-connected installers. To supplement imaging and ensure we can define suitable virtual hardware in the future, the hardware of the physical computer system should be carefully documented. Finally we should continue to advocate for the use of open standards in the development of VR systems where possible, including supporting artists in the creation of new software builds if necessary. Where proprietary technologies are unavoidable, we can try and advocate for greater openness within the industry and ensure that legal provisions better protect those having to bypass copyright protections in order to support legacy access to software. Practitioners in the United States have had success developing best practices for applying the fair use[84] right to digital preservation and a flexible exemption that favors public interest uses[85]. They have also obtained regulatory relief, such as the recent DMCA copyright exemptions adopted by the US Copyright Office[86].

5.5. Code Migration and Related Approaches

Code migration and related approaches are preservation strategies that make changes to software (e.g. a VR application or 360 video player) through the modification or rewriting of code, so that it remains accessible in changing technical environments. There are many different ways we could apply this approach to VR artworks, varying in the extent to which original source code is altered and the frequency with which changes would be made. In this section, we consider the hypothetical application of code migration to a VR application built in a game engine. It is important to note this approach is contingent on access to production materials, including at least the engine project and assets, and engine binaries of the appropriate version with which to open it. The challenges involved in acquiring these are discussed further in 5.1.1. Acquiring Real-Time 3D VR Artworks, where we propose that disk imaging offers a particularly useful tool.

[84] Brandon Butler, Law & Policy Advisor for the Software Preservation Network, noted in a comment on an early draft of this report that fair use exists in several legal regimes outside the US, but countries without fair use (including many countries with ‘fair dealing’ provisions, such as the UK) should explore what other provisions in their law can support software preservation or consider advocacy for their adoption.
[85] Software Preservation Network, *Code of Best Practices for Fair Use in Software Preservation*, n.d., https://www.softwarepreservationnetwork.org/project/code-of-best-practices-for-fair-use/.
[86] Kendra Albert and Kee Young Lee, *A Preservationists Guide to the DMCA Exemption for Software Preservation*, 2018, https://www.softwarepreservationnetwork.org/a-preservationists-guide-to-the-dmca-exemption-for-software-preservation/.

A use of code migration with short-term value would be to add interoperability to an existing VR application. This could involve adding support for a new VR runtime by installing an additional plugin (or adding equivalent code) or creating a new application build targeting a different operating system. In the short term this may be an effective way of opening up more hardware migration options, but is limited by the lack of long-term support for specific engine versions. As discussed in 3.1.1. Engines and Project Files, new engine versions are released multiple times per year, superseding older versions which eventually become publicly inaccessible. This points to a need to migrate between engine versions, an approach we call *incremental migration*. This frames the process of code migration as something occurring in small steps which, given the frequency of engine version releases, would be undertaken as a regular maintenance activity. This diverges from typical approaches to time-based media conservation where intervention happens infrequently, often at moments associated with significant events in the life of the artwork such as acquisition or display. As we are not aware of any examples of this kind approach being applied in the field, drawing any conclusions as to its viability is difficult. Further research is required to determine whether a reframing of migration as maintenance would be practical or desirable in the cultural heritage sector, given the resources it would entail.

Assuming incremental migration is not a practical approach, what would be the impact of less frequent interventions on the viability of code migration? In these cases, it is likely that more extensive modification of the underlying code would be required. While this has been demonstrated as a viable approach in the conservation of software-based art[87,8889], when considered in the context of VR applications (and other real-time 3D software produced using engines) additional complications arise. The first issue is access and rights regarding the use of engine source code, which would be required for this kind of approach. As discussed in 3.1.1. Engines and Project Files, the full Unity source code is not accessible or modifiable without negotiation of a licence. Unreal Engine 4 has publicly accessible source code, with relatively permissive licence conditions, but restricts redistribution of source code. For Unreal Engine 4 then, at least, this level of access and rights would likely be sufficient to carry out any code migration work in order to maintain access. For Unity this is less clear, as we do not know how sympathetic the company would be to requests for access and rights from the cultural heritage sector.

