ISSN:1816-6075(Print) 1818-0523(Online)
Journal of System and Management Sciences
Vol. 10 (2020) No. 1, pp. 94-105
DOI:10.33168/JSMS.2020.0107

3D Scene Description and Recording Mechanism: A Comparative Study of Various 3D Create Tools

Pill-Won Park
Department of Computer Science & Engineering, Dongguk University, Korea
pillwon79@gmail.com

Abstract. Various 3D VR-related content is being produced, and various 3D create tools are being developed to increase efficiency. However, the basic mechanisms of 3D create tools may be the same, but they are different from each other because they had different goals. As a result, content created in one 3D create tool is often not available in another 3D create tool. Storage methods compatible with various 3D create tools have also been proposed, but the research is still insufficient. In this paper, we propose a technique that analyzes and classify 3D content, so other 3D create tools can use it, too. The proposed technique is to classify 3D contents into asset, hierarchy, and script, and then re-analyze and mainly focused on the hierarchy and script transformation. Since the standard of content conversion is needed, WEB 3D create tool is used as a standard for 3D content analysis. Hierarchy and scripts were to reduce as much as possible to minimize the confusion in the WEB 3D create tool form. We proved that this proposed technique can be used in Unreal by transforming 3D content data of WEB3D create tool Babylon.js.

Keywords: Computer Graphics, Language transfer, 3D Computer Graphics, WEB-3D, Web-VR

1. Introduction

As the technology of 3D graphics (Watt and Alan, 1993) develops, 3D contents have begun to be popularized, and various 3D create tools for developing them are being developed and improved. Each 3D create tool has a different direction from the producers, so the development direction and advantages of the create tool are different.

Therefore, content created in one 3D create tool cannot be used in another 3D create tool due to a difference in storage method, except for a 3D object (hereinafter, referred to as an asset) which is an element independent from the 3D create tool. This problem can cause considerable inconvenience in situations where different 3D

create tools must be used or content collaboration must be done remotely.

Therefore, in this paper, we propose a method of formalizing and storing 3D content so that the same content can be implemented in different 3D create tools. In addition, to prove the proposed theory, the 3D production of Babylon.js, a real web 3D create tool, was formalized, saved, and then loaded and verified in Unreal.

2. Related works

2.1. Computer graphics

Computer graphics, which began with the development of computers, began to grow explosively in the 1960s, thanks to the widespread use of displays and the advantages available in various fields. Based on the logical data, it is possible to create a screen that is hard to see in reality because it is not bound by the limitations of reality, and its use has been expanded quickly because the data is easily recycled and improved.

Since the data is processed into mathematical expressions to provide the user with shapes and colors, there is bound to be a deep relationship with mathematics to process them. It is common to use a sequence using the RGB technique as a method of describing the color, and the shape is generally represented by a sequence based on a coordinate axis and processed. There are two ways to rotate an asset: Euler, which describes the angles of three axes, and Quaternion, which is based on the three-dimensional arithmetic of the vertex.

2.2. Object Rotation

Rotation of assets in three-dimensional space is generally described by using Euler angles (Arfken, 1985) and quaternion (Kuipers, 1999).

The Euler angles describe the three axis angles to describe how much the asset has rotated. The rotation information about each axis is converted into a rotation matrix for each axis and multiplied to express how much the asset is rotated in the 3D environment. This approach is intuitive to use angles. However, in order to calculate a rotation, a matrix operation must be performed, and when the axis overlaps, a gimbal lock (Mukundan, 2002) that can no longer be rotated occurs. In order to solve this problem, a quaternion method has been proposed.

The quaternion method uses rotation to describe how much the asset has been rotated using vectors in complex space. While Euler's method uses three variables of x y z for three-dimensional computation, the quaternion's method is described by four methods including imaginary numbers. Unlike Euler angles, rotations can be described by simple four-step calculations rather than matrix calculations, resulting in fast computations and no gimbal-lock problems. However, it is not intuitive to describe rotation as a complex number rather than an angle. Therefore, in many 3D create tools, it is common to take Euler-type input and transform it into quaternion in the create tool. (Diebel, 2006; Verth, 2008).

2.3. 3D Create Tool

Since 3D graphics are made up of logical combinations of numbers, it is almost impossible for a user to create contents by directly inputting numbers. In addition, various physical techniques must be included for natural content production, and complex operations such as particles need to be patterned and processed. Because of this reasons, various 3D create tools have been developed for convenient 3D content development.

