Journal of Theoretical and Applied Information Technology
30th April 2019. Vol.97. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A NEW LIGHTWEIGHT FILE FORMAT BASED ON FBX FOR EFFICIENT 3D GRAPHICS RESOURCE PROCESSING

[1]TAEK-SOO JEONG, [2]YOUNGSIK KIM

[12]Dept. of Game and Multimedia Engineering, Korea Polytechnic University, Republic of Korea

E-mail: [1]xortn3745@kpu.ac.kr, [2]kys@kpu.ac.kr (corresponding author)

ABSTRACT

The FBX format is one of the most popular graphics formats for 3D graphics games. However, the FBX format contains a lot of information for general use. Thus, when producing real 3D graphics games, it is inefficient in terms of file size and 3D graphics rendering, because the FBX file contents are not used at all. This paper proposed a new lightweight FBX file format based on the conventional FBX format. Using the new FBX format created by extracting only the necessary information, the 3D graphics resource file capacity and loading time are reduced. Using the new FBX file format, this paper reduced the size of resource files by an average of 81% and reduced resource loading time by an average of 51%. This experiment proved its efficiency by comparing it with the pure FBX format under the same conditions.

Keywords: *3D Game, DirectX, FBX Format, 3D Graphics Resource, Loading Time*

1. INTRODUCTION

The global games market will reach $108.9 billion in 2017 with mobile taking 42% [1]. A variety of methods can be used for game development, and a performance game engine has recently been developed due to the growth of the game market [2]. The quality of the game released by the game engine has risen [3]. This means that the game engine has become the force of game development [4]. It is also possible to distribute the developed engine code as open source, and engine that can modify artificially by the programmer is on the increase, and quality contents can be developed through level editor [5].

As the quality of the game rises, several methods have been applied to reduce the amount of computation through game scene management and optimization. For example, it is possible to shorten the rendering time by dividing a scene using BSP tree [6] or to output it effectively by dividing the space by using an octree or quadtree [7] and processing the same phase polygons as one [8]. In addition, if an invisible scene to distinguish and to enter the pipeline, even if the object is an object that overlaps were also introduced algorithms that determine whether in front of the camera [9]. But these are optimization methods after loading of the 3D graphics resources.

The geometry (or shape) of a model is often stored as a set of 3D points (or vertices). The surface of the model is then stored as a series of polygons (or faces) that are constructed by indexing these vertices. The number of vertices the face may index can vary, though triangular faces with three vertices are common. Some formats allow for edges (or lines) containing two vertices [10].

FBX (Filmbox) is a proprietary file format (.fbx) developed by Kaydara and owned by Autodesk since 2006. It is used to provide interoperability between digital content creation applications. FBX is also part of Autodesk Gameware, a series of video game middleware [10].

Autodesk provides a C++ FBX SDK that can read, write, and convert to/from FBX files. The FBX file format is proprietary, however, the format description is exposed in the FBX Extensions SDK which provides header files for the FBX readers and writers [11].

There are two FBX SDK bindings for C++ and Python supplied by Autodesk. Blender includes a Python import and export script for FBX, written without using the FBX SDK and The OpenEnded Group's Field includes a Java-based library for loading and extracting parts from an FBX file [11].

This paper proposed a new lightweight format itself is based on the FBX file format to reduce the time and capacity to load a 3D graphics resource. The proposed method extracts the information needed for object rendering and creates a new format to speed up data processing, improve rendering speed, and reduce disk usage. We assumed that the information in the self-format was

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1. A Few 3D File Formats [10].

