Revisiting the Scene-Graph-as-Bus Concept: Inter-networking Heterogeneous Applications Using glTF Fragments

Jaspreet Singh Dhanjan* Anthony Steed†

Department of Computer Science
University College London

Figure 1: Three separate client applications loading the 3D model with textures from the Hyperverse server. Left: Unity; Middle: Python; Right: Three.js. Changes to the scene can then be propagated between clients in real-time.

Abstract

While there are now many examples of successful collaborative mixed reality applications, each application uses its own custom networking framework and applications rarely inter-operate. To enable much larger-scale distributed systems, we will need inter-networking protocols that allow heterogeneous applications to exchange data. We demonstrate a proof of concept implementation that revisits the concept of using a scene-graph as a bus. That is, sharing low-level geometry and rendering information, rather than high-level semantic events. Our networking protocol uses glTF fragments and edits to express scene changes. We use the proof of concept to explore the potential to inter-network very different applications that are based on different underlying graphics engine technology.

Index Terms: Human-centered computing—Human computer interaction (HCI)— Interaction paradigms—Mixed / augmented reality; Networks—Network protocols—Application layer protocols—;

1 Introduction

The inter-networking of mixed-reality (MR) applications is a complex problem, both because typical scenes contain a wide-variety of different media types and because they have a requirement to deliver updates in real-time at low latency. While there are many standards for many single media, such as audio and video, there are no commonly-used standards for sharing virtual environments, including geometry, dynamic objects, shading information, etc. (though see discussion in Section 2). There are several reasons for this, but one of them is that MR applications are often built around graphics engines, such as Unity or Unreal, that embed a lot of engine-specific detail in their data-structures. The problem is further complicated by the typically fast update rates that are needed for simulation of the movement of users and the types of simulation within the environment such as physics simulation. These might need dozens of updates per second, though the updates may be small.

The most common way to enable networking for a specific application is to write a high-level, application-specific set of messages or remote-procedure calls that can be used by code specific to that application to manage a set of assets created within the graphics engine's run-time environment [26]. This approach can be highly efficient, but it does mean that applications are not inter-operable with other applications. It can even be the case that not even different versions of the same application can inter-operate because of the tight binding of network messaging to the specific types associated with data in the application.

There are alternatives to this, with standards such as X3D where there is a common scene-graph data structure, and which is more amenable to networking, since all the run-times that implement the standard should at least preserve the core data and fields in the scene-graph. Unfortunately, despite potential advantages, these models have only had modest success and they are not well supported in the leading game engine systems.

An alternative strategy for networking graphics application is the concept of scene-graph-as-bus (SGAB) from Zeleznik et al. [28]. In this approach, messages are exchanged that describe a form of neutral scene-graph that encodes the visual elements of the scene. These messages can be translated or bridged into a native scene-graph, so the main benefit of the SGAB idea is that application developers need to implement this translation step once. As the neutral scene-graph is quite generic, it can also represent a wide variety of shared state. The main disadvantage is that only the representation is shared, not any high-level state. We return to this discussion later.

In this short paper we revisit this idea using recent tools. We also suggest and show initial implementations of extensions that better fit MR situations where updates are streamed continuously. We demonstrate a proof of concept implementation of the SGAB that uses the GL Transmission Format (glTF) as a neutral scene-graph, called the Hyperverse Transfer Protocol (HVTP). glTF is designed for transmission, but it is most commonly used as a file format for asset interchange and loading into run-time environments. In our proof of concept, applications with different code bases (Unity, Python, JavaScript), can share and modify glTF from within applications. We then do two extensions to this proof of concept. Rather than sharing whole glTF data blobs, we include a glTF diff function. We then show an example of extending the mechanism to very rapidly changing data, such as rigid transformations.