Representative 3D create tools are the Unreal engine and Unity. Both 3D create tools have the advantages of high reliability, high engine performance, and the ability to create 3D content running on multiple platforms. Unity supports simple basic functionality and those are not, it can be done by purchasing an asset or adding features. Unreal Engine provides most of the powerful features with the engine itself, but has a relatively complex structure that requires high user experience.

Scripting features that describe the interaction between assets also support visual scripting and complex programming in Unreal and Unity. Unreal supports its own visual scripts called C ++ and Blue Print, while Unity supports C # and Unity visual scripting. Specially, Unreal's visual script, Blue Print, can be used to create simple content even if you're not a professional programmer.

However, even though the basic theory and appearance of these 3D create tools are similar, the internal mechanisms and recording methods are inevitably different because the important factors and development directions that developers are different. As a result, it is virtually impossible to load content developed by one create tool into another. In addition, as the quality of 3D content increases, the capacity required for content storage increases exponentially, making it difficult to share content over a network. As a result, collaboration between developers, who must use different 3D create tools, or rapid collaboration over networks has become almost impossible. Even if the problem with the content storage capacity is solved by separately sharing an asset that is independent of the 3D create tool, the problem of incompatible 3D scene data such as asset placement is not solved.

2.4. Web 3D Create Tool

Professional 3D create tools mentioned above can produce high quality 3D content similar to reality. However, in order to show the contents produced by oneself to others, there is a disadvantage that the entire contents and its related works must be delivered to the other party. Even small tasks like simple concept design production rather than regular production content can consume significant capacity and computing power.

A web 3D create tool that works in an Internet browser is a good solution for this situation. Many web 3D create tools run on JavaScript engines such as three.js (online). In addition, it is very easy to share by simply passing the URL to the other party due to the nature of HTTP. In addition, there is an advantage in that the

content can be output in a desired format by adding a simple plug-in due to the characteristics of the Internet browser.

However, using a simple JavaScript-based engine and being limited by the Internet browsers rather than specialized 3D create tools, the quality is not high. Representative Web 3D create tools are A-frame and Babylon.js.

2.5. JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is an open standard format for carrying data consisting of "attribute-value pairs and array data types or any other serializable value". Generally used to exchange data or variables between programs on the Internet. There is no limit to the kind of data to be delivered, and there is an advantage in using a general text format. As the name implies, it is derived from the JavaScript language and follows the syntactic form of JavaScript but is a highly versatile data format because it is not dependent on any programming language.

2.6. GL Transmission Format (glTF)

glTF (GL Transmission Format) is a file format for expressing 3D scenes and models. It is based on JSON. It is a standard established by the Khronos Group in 2016, and it is a file format that emphasizes efficiency and compatibility on various platforms.

As one of the sharing standards for the latest 3D data, it can be used in various 3D create tools. It is also being enhanced to embed geometry attributes such as vertex coordinates, connection information, texture coordinates, color information, normal vectors, and so on. As a result, the data has been significantly reduced, allowing users to download apps, scenes, and models faster, allowing browsers to load 3D graphics faster, and to transmit VR and AR scenes with less bandwidth.

However, because it is a recently proposed standard, there are still differences in interpretation in 3D create tools, which may cause some loading problems. In the case of Unreal Engine, a glTF file that combines multiple objects is recognized as separate parts. For example, a clothed mannequin, glTF, has only one loaded glTF file, but when it is loaded, the mannequin and the clothes are recognized as separate objects. However, this problem is a matter of interpretation, not a structure, but a problem to be solved by upgrading the glTF format and the glTF interpretation tool.

3. Proposed Scheme

3.1. Architecture

The structure assumed in this paper is shown in Figure 1.

This paper, the goal was to convert 3D scene data so that clients can use the contents stored on the Internet in Web 3D create tools or commercial 3D create tools. Therefore, wanted to standardize the content created by the client using the Web 3D create tool so that the same screen can be seen by the 3D create tool on the

other client.

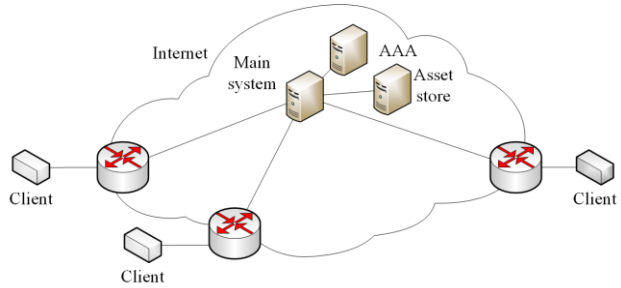

Fig. 1: Proposed Architecture.