Extension	Name	Description
3ds	3D Studio	The 3ds file format is the primary format of AutoDesk's 3ds Max software. It is a binary format consisting of chunks that hold various pieces of information. Chunks contain an identification indicating what information is stored there and the offset to the next chunk [13]. In this way software that doesn't support certain rendering properties can simply ignore them. The 3ds file format supports geometry in the form of vertices/faces and parametric surfaces, textures, physical material properties, transformations, camera information, and lights.
obj	Wavefront	The obj file format is a text-based, open file format developed by Wavefront Technologies (now Alias|Wavefront) [13]. The format has been adopted by other 3D graphics applications vendors and can be imported/ exported by a number of them. The obj file format consists of a number of lines each containing a key and various values. The key on each line indicates the type information to follow. Because of this obj file format doesn't require a header. Below is a list of some of the keys that can be used:
igs	Initial 2D/3D Graphics Exchange Specification	The Initial Graphics Exchange Specification (or IGES) format, published by the National Bureau of Standards in 1980 (NBSIR 80-1978), is a popular neutral format for digital the exchange of CAD information. The iges format is designed to store both 2D and 3D data.
ply	Standford PLY	The polygon file format (or Stanford triangle format), was designed for the purpose of being both a flexible and portable 3D file format [13]. The ply format has both an ASCII and a binary version. The binary version includes information to make it machine independent, specifying the types used for each value, number of bytes per type, and whether it's big or little endian. In addition, the format allows for user-defined types allowing it to be extensible to the needs of future 3D data. Because of its simplicity and flexibility the ply format is very popular in the academic and research world.
stp	Standard for the Exchange for Product Data	The Standard for the Exchange for Product Data, ISO 10303, was developed as a successor to the iges format. The step format is a plain text format that deals with named objects rather than just raw geometric information [13].
u3d	Universal 3D	The Universal 3D format [14] was developed by the 3D Industry Forum which consisted of companies such as Intel, Boeing, Adobe and HP. Their goal was a universal standard for 3D data of all kinds that would facilitate exchange with a focus on promoting 3D graphics development in manufacturing, construction and various other industries. The format was approved in 2005 by the European Computer Manufacturer Association (Ecma-363). The format is largely backed by Intel who began work on it after leave the Web3D Consortium after disagreements over x3d [15]. Like x3d this format is intended to be the 3D standard. It has since been adopted by Adobe to embed 3D graphics within PDF documents.
fbx	AutoDesk Kaydara FBX	FBX (Filmbox) is a proprietary file format (.fbx) developed by Kaydara and owned by Autodesk since 2006. It is used to provide interoperability between digital content creation applications. FBX is also part of Autodesk Gameware, a series of video game middleware [10]. Autodesk provides a C++ FBX SDK that can read, write, and convert to/from FBX files. The FBX file format is proprietary, however, the format description is exposed in the FBX Extensions SDK which provides header files for the FBX readers and writers [11].

Journal of Theoretical and Applied Information Technology
30th April 2019. Vol.97. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

saved as much as possible and saved only the basic information that can be seen in the scene except for the animation. And the efficiency of each situation is proved through an experiment.

This paper analyzes and tests based on DirectX11, and assumes that Draw and DrawIndexed functions using vertex and index are already implemented.

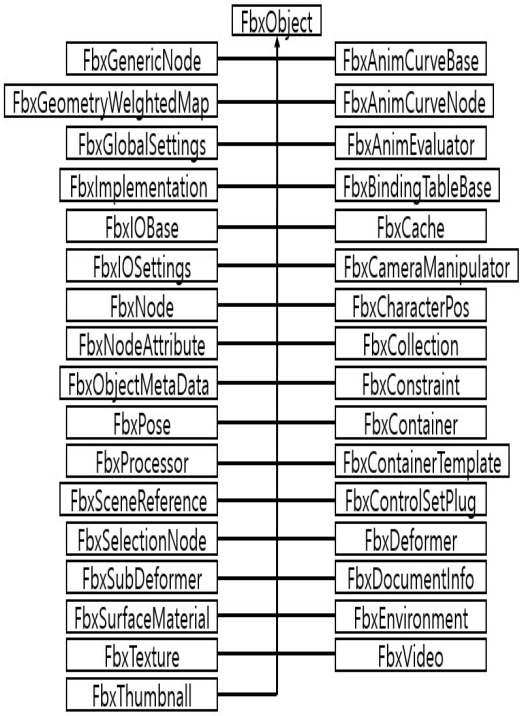


Figure 1. The Inheritance Diagram for FbxObject.

2. FBX FILE FORMAT

Table 1 shows a few 3D file formats. An FBX object is an instance of another class derived from FbxObject or FbxObject [12]. FbxObject has 33 classes that can be used to construct a 3D model as shown in Figure 1. This paper adopts the method of reading Node by using FbxScene object. FbxScene also inherits the FbxObject as the top-level parent, so it can fetch the information needed to configure the FBX as shown in Figure 2 (a). FbxScene is again composed of a node hierarchy. They can get the root note value through FbxScene::GetRootNode (), and they can navigate the child root of the tree structure through the root node and search the necessary information in Figure 2 (b).