This proof of concept illustrates that the SGAB concept can have a role in inter-networking of diverse applications. We discuss some of

*e-mail: jaspreet.dhanjan.19@alumni.ucl.ac.uk
†e-mail: A.Steed@ucl.ac.uk

the implementations challenges that are faced in making it scale and some of the conceptual limitations that will necessitate extensions to glTF, or support within a replacement for glTF, to encapsulate important state that is otherwise not visible to the scene-graph layer.

2 Background

2.1 Networking Graphics Applications

There are a diverse set of strategies, protocols and toolkits to enable the networking of real-time graphics applications [24, 26]. Mixed-reality applications have complex requirements in that the types of data that they need to exchange are complex (e.g. geometry, textures) and fast-changing (e.g. motion of dynamic objects). Mixed-reality applications can exploit streaming formats for certain media such as audio and voice, but fast, dynamic interactions typically require custom networking solutions (e.g. see [9]). Networking these applications can also involve configuring servers and relay services. There are toolkits that help with a lot of this functionality (e.g. Photon Realtime, see [8]), but responsibility for the sharing of application-specific data still falls to the application developer. For example, they may need to develop and update serialisation methods for their application-specific data, and then deal with synchronisation discrepancies, ownership, etc.

One popular approach to networking of graphics applications was to base the software around a scene-graph that could be shared [10, 15, 16, 21, 27]. The observation was that given the graphics engine of the application was written in a scene-graph system, that scene-graph, with some simple additions, could represent the whole state of the application, and then networking could be achieved by encoding changes to the scene-graph in different messages. This approach can supported multi-client applications, but saw particular use in cluster-based rendering [12, 19].

In many of these tools, and their successors, there was strong coupling between the networking and graphics scene-graph (e.g. the Inventor library in DIV [15]). Others were more loosely coupled in that the shared scene-graph was somewhat generic, and had multiple different rendering implementation (e.g DIVE [11] which had multiple renderers of quite different types including a direct mode renderer in OpenGL and scene-graph mirroring into Performer [25]).

The theme of basing networking around the scene-graph was taken to its logical extreme with Zeleznik et al.'s scene-graph as bus (SGAB) proposal [28]. In that proposal a neutral scene is constructed, and applications shared changes to that neutral scene-graph. Any application would need to transform the neutral scene-graph into a rendering scene-graph or other structures necessary. However, multiple heterogeneous applications could communicate only by observing change to this neutral scene conveyed in a standard set of messages. The main advantage of this approach is that the neutral scene-graph can be relatively compact, and the messages types need only to be constructed once. The main disadvantage is that all communication or signalling must be done through the scene-graph, so for any collaborative application, certain conventions would need to be followed. While there might be standardised libraries for run-time representation of the neutral scene-graph and updates to this, there is a development hurdle in that this must be mirrored into the application's own data structures.

We also note the long history of 3D standards for the web such as X3D. We consider these to be out of scope for the purpose of this proof of concept because the support for serialising and deserialising to such standards is very highly variable across implementations, and they are not inherently designed themselves for network transmission. We do note that a binary version of VRML97 was embedded in MPEG4 for streaming of scenes [23], and that extensions to X3D to support networking between engines that have already loaded scenes were proposed [4]. We refer the reader to a recent survey [7].

2.2 Graphics Exchange Formats

Our goal is to revisit the SGAB concept as presented by Zeleznik et al. [28]. The most complex part of any implementation would be representation of standard assets that might be found in graphical scenes. There have been significant developments in 3D file formats in recent years. There are a plethora to choose from and each may serve a different purpose. Pixar's Universal Scene Description (USD) [18] is a highly scalable format intended for rapid previewing of 3D geometry and shading. Autodesk's FBX [1] is another favoured by modellers. However, perhaps the most promising format to consider is the GL Transmission Format (glTF) 2.0 by the Khronos Group [13].