In this paper, the components of 3D content are classified as follows.

- Asset which is 3D objects and a combination of 3D objects
- Multimedia files used in 3D scenes
- Hierarchy, a layout and relationship diagram of 3D objects
- Script that describes behavior patterns of assets.

Among them, we focused on the conversion of hierarchy and scripts, except for assets and multimedia files that are independent of 3D create tools. Since the delivery method of the multimedia file or the asset does not belong to the subject of this paper, it is assumed that it is received in advance, not covered in this paper. In addition, although there are various web 3D create tools, this paper is based on Babylon.js, but it can be applied to other similar web 3D create tools.

3.2. Procedure

The procedure in this paper is as follows.

The 3D content created by the Web 3D create tool is downloaded as shown in Figure 2. 3D content provides hierarchy data and script data as Json files, and Parser transforms the data into a predefined format. The 3D create tool then accepts the data through plug-ins and reconstructs the 3D content.

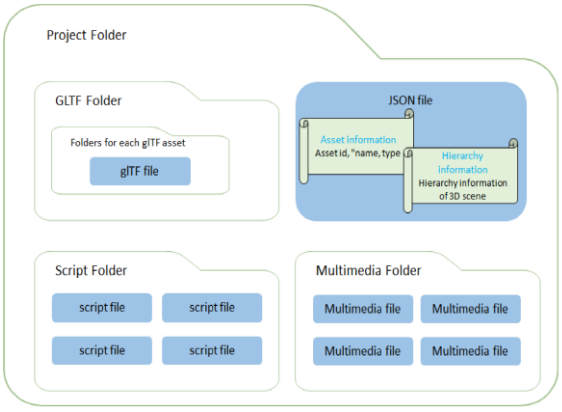

Fig. 2: Contents Formalized Form.

3.3. Asset information

Json files that describe 3D content contain asset information, hierarchical information, and linkage to scripts.

The asset information has an ID for distinguishing it from other assets, a name that is displayed externally, and a type field indicating characteristics of the content. For example, the {"id": "0004", "name": "Avocado", "type": "GLTF"} described in the asset field have a displayed name of Avocado for asset with ID 0004 The type is glTF.

It consists of a component that contains other information, and a script field that points to the script to link to. For example, hierarchy is the part where the relationship information of assets called hierarchy in Unity and world outliner in Unreal is described. In most 3D create tools, asset management is managed in a tree form, and Babylon.js, Unreal, and Unity are also managed in that way.

3.4. Hierarchy information

Hierarchy contains information about hierachyID for hierarchy placement, parent pointing to its parent node, type pointing to its type, name indicating external name, assetID indicating the ID of the asset to be referred to, and transform with position rotation scale information. For example, "hierachyID": "0001", "parent": "root", "type": "camera", "name": "Main Camera", "assetID": "001", "transform": {" position ": {" x ": 40," y ": 10," z ":-30}," rotation ": {" x ": 15," y ": 0," z ": 0}," scale ": {" x ": 1," y ": 1," z ": 1}}," component ": {" script ": []} means that text in hierachyID 0001 placed in Assets of camera type belonging to root. The name is Main Camera, and the ID of the asset to be referred to is 001. Position information is (40,10, -30), rotation information is (15,0,0), and scale information is (1,1,). There is no specific content in the component field where other information is described, and there is no script associated with the asset.

The reason for keeping the asset ID and the hierarchy ID separately is that the asset ID is for identifying the asset, and the ID of the hierarchy is data for describing position and which asset are connected to the hierarchy. Therefore, you must have both IDs in order to correctly configure the 3D scene.

3.5. Script information

The script describes how objects should behave in certain situations. For example, if an object is moved to a pre-specified point under certain conditions, a variety of descriptions such as condition determination, location determination, and speed of movement are required. Much of this is done in the physical engine, but basic parameters need to be described, so a description of that technique is required.

Scripts used on the web were analyzed and grouped as shown in Figure 3 by function. For example, the AddVariable function in the User Variables category is a function that is responsible for adding the object attribute variables required to perform a special function in a scene, and the Input Parameters are basically

Boolean-type, and can receive array, float-based tuple. In addition, a string to describe the function name could be obtained and stored in a non-return manner.