The top node has FbxMesh information and can read the number of polygons and the number of vertices of current mesh data using GetPolygonCount function and GetControlPoints function of FbxMesh. They can read the material of the current polygon as many times as the number of polygons. They can read information of vertex position, Normal, UV, Tangent, and Binormal of polygons constituting polygon by using GetPolygonSize function.

As in Figure 4, to get normal information, first use GetElementNormalCount function of FbxMesh. This function reads the number of normal elements. The normal element information is stored while repeating the loop as many times as the number of normal elements.

As shown in Figure 5, To get the UV value, use the GetElementUVCount function to get the UV number. As in the case of obtaining the above normal information, UV element information is read while repeating the number of UVs. UV can be saved as FbxVector2 type. Tangent information and BiNormal information Also, the GetElementTangentCount function and GetElementBinormalCount function are used to find out the number and then read the information through the loop. Finally, read the material information and read the necessary information such as material type, diffuse, ambient, specular, emissive, power and texture.

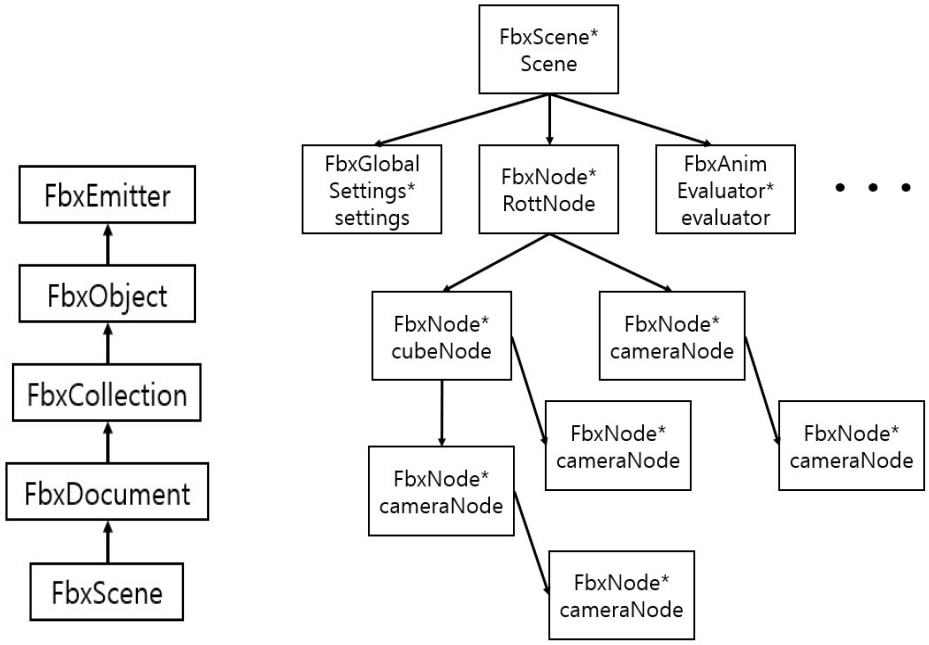
3. A NEW LIGHTWEIGHT FORMAT BASED ON FBX

As shown in Figure 1, FBX contains information that is not used, which is inefficient in terms of capacity usage. If they extract only the information they need and reformat it based on that information, they will see greater efficiency in terms of capacity usage.

Table 2 shows that the size of the FBX format is about 7.40 MB. However, the proposed format, which extracts only the necessary information and reformats it, is about 1.88MB in size. If the number of objects in one scene in the game is more than 100 and 200, then the difference will be bigger.

Journal of Theoretical and Applied Information Technology
30th April 2019. Vol.97. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



(a) FbxScene Parent Classes (b) FbxScene Organization

Figure 2. FBX Scene.