Since its initial introduction in October 2015, glTF has received wide-support from the computer graphics community. A glTF 2.0 asset is represented by a JSON-formatted file which contains the scene description. The scene can describe many basic elements of a 3D environment, such as cameras, animations, textures and the node hierarchy. To describe the geometry, the scene description points to a binary file via a URI which contains the buffer-based data, such as vertices. The key advantage of this is that large vertex data sets can be compressed and since the buffered data is already in an OpenGL layout, it can be forwarded directly to the graphics card. Textures are also supported and these too can be referenced in the scene and point to an image file via a URI. Binary glTF (GLB) is an extension of glTF, where all the glTF components such as the JSON file, binary files and image files are merged into a single binary blob.

However, some do not consider the glTF specification as a complete scene-graph. There is currently no way of including lights within the scene description without the use of an extension. Extensions are supported in glTF and the Khronos Group have developed several of their own, such as the KHR Lights or the Specular-Glossiness PBR extension.

Researchers have understood the significance of this format and much work has been conducted around the transport of glTF formats. Scully and Friston et al. describe a method of streaming glTF models from a database to an X3DOM client [22]. Schilling et al. also describes a method of streaming CityGML models using glTF and Cesium.js [20]. While these systems show that glTF is suitable as an inter-application transfer mechanism, they are focused on the client-server sharing of static models. Our focus is on glTF as a neutral format in a run-time and on the sharing of updates as multiple clients change the scene.

Beyond academic research, there is a plethora of open-source projects designed that support glTF in traditional game engines or rendering applications. The most critical to the success of this project is the range of glTF/GLB exporters and loaders available. Examples include UnityGLTF by the Khronos Group [14], Trimesh [5] by Dawson-Haggerty et al. and glTFForUE4 [3].

3 Hyperverse Transfer Protocol

3.1 System Design

We refer to our networking protocol as the Hyperverse Transfer Protocol (HVTP) [6]. This is arranged as a server-client system. Clients are responsible for managing shared glTF fragments whilst the server ensures changes made to the environment are propagated to all other clients. In our system, the shared environment is referred to as the Hyperverse. Figure 2 provides a snapshot of the components within our distributed system.

Each client maintains a GLB representation of the shared state. This representation will initially be synchronised with the state of the server when the client joins. Each client must convert the GLB representation to an engine format. Fortunately the data is already in OpenGL-compatible formats, so often this is just a case of copying data into a scene or directly to the graphics driver. Then updates must be computed and shared, as discussed below. All transport is

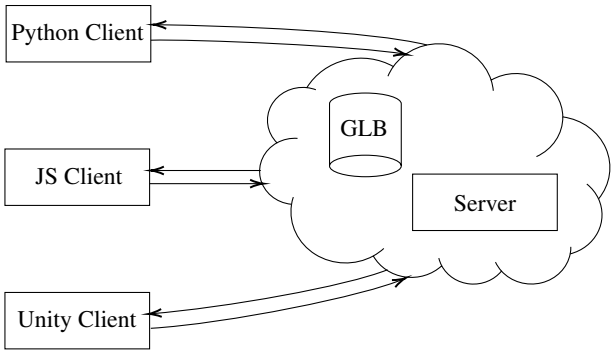


Figure 2: Message passing between clients and the server within the system.

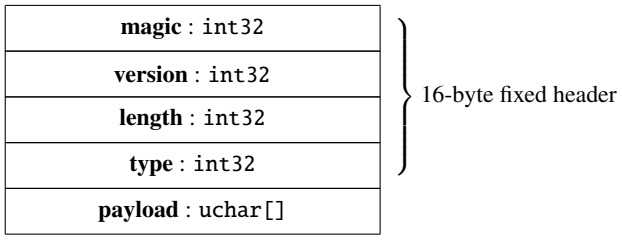


Figure 3: HVTP packet layout.

currently done over a TCP streaming socket, except in the case of the JavaScript application which uses WebSockets.

3.2 Packet Structure

Messages that are passed between the clients and the server are designed to be as simple as possible. Since this is a bidirectional protocol, it is up to both the server and the client to effectively communicate state. Figure 3 describes the packet layout for HVTP messages.