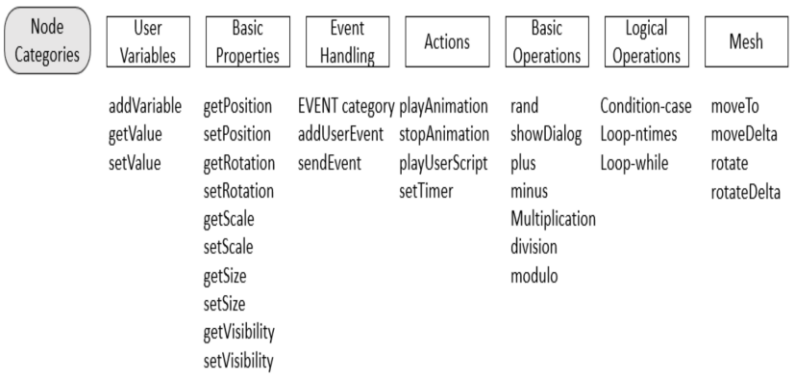

Fig. 3: Script functions.

However, various high-level scripts supported by commercial 3D create tools in the web editor cannot be supported. Therefore, similar functions are treated as a combination of representative functions or simple functions. For example, in Unreal, there are many functions related to moving objects, but this paper only supports moveto and moveDelta.

3.6. Date formalized

Figure 4 shows a modified version of Babylon.js, the web editor targeted in this paper.

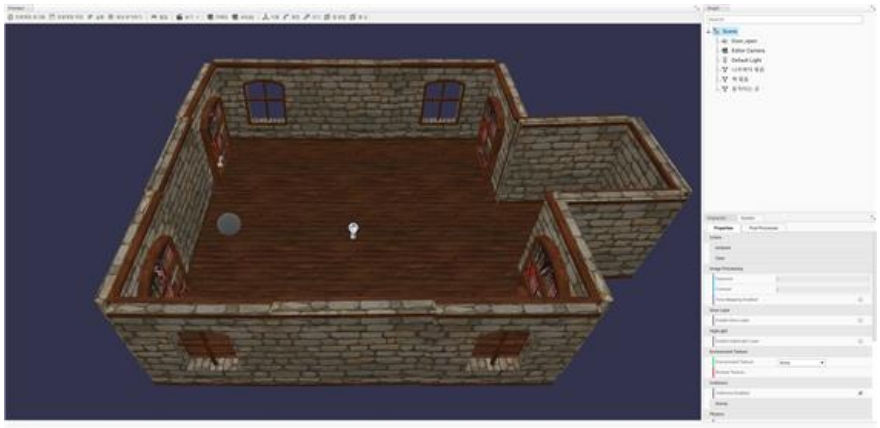

Fig. 4: Modified Babylon.js.

The 3D scene is depicted on the left side of the picture, and the Hierarchy information and asset information are described in tree form on the Graph tab at the top right.

At the bottom right is the Inspector menu, which displays detailed information about the selected object.

In the graph tab, you can designate one group and add various assets to the group.

One folder and glTF are allocated to one asset, and the corresponding folder and asset portion include information on higher objects and position, rotation, and scale information of the object.

However, in case of default Babylon.js, the position, rotation, and scale information of the object may be stored scattered in the folder and asset information, so it is necessary to be careful about this.

Figure 5 shows the graph field for Babylon.js.

The inverted triangle icon in the graph field of Babylon.js means that it contains significant data.

There is a folder named "Wooden Floor" in the folder called "Wooden Floor Bundle" and an asset called "XR_Floor_004" is located under it. Since the part actually used is data called XR_Floor_004, it is necessary to organize the parts except those parts.

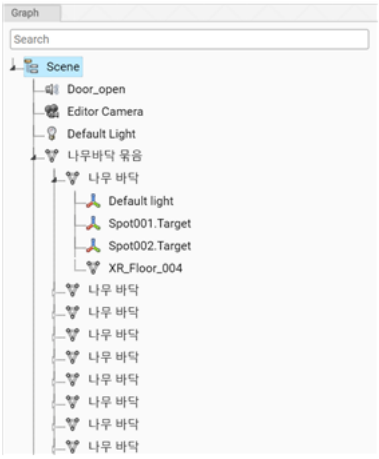

Fig. 5: Graph field of Babylon.js.