Part of Source Code for Extracting Information that Makes Up Polygons

```
    // Find out the number of polygons in the mesh data.
1.  int iPolygonCount = pMesh->GetPolygonCount();
2.  for (int i = 0; i < iPolygonCount; ++i)
3.  {
4.    int iPolygonSize = pMesh->GetPolygonSize(i);
5.    int iMaterialID = -1;
      // Retrieves the material ID of the current polygon.
6.    FbxGeometryElementMaterial* pMaterial = pMesh->GetElementMaterial(0);
7.    iMaterialID = pMaterial->GetIndexArray().GetAt(i);
8.    for (int j = 0; j < iPolygonSize; ++j)
9.    {
     // Read vertex location information.
10.     int iControlIndex = pMesh->GetPolygonVertex(i, j);
       // Read other information such as Normal, UV, etc..
11.     LoadNormal(...);
12.     LoadUV(...);
13.     LoadTangent(...);
          ...
14.   }
15. }
```

Figure 3. Part of Source Code for Extracting Information that Makes Up Polygons.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Part of Source Code for Extracting Normal Information

```
    // Obtain the number of normal elements.
1.  int iNormalCount = pMesh->GetElementNormalCount();
2.  for (int i = 0; i < iNormalCount; ++i)
3.  {
4.    if (pNormal->GetMappingMode() == FbxGeometryElement::Mapping_Mode)
5.    {
6.      FbxVector4 vFbxNormal;
7.      FbxVector4 vNormal;
8.      int iNormalIndex = 0;
9.       switch(pNormal->GetReferenceMode())
10.     {
11.        case Reference_Mode:
12.          vFbxNormal = pNormal->GetDirectArray().GetAt(iVertexID);
          // FbxVector4 is stored in array format
          // 0 is for x or r
          // 1 is for y or g
           // 2 is for z or b
13.       vNormal.mData[0] = vFbxNormal.mData[0];
          ...
14.      }
15.   }
16. }
```

Figure 4. Part of Source Code for Extracting Normal Information.

Part of Source Code for Extracting UV Information

```
    // Find out the number of polygons in the mesh data.
1.  int iPolygonCount = pMesh->GetPolygonCount();
2.  for (int i = 0; i < iPolygonCount; ++i)
3.  {
4.    int iPolygonSize = pMesh->GetPolygonSize(i);
5.    int iMaterialID = -1;
      // Retrieves the material ID of the current polygon.
6.    FbxGeometryElementMaterial* pMaterial = pMesh->GetElementMaterial(0);
7.    iMaterialID = pMaterial->GetIndexArray().GetAt(i);
8.    for (int j = 0; j < iPolygonSize; ++j)
9.    {
     // Read vertex location information.
10.     int iControlIndex = pMesh->GetPolygonVertex(i, j);
       // Read other information such as Normal, UV, etc..
11.     LoadNormal(...);
12.     LoadUV(...);
13.     LoadTangent(...);
          ...
14.   }
15. }
```

Figure 5. Part of Source Code for Extracting UV Information.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2. File Size Comparison between the conventional FBX and the Proposed Format.

Items	The Conventional FBX	The Proposed Format
Name	Baraka.FBX	Baraka.xxx
Category	FBX file	XXX file
Size	7.40MB	1.88MB

Part of Source Code for Storing the Mesh Data and Location Information

```
        //m_vecMesh is stored mesh information
        //m_vecMesh.size() is the number of stored informations
17. for (iterData = m_vecMeshData.begin(); iterData != m_vecMeshData.end();  ++iterD
    ata)
18. {
19.   //the number of polygons
20.    fwrite(&(*iterData)->iPolygonCount, sizeof(UINT), 1, pFile);
21.    // Store location information of mesh data
22.    fwrite(&(*iterData)->vecPos.size(), 4, 1, pFile);
23.    A::iterator DataPos = (*iterData)->vecPos.begin();
24.    A::iterator DataPosEnd = (*iterData)->vecPos.end();
25.    for (DataPos; DataPos != DataPosEnd; ++DataPos)
26.    {
27.     fwrite(&(*DataPos), sizeof(FbxVector4), 1, pFile);
28.     }
29.  }
```

Figure 6. Part of Source Code for Storing the Mesh Data and Location Information.

(A : vector<FbxVector4>).

To create the new format, this paper must have the information you want to store in the FBX file in its own format. The information is traversed by iterators and the format is saved by file I / O. The proposed format was created with the information they saved in the previous section. First, the number of polygons and the location of mesh data are stored. The location information is circulated and stored as much as the location information contained in the vector as in Figure 8. The reason for fwrite size of position information first is to read the information of the mesh exactly as it is stored when loading the format as shown in Figure 6. Next, the normal information and UV information of the mesh data are stored. As with location information, fwrite the size of the information to prevent it from reading the range of other data when reading as shown in Figure 7.