Magic A constant value of 0x48565450, which is the ASCII representation of "HVTP".

Version Describes the version of the protocol. For this proposal, it is a constant value of 0x1.

Length Indicates the logical size of the payload.

Type Describes the type of payload. It is used to signal to the client or server application how the payload should be processed.

Payload glTF fragments for transmission - see Section 3.3.

3.3 Payload Types

Table 1 lists the types of payload that can be included in a packet.

3.3.1 INIT

Once the client connects to the server successfully, it is the responsibility of the server to send the entire GLB file to the client in the payload. This type of message is referred to as an INIT type.

Table 1: Payload description per packet type.

Type	ASCII	Payload Description
INIT	0x494e4954	The complete GLB scene description
UPDT	0x55504454	A byte diff of scene changes made
TRNS	0x5452534c	Transformation to specified object

3.3.2 UPDT

We want to be as efficient as possible when describing changes that have been made to the scene-graph. If a client has modified the scene, such as introduce new geometry or textures, then it is this client's responsibility to calculate the diff of the new GLB file against the original. The diff is then transferred to the server under the UPDT message type and it is then propagated to all other clients in the Hyperverse. The clients and server each merge the diffs into their own copies of the scene-graph.

Our proof of concept implementation is simple but deals comprehensively with change: a diff is calculated as a bitwise XOR operation of the original GLB file against a new GLB file. The result of the XOR operator can be interpreted as the difference of bits across the two GLB files. Theoretically, since the number of changed bits will be less than most of the total bits, we will have more zeros than ones. This can be referred to as a sparse binary string and it can be compressed. The entire UPDT process should be faster than solely using the INIT packet type. This diff can be compressed efficiently as it will be mostly blank unless a pervasive scene change is made.

3.3.3 TRNS

One very common change in real-time mixed-reality scenes is the movement of objects, not least any avatar or other representations of the users. These updates can be frequent (60Hz+). Thus we propose to encapsulate some changes very specifically: not as diffs but as "higher-level" specific changes. Thus TRNS essentially needs to reference a binary location in the GLB, and then a transformation. We encode position, scale and a quarternion for rotatations.

This message type must also include a reference to the location of the node within the scene that we are transforming. In glTF 2.0, the node hierarchy is not represented as a scene-graph, but a disjoint union of strict trees. Since no node can be a direct descendant of more than one other node, we cannot apply a standard traversal algorithm and to indicate where the node is located in the tree. Fortunately, the glTF specification allows for an optional node "name" field. When the scene is exported to bytes, we can label each node name using a universally unique identifier (UUID).

4 Implementation

4.1 Clients

To demonstrate the interoperability of our protocol, we demonstrate three clients:

- **Unity** An example of a high-level game engine. It utilises Microsoft's .NET framework to manage code. See also Figure 4.
- **Python (using Pyrender)** A more accessible language used extensively in data science, but not necessarily in computer graphics. Paired with Python is a rendering framework Pyrender [17], which supports the glTF specification. See also Figure 4.
- **JavaScript (using Three.js)** Used increasingly in computer graphics due to its extreme portability on modern web browsers. JavaScript is also an interpreted language but is confined to the web model. Three.js is a highly-popular 3D rendering framework which uses WebGL [2].

By selecting Unity, Python and web-based clients, we can demonstrate that 3D graphics can be shared between very different languages, runtimes and rendering APIs. For this proof on concept Unity was the only engine that modified the shared environment. This was because it was straightforward to implement a 3D user interface to allow user manipulation of objects as a demonstration.

4.2 Server

In this current proposal, the server does not perform any significant processing or rendering. That allowed us to explore server engines that are optimised for message passing tasks. Two servers were implemented, one in Java and one in Python. Both used the built-in socket API.

The server has its copy of the scene-graph. Any changes that a client sends to the server will also modify the server's scene-graph in addition to the other client's scene-graphs. The main purpose of this is to have a persistent and reliable store of the environment for late joiners, or in the case that the server must restart.