Assuming the necessary source code access and modification rights can be gained, we are presented with a secondary issue in the feasibility of carrying out modifications or rewrites. Modern game engines such as Unity and Unreal Engine are complex and sizeable pieces of software, developed by large teams over many years. What level of effort would be required to modify or rewrite the code base behind such an engine, in order to get it running in a different technical environment? Would an organisation working in the cultural heritage sector have

[87] Deena Engel and Glenn Wharton, Reading between the Lines: Source Code Documentation as a Conservation Strategy for Software-Based Art, Studies in Conservation 59 (6): 404–15, 2014, https://doi.org/10.1179/2047058413Y.0000000115.

[88] Deena Engel and Joanna Phillips, 'Introducing 'Code Resituation': Applying the Concept of Minimal Intervention to the Conservation Treatment of Software-based Art', *Electronic Media Review* 5, 2018, https://resources.culturalheritage.org/emg-review/volume-5-2017-2018/engel-2/.

[89] Mark Hellar, The Artist and the Technologist, 2019, https://journal.voca.network/the-artist-and-the-technologist/

access to the resources that would be required to carry out such a task? Without case studies from the field of time-based media conservation, we can instead look for insights in other fields. In the video game industry the process of *porting* — a term essentially analogous to code migration — is a common activity in the development of video games. This is a useful reference point as video games use the same real-time 3D technologies that VR artworks employ. Within the industry, porting is often handled by specialist studios. There are relatively few examples of the details of this work being discussed in public forums, but insights from porting studio indicate that this is a complex and time-consuming process[90][91]. Both articles touch on ways in which changes to a game can be wrought by the process of porting, with Frank Cifaldi arguing in the latter that, "with all the moving parts involved in engineering for specific platforms it's I would say impossible to exactly replicate." This is a significant concern in an art conservation context, where issues of authenticity and the potential loss of work-defining characteristics are important considerations.

If engine source code is not available, we might consider migration as theoretically viable *between* game engines, a process which would involve asset or scene migration and manual reconstruction of other dynamic elements such as scripting and custom programming or plugins. There are many potential pitfalls to this process, however. The first is that it is reliant on being able to export assets in a suitable format and without loss of any of their properties. Further research is required to understand how feasible this is, but there may be benefits to preserving assets independently of the engines they were created in — a significant challenge given the number of formats and tools involved in these workflows. It is also likely to be limited to those assets (or scene components more generally) which are exportable at all. Some, such as particle systems, lights and components using custom code, will need to be manually reconstructed in the new engine as they cannot be exported to an interchange format. Whether this might result in loss of important characteristics is uncertain without further research, and dependent on the design of future game engines. Even in the best-case scenario that assets can be migrated, features built into the new engine may not match those of the original engine. For example, the implementation of material shaders may differ, resulting in changes to characteristics of the rendered image. Again, this is not something we can hope to fully understand at this time; there are no case studies that we are aware of and we cannot predict the future development of real-time 3D software.

As this discussion has highlighted, there is a great deal we don't yet understand about code migration's potential uses in preserving VR artworks. However, there are certain essential steps that can be taken now to prepare for the possibility of code migration. The most important of these is to ensure that production materials are acquired where possible, and that efforts are made to gather all the third-party dependencies required to open and compile these source projects. As discussed in 5.1.1. Acquiring Real-Time 3D VR Artworks, disk imaging appears to offer a suitable solution to capturing production environments, with some caveats. Where only the source project is acquired, future access to this will be contingent on access to engine binaries and other third-party software. This points to a need for this kind of software to be

[90] Alex Wawro, *What exactly goes into porting a video game? BlitWorks explains*, 2014, https://www.gamasutra.com/view/news/222363/What_exactly_goes_into_porting_a_video_game_BlitWorks_explains.php.
[91] Tom Bennet, *Bridging the generation gap: Porting games to new platforms*, 2015, https://www.polygon.com/features/2015/11/30/9790028/video-game-ports-remasters-the-last-of-us.

preserved and access to it maintained, be it by its manufacturers or an independent body. While this work may be occuring within the industry, this is not clear from available information. What a body outside of the industry might look like, or whether it might already exist is also unclear and requires further collaboration within the cultural heritage sector to determine. For now, the feasibility of code migration as a preservation strategy for VR artworks remains difficult to meaningfully assess. It offers a rich area for future research however, and case studies testing the viability of the process would be particularly valuable. One potentially interesting approach would be to explore artworks produced for older VR technologies from the 1990s as case studies for migration to current generation hardware.