Next, save the Tangent and Binormal information using the above code. Finally, save the material information.

Since there is no guarantee that a single mesh will have only one material, the size of the material must be found and the size value must first be written. Similarly, the loop is looped to store the read information. Once you have saved your own format, you should also write code to read your own format. Since the file was created by input / output, the order of reading should be the same as the order of saving. This is why this paper first stores each size before this paper saves it. Normal, UV information is read. Since the normal information is stored first when storing, the normal information is read first as shown in Figure 7. The remaining Tangent and Binormal values are read as above, and then the material values are read in the order in which they are stored.

Figure 9 (a) is the image when the original FBX format is read and rendered and Figure 9 (b) is the image when the proposed format is read and rendered. The information needed is the same, so it can not fall behind in quality.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Part of Source Code for Storing the Mesh Normal and UV Information

```
// Let the iterator of m_vecMeshData write the size of the data first.
fwrite(&(*iterData)->vecNormal.size(), 4, 1, pFile);
A::iterator DataNormal = (*iterData)->vecNormal.begin();
A::iterator DataNormalEnd = (*iterData)->vecNormal.end();
for (DataNormal; DataNormal != DataNormalEnd; ++DataNormal)
{
    fwrite(&(*DataNormal), sizeof(FbxVector4), 1, pFile);
 }
 // UV information also stores size first.
fwrite(&(*iterData)->vecUV.size(), 4, 1, pFile);
B::iterator DataUV = (*iterData)->vecUV.begin();
B::iterator DataUV = (*iterData)->vecUV.end();
for (DataUV; DataUV != DataUVEnd; ++DataUV)
{
   fwrite(&(*DataUV), sizeof(FbxVector2), 1, pFile);
}
```

Figure 7. Part of Source Code for Storing the Mesh Normal and UV Information.

(A : vector<FbxVector4>, B : vector<FbxVector2>)

Part of Source Code for Reading the Number of Polygons and Mesh Data to be Stored in the Proposed Format

```
// Reads the number of stored meshes. Makes iterative statements of this size.
size_t iSize;
fread(iSize, 4, 1, pFile);
 // PFBXMESHDATA is a structure containing information to be stored.
PFBXMESHDATA temp = new PFBXMESHDATA;
for(size_t i = 0; i < iSize; ++i)
{
  // Read the number of polygons.
   fread(&temp->iPolygonCount, sizeof(UINT), 1, pFile);
  // the size of location information of mesh data
   size_t vecPosSize;
   fread(&vecPosSize, 4, 1, pFile);
   // It reads the location information using the loop as much as the size.
   for (size_t j = 0; j < vecPosSize; ++j)
   {
      FbxVector4 pos;
      fread(&pos, sizeof(FbxVector4), 1, pFile);
      temp->vecPos.push_back(pos);
   }
}
```

Figure 8. Part of Source Code for Reading the Number of Polygons and Mesh Data to be Stored in the Proposed Format.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(a) The Rendering Image by the FBX Format

(b) The Rendering Image by the Proposed Format

Figure 9. The Rendering Images.

Table 3. File Size Reduction of the Proposed Format compared to the conventional FBX.

Files	The Conventional FBX	The Proposed Format	The Reduction Ratio of File Size
File 1	4.50MB	1.01MB	-77.50%
File 2	1.63MB	390KB	-76.07%
File 3	1.34MB	28.4KB	-78.80%
File 4	23.4MB	1.54MB	-93.41%
File 5	9.45MB	2.57MB	-72.80%
File 6	12.8MB	1.10MB	-91.40%

4. PERFORMANCE EVALUATION

All experiments for comparison were made under the same conditions. The information on the PC used in the comparison experiment is the processor: Intel Core i7-6700HQ, CPU @ 2.60GHz, memory 16.00GB, 64bit operating system, graphics card: NVIDIA Geforce GTX 950M. Experimental classification compares the capacity of the FBX with its own format and the loading time. Loading time comparisons were broken down according to the number of output objects.