4.3 Implementation Detail

An initial implementations of the INIT protocols was prototyped quickly in Unity using the GLTFSceneExporter [14]. Recall that Unity is the only source of dynamic objects in this proof of concept, and one implementation hurdle for future work is the implementation effort to construct the GLB change for the target platform. The loading applications (JavaScript, Python), can use a standard glTF importer to load buffers transferred within the INIT payload. From our experience in exploring glTF, serialisation implementations are lagging behind in some cases. However, we might expect that many clients (e.g. end-user applications) will typically load and deserialise significant amounts of data, but that any serialisation might be limited to simple updates, e.g. control of avatars. For this case, the TRNS or a similar targeted update protocol might be sufficient and these packet types are easy to construct and manage.

UPDT was prototyped by triggers that observed change in the Unity scene, and then triggering serialisation of that scene branch. A binary diff was calculated and then compressed with gzip.

To facilitate TRNS, we needed to index the GLB blob. Thus each client has an internal dictionary, where the key represents the UUID for each node within the GLB file and the value is a reference to the geometry of the node within the client scene-graph. As clients received TRNS packets, they performed a look-up of the node using the UUID and applied the relevant translations.

We note that there are some implementation hurdles to overcome, in particular the fact that some engines, e.g. Unity, are single threaded. This leads to issues with blocking on reads of large updates, which is not ideal for the mixed-reality case. There are workarounds for this. For more detail of the implementations, see [6].

5 Discussion and Conclusion

HVTP is a proof of concept of using glTF as a neutral scene-graph to facilitate real-time communication between heterogeneous rendering engines. Given glTF is designed for sharing static 3D graphic assets, our first demonstration can be compared to simply coordinating the loading of different glTF files. However, the main point is that the asset could have originated inside one of the engines, and thus be serialised from one engine in real-time. We then extended this to show that differences could be transmitted as updates, reducing the bandwidth required. Finally, we showed a real-time update extension, which deals with a practical problem of animating various objects. This can be considered to be an optimisation for specific types of common edit.

The main advantage of the SAGB approach is that the protocol can share diverse data at real-time. Indeed, it can be used to synchronise state between different applications, something that might otherwise involved the application developer writing quite significant functionality into the engine. This supports other run-time features, such as naturally supporting late joiners to the scene and not having to pre-load assets into an engine.

The main disadvantage is the the neutral scene-graph is not that amenable to sharing abstract data types or isn't necessarily efficient in some situations. For example, if a physics simulation were completely deterministic, applications clients could all run the simulation and would only need to share the starting conditions and time, rather than the complex set of results. One way to share abstract with this would be have invisible objects that represent state (e.g. a vertex array in glTF with no corresponding geometry), but this then assumes the different clients have the semantics of interpretation of such data. Another approach might be to propose extensions to glTF that specifically support the data sharing use.

In conclusion we believe that glTF deserves further study as a way to share mixed reality scenes in real-time. We have demonstrated a proof of concept that highlights that glTF is already useful for sharing scenes in real-time. glTF 2.0 proved to be a good starting point as it is already a binary format and it provides data structures amenable for direct use in real-time engines with little parsing and re targeting. For future implementations, we suggest to focus on protocols that deal with optimisations for common real-time operations on scenes that extend our TRNS payload. This can complement work that focuses on compressing the binary edits. Our own immediate future work will extend the proof of concept to more graphics engines.

Acknowledgments

Jaspreet was supported by the 2019 DeepMind Computer Science Scholarship.

References

[1] Autodesk FBX SDK programmer's guide. `http://docs.autodesk.com/FBX/2014/ENU/FBX-SDK-Documentation/index.html`. [Accessed: 2020-07-05].

[2] T. Authors. three.js. `https://threejs.org/`. [Accessed: 2020-03-05].