Basic Babylon.js stores all asset data, hierarchy information, and scripts in a single file inside the storage file. Therefore, unnecessary information is entered in this paper assuming that glTF is delivered separately. Of course, asset data can be extracted and used from Babylon.js's stored data, but it is necessary to provide it separately for use in other 3D create tools. In addition, it is judged that making an asset by re-interpreting vertex data is out of the scope of this paper.

Generally, 3D content data contains a wide variety of information. Babylon.js is divided into Global, morphTargetManagers, Lights, Cameras, Animations, ReflectionProbe, Materials, MultiMaterial, Skeletons, TransformNodes, Geometries,

Meshs, ParticleSystems, Actions, and more.

Global contains information such as whether the physics engine is applied and gravity value, morphTargetManagers contains information about morphTarget.

Lights contain data related to the light source. Cameras include the maximum and minimum distance to the camera and manipulation information. Animations contains frame and types information of the animation. ReflectionProbe contains rendering related data. Materials contain material-related information, and MultiMaterial contains data on how complex materials are handled. Skeletons contain the skeleton data to be applied to the asset with the bones value. TransformNodes contain data related to the transform and whether or not it can be transformed. Geometries describe data of assets such as cylinders and cubes that are basically provided. Meshes describe the id, type, position, rotation, rotationQuaternion, and scaling values for the asset. ParticleSystems records particle-related data.

In this paper, decided to format and store Global, Lights, Cameras, Geometries, and Meshes, which are necessary for the composition of contents, and describe the data as a json file as described above.

Since 3D hierarchical data contains unnecessary information including vertex data of an asset, the 3D hierarchical data is arranged in the following order.

1. Delete information that is not recorded in the asset and hierarchy fields.

2. Classify each item : Camera, Transform Node, Mesh, Light, Sound File, etc.

3. Remove unnecessary elements in lights, cameras and meshes.

3-1 Remove unnecessary elements in a light by loading a light file.

3-2 Import Mesh Files to Remove Unnecessary Elements in the Mesh.

3-3 Remove unnecessary elements in the camera by loading camera files.

4. Import mesh file and delete folder asset data (dummy folder).

4-1 Remove Unnecessary Elements Once Again in Mesh Files.

5. Import steps 3 ~ 4 of light, camera and mesh data and merge them into asset list.

6. If the folder contains the location information of the assets in the mesh, merge the asset location and the folder location information.

6-1 Changing the Rotation Quaternion Values in the Mesh Data to Rotation Values Using Euler Functions.

7. Creating asset and hierarchy structures.

7-1 asset - Merge data by bringing mesh data of step 6 and only folder location information of asset.

7-2 Hierarchy - Merging 6 levels of mesh data, 5 levels of light and camera information asset data.

Fig. 6: first step of Json Parser.

In this paper, Json parsing is separated into two stages.

Basically, it consists of 1 ~ 5 steps of filtering out unnecessary data and 2 steps of creating actual data based on refined data.

The first step handles the previous steps 1-5, and includes a folder removal structure to simplify unnecessary data when multiple inheritance occurs, such as a prefab structure.

In step 2, a Json file is generated as follows based on the stored asset list.

Fig. 7: Example of Asset data after PARSING.

The Json file describes asset data in the form shown in Figure 7. Included information is the asset name, asset id, data type, the location of the actual glTF data, whether it can be modified, and the associated Hierarchy ID.

After describing the asset data, hierarchical information on how to arrange the assets is described.

{"name":"XR_Floor_004","id":"35c9d409-37e3-4ef9-a255-499150ce4be5","rotationQuaternion":[-4.371139E-08,4.371139E-08,-4.371139E-08,1],"scaling":[1,1,1],"parentId":"4c505b68-ebec-4c05-a8bb-eea3c1df9909","metadata":{"gltf":{"pointers":["/meshes/0/primitives/0","/nodes/0"]},"baseConfiguration":{"isPickable":true}},"animations":[],"ranges":[],"HierachyID":9,"rotation":[{"x":-3.1415925,"y":3.1415925,"z":-8.74227837E-08}],"position":[0,0,0],

Fig. 8: Example of hierarchy information.

The interpretation of the hierarchical data is as follows.

An XR_Floor_004 item with an asset ID of 35c9d409-37e3-4ef9-a255-499150ce4be5. The Hierarchy ID is 9.