4.1 File Format

In order to derive the average value of the comparison results, six FBX files were converted into their own formats and the increase and decrease of capacity were compared. In addition, since the animation information is not stored in the self-format, the FBX file which does not have animation information is used in order to calculate the correct numerical value, and the experiment is carried out that all the information has Diffuse, Specular and Normal Texture information. The size shown in the experiment Table 2 is based on the information displayed in the size of the file attribute. It is a file that confirms that there is no abnormality in the rendering test after format conversion. In Table 3, experimental results show that the average capacity reduction effect is 81.66%. Considering that the number of objects in a game scene is several hundreds or more, a tremendous capacity reduction effect can be expected.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 10. 3D Model used in Loading Experiment.

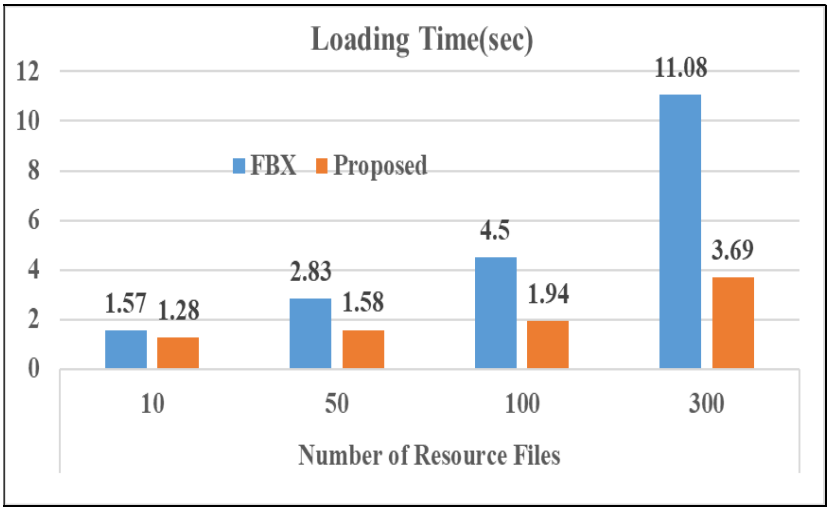


Figure 11. Loading Time Comparison According To Various Resource Files.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.2 File Format

The FBX SDK scene graph is abstracted by the FbxScene class. The scene is organized as a hierarchy of nodes (FbxNode). The root node of the scene is accessed via FbxScene::GetRootNode(). A scene element, for example, a mesh, a light, or a camera is defined by combining a FbxNode with a subclass of FbxNodeAttribute. For more information, see FBX Nodes and FBX Node Attributes [12].

Nodes are primarily used to specify the position, rotation and scale of scene elements within a scene. Nodes are abstracted by the FbxNode class. A FbxScene contains a parent-child hierarchy of nodes. The root node of this tree is accessed via FbxScene::GetRootNode(). As detailed in FBX Scenes, additional nodes can be created and added to this root node [12].

The node hierarchy is traversed using methods such as FbxNode::GetChild() and FbxNode::GetParent(). FbxNode::GetChildCount() returns the number of children of that node [12].

Nodes are organized in a hierarchy such that the position, rotation and scale of a node is described in relation to its parent's coordinate system. For example, in the diagram below, if the cubeNode is translated by 4 units along the rootNode's x-axis, the lightNode will also be affected by this translation. However, cameraNode will not be affected by this translation because cameraNode is not a child of cubeNode [12].

The order in which the rotation and scaling transforms are applied to a parent and its children is specified by the node's inherit type (ETransformInheritType). This transformation inheritance can be set using FbxNode::SetTransformationInheritType(). Consult the FbxNode class documentation for more details [12].

A FbxNodeAttribute is paired with a FbxNode to define a scene element with a specific position, rotation and scale. Calling FbxNode::GetNodeAttribute() will return NULL if no node attribute was set for that node [12].

Preservation of 3D data involves basic understanding of 3D data characteristics, 3D file formats and viewing software [10]. One of our objectives is to understand the information loss introduced by 3D file format conversions with many of the software packages designed for viewing and converting 3D data files [10]. In order to quantify the information loss, a possible approach is to rank the characteristics of 3D data sets and design metrics for scoring 3D file conversions. This approach would depend on the application defining why 3D models would be preserved [10]. For example, if the 3D model contained a 3D simulation of a crime, then the scene information would be ranked higher than the appearance and geometry of the individual objects. On the other side, if the 3D model was being preserved for the future users of the model in order to replace a part of the object being modeled, then the ranking of 3D data characteristics would follow the order of geometry, appearance and scene [10]. It is also conceivable to build 3D models of wild fire where the appearance of flames would have higher preservation priority than the geometry and scene since the appearance conveys the information about what burned [10].