6. Summary and Recommendations

In Section 2 we identified the key components of a VR system and their complex interdependencies. We ascertained how variance in the characteristics of a VR artwork might occur through the use of different hardware and software components. Due to the rapidly evolving commercial landscape of VR manufacturing, hardware components are at high risk of obsolescence, threatening the sustainability of VR systems. This is exacerbated by a lack of standardisation in VR runtime software, which has a critical role in allowing access to VR hardware. We discussed how open standards in this area could provide potential for migration in the future, particularly the OpenXR specification.

In Section 3 we examined the production environments in which RT3D artworks are created. We highlighted their reliance on game engine technologies and the difficulty of accessing legacy versions of these. We identified long-term access to source materials as a potential challenge for preservation, particularly due to restrictive access conditions for engine source code. Later in Section 5, we discussed an approach to capturing production environments using disk imaging, although this requires further practical case studies to fully understand the implications of creating and managing these. We found that compiled VR applications can be acquired using a similar approach to other real-time 3D software, although careful verification of rendering and performance characteristics is required during playback. Finally, we briefly explored approaches to the documentation of real-time 3D VR, focusing on our own acquisition information gathering template and video capture as key topics for future research.

In Section 4 we explore how 360 video can be generated in monoscopic or stereoscopic formats from a variety of camera types and layouts. The process of making this raw footage viewable through stitching is examined, along with the implications for preservation. Two popular projection formats are looked at in detail and it is acknowledged that there is likely to be development in projection format types, driven in part by the popularity of streaming 360 video. We then examine the audio components of 360 video and highlight different file conventions that could cause discrepancies in playback. Due to the many permutations of projection type and audio conventions, we highlight the importance of metadata accuracy for preservation. Briefly, we explore the generation of depth maps that enable 360 video to have a sense of movement that mimics the 6DoF afforded by RT3D. We note that the development of depth maps in 360 video potentially has computational requirements similar to RT3D processing.

In Section 5 we discussed the feasibility of applying existing acquisition workflows and preservation strategies to VR artworks as a way of managing change. We proposed that real-time 3D VR can benefit from the use of approaches to acquisition used for other forms of software-based art, such as sourcing duplicate hardware and creating disk images. Benefits and challenges to acquiring source materials and generating multiple builds were also discussed. 360 video was found to have many similarities to other forms of digital video, although extra care is required to ensure that the most appropriate masters are sourced and that metadata is appropriately captured.

We also discussed the potential feasibility of several treatment approaches for real-time 3D VR artworks. Stockpiling hardware is unlikely to offer a feasible strategy, particularly given uncertainties around access to networks required for ongoing maintenance and repair. If like-for-like replacement of hardware becomes impossible, we will be relying on approaches which are currently uncertain and untested. The usefulness of hardware migration is likely to be limited by adoption of open standards for XR runtimes in the short-term, and then eventually by the obsolescence of other software environment components (e.g. operating system and drivers) on which the target VR application depends. We noted that it is also likely to change characteristics of the artwork. The usefulness of emulation is limited by lack of support for XR and 3D rendering in emulation tools in current tools. Code migration presents the most uncertainties of all the strategies, given the lack of precedent in art conservation for migrating real-time 3D software to new engines. We proposed incremental migration to new engine versions as one treatment approach with short-term use.

We conclude this report by offering a set of recommendations for artists and institutions who are dealing with the immediate problem of caring from VR artworks. These strategies are based on a set of case studies, development of established acquisition workflows and steps required to prepare for future preservation procedures. They represent a snapshot of our understanding of this topic at this time and we hope will be refined and built upon by others. With that in mind, we also provide a set of recommendations for future research topics in this area.