[3] code4game. gltfforue4 by code4game. `https://github.com/code4game/glTFForUE4`. [Accessed: 2020-08-10].

[4] W. Consortium. Basic X3D Examples Archive, Networking. `https://www.web3d.org/x3d/content/examples/Basic/Networking/index.html`. [Accessed: 2021-01-18].

[5] Dawson-Haggerty, Michael. trimesh. `https://trimsh.org/`. [Accessed: 2020-08-09].

[6] J. Dhanjan. Inter-networking heterogeneous 3d applications using the hyperverse transfer protocol (hvtp). Masters Dissertation from Department of Computer Science, University College London, 2020.

[7] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo, and J. Blat. 3D graphics on the web: A survey. *Computers & Graphics*, 41:43–61, June 2014.

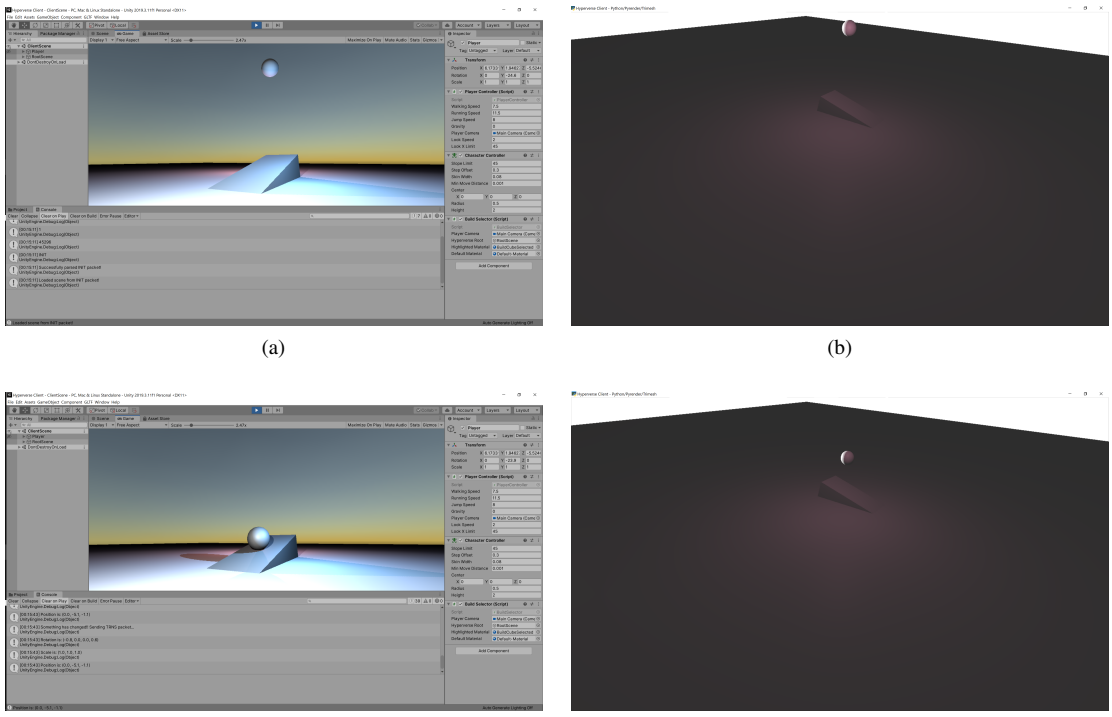
[8] Exit Games. Global Cross Platform Realtime Multiplayer Game Framework. `https://www.photonengine.com/en-US/Realtime`, 2021. [Accessed: 2021-01-18].

[9] G. Fiedler. Networked Physics in Virtual Reality: Networking a stack of cubes with Unity and PhysX. `https://developer.oculus.com/blog/networked-physics-in-virtual-reality-networking-a-stack-of-cubes-with-unity-and-physx/`, 2018. [Accessed: 2021-01-18].

[10] E. Frécon and M. Stenius. Dive: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(3):91, 1998.