The asset is a lower node of an asset having an ID number of 4c505b68-ebec-4c05-a8bb-eea3c1df9909. The type is glTF and no script node is connected. This asset can also be clicked or pinched as described in isPickable, and no animation effects are applied. It has a rotation value of (-3.1415925, 3.1415925,-8.74227837E-08), a scaling value of (1,1,1), and a position value of (0,0,0).

As a problem that may occur during this conversion, the problem of the coordinate axis is likely to occur. However, if a consistent analysis is maintained and the Z axis is clear, there is no problem, so the axis description method should be unified as defined in advance. In this paper, although it is described as Y Z X axis, it is judged that there is no limitation of the technique if it can be equipped with uniformity in advance.

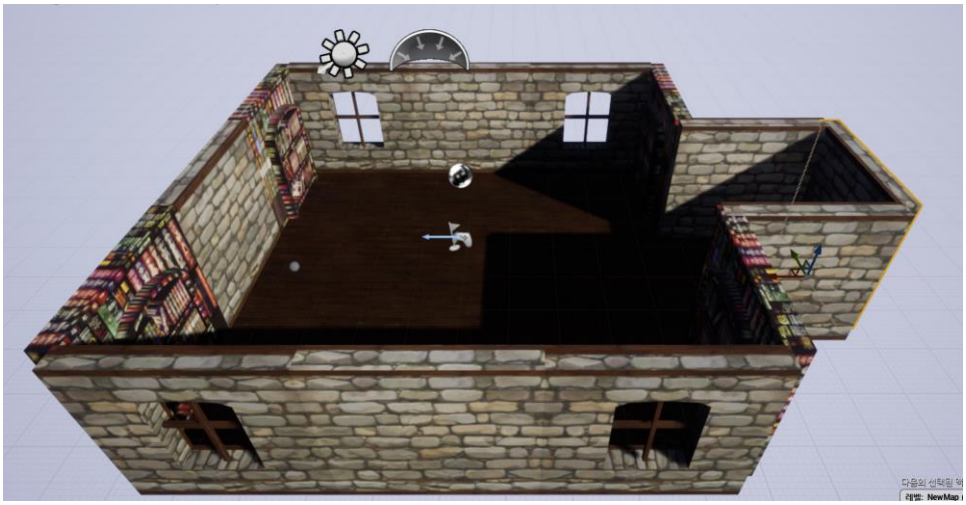

Fig. 9: Unreal loading screen of Json file.

Figure 9 shows the output of Unreal loading the summary file of the Web 3D create Tool. Compared to the web asset shown in Figure 4, the texture of the bookshelf can be found to be inconsistent, but the loading and placement of the assets except for the corresponding elements can be seen as normal. The texture inconsistency in the bookshelf part is due to the lack of Unreal's glTF importer and you can expect it to load normally when the feature is upgraded.

4. Conclusion

This study proposed 3D scene description method and mechanism that can be used in various 3D create tools. The basic data that forms the 3D scene is formatted and stored, and the data is read and reconstructed from other 3D create tools. The

proposed technique was successful to reconstruct the content which is written in Babylon.js, the actual web 3D create tool, in Unreal. A separate plug-in was required to accept the data in Unreal, but it was proved that other 3D create tools could share the content in the method proposed in this paper.

5. References

A-frame. Retrieved from https://aframe.io/

Arfken, G. (1985). *Mathematical Methods for Physicists, 3rd ed.* Orlando, FL: Academic Press, 198-200.

Babylon.js. Retrieved from https://www.babylonjs.com/

Blueprint. Retrieved from https://docs.unrealengine.com/ko/Engine/Blueprints/Getting Started/index.html

Diebel, J. (2006). *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. Retrieved October 20, 2006, from Stanford, California 94301–9010.

glTF. Retrieved from https://www.khronos.org/

JSON. Retrieved from https://tools.ietf.org/html/rfc7159

Kuipers, JB. (1999). *Quaternions and rotation sequences*.

Mukundan, R. (2002). *Quaternions: From Classical Mechanics to Computer-Graphics, and Beyond.*

three.js. Retrieved from https://threejs.org/

Unity. Retrieved from https://unity.com/kr

Unreal engine. Retrieved from https://www.unrealengine.com/

Van Verth, J.M. (2008). "*Essential Mathematics for Games and Interactive Applications: A Programmer's Guide (The Morgan Kaufmann Series in Interactive 3D Technology)*.

Watt & Alan. (1993). *3D Computer graphics*. Addison-Wesley, 1-13.