In this experiment, the load times are compared. Experimental results show that 10, 50, 100, and 300 objects are loaded and compared. Objects are individual objects that do not use instancing and are single-threaded. Each experiment was performed three times to calculate the correct value and the average time was calculated and compared. The compilation mode used in the experiment is Release. Figure 10, the number of vertices in the 3D model used is 128, and the number of polygons is 106.

As shown in Figure 11, the larger the number of objects used, the greater the time difference. When comparing the time of loading 10 objects, the difference was about 0.29 seconds, but when loading time of 300 objects was compared, there was a difference of about 7.38 seconds. The larger the number of objects to be loaded, the bigger the difference was. Unlike the FBX file, which requires a lot of information to find the required value, it only has the necessary information, and if you just read the information in order, it is a result of the structure of the own format. Reduced resource loading time is an average of 51%.

5. CONCLUSION

FBX files have a lot of information. However, it also has information that is not used in games, which is inefficient in terms of file size and load time. If they analyze the information they need in their own game, extract the necessary information based on it, and create their own file, then it is efficient in terms of file size and load

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

time. This was demonstrated by the above experiment. If you have an FBX file with animation, they will need more information such as this information, animation time, etc., so the file size and load time will be bigger. In addition, if instancing, multi-threading techniques that can be implemented in DirectX are used, the proposed file format can expect a dramatic speedup.

ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. 2016-0-00204, Development of mobile GPU hardware for photo-realistic real time virtual reality).

REFERENCES:

[1] Newzoo's global game markets report, https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/

[2] David H. Eberly, "3D game engine design: a practical approach to real-time computer graphics", CRC Press, 2006.

[3] Seung Seok Noh, Sung Dea Hong, and Jin Wan Park, "Using a game engine technique to produce 3D Entertainment contents", *Artificial Reality and Telexistence--Workshops, 2006. ICAT'06. 16th International Conference on*. IEEE, 2006.

[4] Hsu, Chia-chun Alex, et al., "The design of multiplayer online video game systems", *Multimedia Systems and Applications VI*. Vol. 5241. International Society for Optics and Photonics, 2003.

[5] R.Darken, P.McDowell, E.Johnson, "Projects in VR: the Delta3D open source game engine", *IEEE Computer Graphics and Applications*, Vol 25, pp.10-12, 2005.

[6] Mingshao Zhang, et al., "Recent Developments in Game-Based Virtual Reality Educational Laboratories Using the Microsoft Kinect", *International Journal of Emerging Technologies in Learning (iJET)*, Vol. 13, No.1, pp.138-159, 2018.

[7] Sukkyung You, Euikyung Kim, and Donguk Lee, "Virtually real: exploring avatar identification in game addiction among massively multiplayer online role-playing games (MMORPG) players", *Games and Culture,* Vol.12, No.1, pp.56-71, 2017.

[8] Jason Zink, Practical Rendering & Computation with Direct3D 11, *WOW Books*, pp. 595 – 599, 2013.

[9] Dongryul Lee, Youngsik Kim, "A Shadow Mapping Technique Separating Static and Dynamic Objects in Games using Multiple Render Targets", *Journal of The Korean Society for Computer Game*, Vol.28, No.4, pp. 1-10, 2015.

[10] Kenton McHenry and Peter Bajcsy, "An overview of 3d data content, file formats and viewers", Technical Report: isda08-002, National Center for Supercomputing Applications, 1205:22, 2008.

[11] FBX file format, https://en.wikipedia.org/wiki/FBX

[12] AutoDesk FBX Document FBXObject, http://docs.autodesk.com/FBX/2014/ENU/FBX-SDK-Documentation/index.html?url=files/GUID-5D8BFE45-723E-4BD9-846B-E2B2540157C9.htm,topicNumber=d30e6836

[13] P. Bourke. Data Formats, URL: http://local.wasp.uwa.edu.au/~pbourke/dataformats

[14] Universal 3D File Format, ECMA 363, 2006.

[15] Extensible 3D (X3D) Specification, ISO/IEC 19776-1, 2006.