[11] E. Frécon and M. Stenius. DIVE: a scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(3):91–100, Sept. 1998. Publisher: IOP Publishing. doi: 10.1088/0967-1846/5/3/002

[12] M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. V.



(a) (b) (c) (d)

Figure 4: A demonstration of the TRNS type: a ball is released from a height by the Unity client. The exact motion of this object is faithfully reproduced on the Python client in real time. (a) and (c) show Unity client, (b) and (d) show Python client. (a) and (b) show start time, (c) and (d) show later times with the ball on the Python client showing the network delay.

Moere, and O. Staadt. blue-c: a spatially immersive display and 3D video portal for telepresence. *ACM Transactions on Graphics*, 22(3):819–827, July 2003.

[13] K. Group. glTF 2.0 specification. `https://github.com/KhronosGroup/glTF/tree/master/specification/2.0`. [Accessed: 2020-07-05].

[14] K. Group. Unitygltf by khronosgroup - github. `https://github.com/KhronosGroup/UnityGLTF`. [Accessed: 2020-08-10].

[15] G. Hesina, D. Schmalstieg, A. Furhmann, and W. Purgathofer. Distributed Open Inventor: a practical approach to distributed 3D graphics. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '99, pp. 74–81. Association for Computing Machinery, New York, NY, USA, Dec. 1999.

[16] B. MacIntyre and S. Feiner. A distributed 3D graphics library. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pp. 361–370. Association for Computing Machinery, New York, NY, USA, July 1998.

[17] Matl, Matthew. Pyrender. `https://github.com/mmatl/pyrender`. [Accessed: 2020-08-09].

[18] Pixar. Introduction to universal scene description. `https://graphics.pixar.com/usd/docs/index.html`. [Accessed: 2020-07-05].

[19] B. Schaeffer and C. Goudeseune. Syzygy: native PC cluster VR. In *IEEE Virtual Reality, 2003. Proceedings.*, pp. 15–22, Mar. 2003. ISSN: 1087-8270.

[20] A. Schilling, J. Bolling, and C. Nagel. Using gltf for streaming citygml 3d city models. In *Proceedings of the 21st International Conference on Web3D Technology*, pp. 109–116, 2016.

[21] D. Schmalstieg and G. Hesina. Distributed applications for collaborative augmented reality. In *Proceedings IEEE Virtual Reality 2002*, pp. 59–66, Mar. 2002. ISSN: 1087-8270. doi: 10.1109/VR.2002.996505

[22] T. Scully, S. Friston, C. Fan, J. Doboš, and A. Steed. glTF streaming from 3D Repo to X3DOM. In *Proceedings of the 21st International Conference on Web3D Technology*, pp. 7–15, 2016.

[23] J. Signès, Y. Fisher, and A. Eleftheriadis. MPEG-4's binary format for scene description. *Signal Processing: Image Communication*, 15(4):321–345, Jan. 2000.

[24] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison Wesley, Reading, MA, Aug. 1999.

[25] A. Steed, J. Mortensen, and E. Frécon. Spelunking: experiences using the DIVE system on CAVE-like platforms. In *Proceedings of the 7th Eurographics conference on Virtual Environments & 5th Immersive Projection Technology*, EGVE'01, pp. 153–164. Eurographics Association, Goslar, DEU, Jan. 2001.

[26] A. Steed and M. F. Oliveira. *Networked graphics: building networked games and virtual environments*. Elsevier, 2009.

[27] H. Tramberend. Avocado: a distributed virtual reality framework. In *Proceedings IEEE Virtual Reality*, pp. 14–21, Mar. 1999.

[28] B. Zeleznik, L. Holden, M. Capps, H. Abrams, and T. Miller. Scene-Graph-As-Bus: Collaboration between Heterogeneous Stand-alone 3-D Graphical Applications. *Computer Graphics Forum*, 19(3):91–98, 2000.