6.1. Recommendations for Artists

For artists we recommend the following steps are taken as a short-term stabilisation strategy for the VR works they are caring for:

- Ensure you have a complete offline VR system (including hardware and software) configured and correctly running on the target hardware.
- Capture and archive a disk image(s) of the contents of the primary storage volumes of this computer system.
- Ensure you have a duplicate backup of this system (including hardware and software).

For VR artworks with a real-time 3D component, we recommend the following additional steps are taken:

- Create software builds for as many suitable platforms as possible, test them and archive these with configuration instructions.
- Maximise application support for a variety of VR hardware by using all suitable VR plugins and SDKs in the software builds created.
- Carefully manage engine projects and assets so that they are contained within a single location, and retain an installer or package of the relevant engine version binaries.
- Archive snapshot(s) of a production environment (typically consisting of at least configured game engine binaries and project files), ideally as a disk image.

For VR artworks with a 360 video component, we recommend the following additional steps are taken:

- Consider archiving raw camera output to allow footage to be re-stitched in higher resolution as technology progresses.
- Consider archiving the complete production environment for re-export, ideally as a disk image.
- In the case of works exported from RT3D engines, consider archiving the production environment, including project files and engine binaries, ideally as a disk image.
- Ensure that the file metadata accurately describes all parameters such as projection format, distortion map and spatial audio conventions. If possible, test on a variety of players for consistency.

6.2. Recommendations for Collecting Institutions

For collecting institutions, we recommend the following steps are taken as a short term stabilisation strategy for the VR works they are caring for:

- Ensure you have a complete offline VR system (including hardware and software) configured and correctly running on the target hardware.
- Capture and archive a disk image(s) of the contents of the primary storage volumes of this computer system.
- Ensure you have a duplicate backup of this system (including hardware and software).

For VR artworks with a real-time 3D software component, we recommend the following additional steps are taken:

- Acquire or create software builds for as many suitable platforms as possible, test them and archive these with configuration instructions.
- Maximise application support for a variety of VR hardware by using all suitable VR plugins and SDKs in the software builds created.
- Acquire or recreate a production environment (typically consisting of at least configured game engine binaries and project files), verify it and archive as a disk image or component parts.

For VR artworks with a 360 video component, we recommend the following additional steps are taken:

- Attempt to play the video file on a variety of different software players and untethered headsets as reasonably possible, identifying any variances in audio or video.
- Verify that the metadata correctly describes the projection format, distortion map, and spatial audio convention.
- In the case of works captured from camera, consider archiving raw camera files to enable re-stitching at higher resolutions as technology progresses.
- In the case of works exported from RT3D engines, consider archiving the production environment, including project files and engine binaries, ideally as a disk image.
- Consider a regular maintenance task of playing the artwork on the latest generation of headset to identify potential change.

6.3. Recommendations for Further Work

For VR artworks in general we have identified the following priorities for further work in this area:

- Monitor development and support adoption of open standards for VR.
- Development of a framework for gathering and interpreting artwork documentation, including external video capture, screen capture, artists description, narration etc.

For real-time 3D VR artworks we have identified the following priorities for further work in this area:

- Monitor development and support adoption of open standards for real-time 3D software.
- Support further research into the practicality of maintenance as a preservation strategy, including how frequently maintenance would be required.
- Support further research in understanding variability in real-time 3D rendering, and the effective documentation and management of performance and rendering characteristics.
- Monitor and support the development of emulation and virtualization and their support real-time 3D rendering.
- Support further research into 3D formats for stabilising 3D model assets.
- Support further research into better understanding how to effectively capture fixed-view and 360 video from real-time 3D VR artworks.

For 360 video we have identified the following priorities for further work in this area:

- Monitor the evolution of metadata standards and their adoption.
- Monitor the evolution of projection formats and the implications for player compatibility and sustainability.
- Monitor the evolution of tools that generate depth or 6DoF information from stereoscopic 360 video and consider the implications for future playback.
- Monitor the evolution of volumetric video capture, and the increased dependency on real time rendering for its playback.
- Monitor the evolution of 360 video file types and assess the suitability of media migration.