Journal of Theoretical and Applied Information Technology
15th July 2021. Vol.99. No 13
© 2021 Little Lion Scientific

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

DESIGN AND EVALUATION OF A MOBILE 3D ARCADE GAME WITH MESH CLIPPING

YOUNGSIK KIM

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

E-mail: kys@kpu.ac.kr

ABSTRACT

To cut a polygon of the 3D mesh object from a computer, it is necessary to create a plane using the plane equation, calculate the distance between the point and the plane, and calculate the intersection point and the plane. In this paper, we propose a 5-step mesh clipping method to cut polygons in a 3D game. The proposed method consists of (1) plane generation, (2) point separation according to the distance between vertex and plane of polygon, (3) intersection of plane and polygon, (4) separation of the triangle using intersection, and (5) creation of cross-section. The proposed method is applied to the mobile 3D arcade games. Experiments are conducted to measure the variation of the frame rate (Frame Per Second (FPS)) according to the type of objects displayed during the game. Moreover, computer experiments have verified that the mesh clipping effect is natural and accurate. It can be confirmed that noticeable frame deterioration does not occur when the cube is cut. On the other hand, if they have a monster with a more significant number of triangles and vertices, they may notice a slight frame drop. Also, we confirmed that the effect of the degradation of rendering speed on game progress is insignificant when applying the proposed mesh clipping to 3D games.

Keywords: *Mobile 3D Arcade Game, Mesh Clipping, Computer Simulation, Rendering Speed, Unity3D*

1. INTRODUCTION

As 3D modeling and simulation technology and computer graphics technology have been advanced in recent years, game technologies based on these technologies have been applied in various industrial fields. Interactive multimedia content, sensory entertainment systems, virtual worlds, 3D training systems, and the like. The shape of things on a computer is polygon.

The mesh formation and processing technique that expresses the shape of objects is essential as the underlying technology.

Recently, in the Virtual Reality (VR) game market, it is easy to find a game against an opponent using a controller as a sword [1]. Among them, games such as Fruit Ninja VR [2] in Figure 1 (a) and Beat Saber [3] in Figure 1 (b) are genres that can directly cut objects and enjoy hitting feeling. The fact that smartphone games, such as Fruit Ninja VR [2] and Bet Saber [3], have been released to the market and similar games continue to be released suggest that people like this intuitive interactive game approach.

The mesh cutting technique of the 3D model makes it possible to observe the inside of the shape closely in real-time so that it can be widely used in various fields.

(a) Fruit Ninja VR [2]

(b) Beat Saber [3]

Figure 1. 3D Games with Object Cutting.

For example, there have been cases of applications in the fields of medical research, mechanical design, geological exploration and architectural design, including virtual surgery [4,5].

The 3D game to be designed and verified in this paper is also similar to the two games in Figure 1. It is a game that follows the road and cuts the objects coming out, and enjoys a sense of hitting. In this paper, we implement mesh clipping that divides object, which is the most prominent feature of such game, and keeps each part of the divided objects. Moreover, mesh clipping is applied to the mobile 3D Arcade game and verified through experiments. For the mesh to be cut according to the shape input by the user on the game screen, creating a virtual plane is required. Create a plane using the previous position and the current position of the straight line the player entered. After the plane is created, it is determined whether each point of the mesh is before or after the plane based on this plane, separated and stored. Then, the intersection of the plane and the mesh is obtained to create the section. It is generated by dividing each mesh through the points divided by the plane reference and the created cross section.

Related work [7] introduces clipping and explains the principle and meaning of plane equations. We are also approaching how to clone a mesh. In the related work [9], clipping is approached more theoretically, and briefly summarizing what kind of clipping is done, and then explaining it in detail is explained. The clipping problem is usually solved without considering the differences between Euclidean space and the space represented by homogeneous coordinates. For some constructions, this leads to errors in picture generation which show up as lines marked invisible when they should be visible. Previous research [11] examined these cases and presented techniques for correctly clipping the line segments.

In [12], a new family of clipping algorithms was described. These algorithms can clip polygons against irregular convex plane-faced volumes in three dimensions, removing the parts of the polygon that lie outside the volume. In two dimensions, the algorithms permit clipping against irregular convex windows.

Barycentric coordinates are used to define all vertices: original, intermediate, and final intersection points. The use of barycentric coordinates results in less storage space. A circular buffer is used during the clipping process to store input and output polygons. The use of the circular buffer also results in reduced storage requirements [13].

The Unity3D game engine is an integrated authoring tool for creating other interactive content such as 3D video games, architectural visualizations, and real-time 3D animations [14]. In particular, the Unity3D game engine provides a hierarchical integrated development environment and supports 3D model source files generated from various 3D modeling software such as 3ds Max, Maya, Blender, and Cinema 4D. The Unity3D game engine has been implemented by Noble Muffins [15].

The scene manager applied to [15] is BSP-Tree (Binary Space Partitioning Tree) [16,17]. BSP-Tree is a kind of Binary Tree that contains information for partitioning space, initially designed for Hidden Surface Removal. The application range has been expanded widely due to its characteristics. In computer science, binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyperplanes. This subdivision gives rise to a representation of objects within the space employing a tree data structure known as a BSP tree [17].

Binary space partitioning was developed in the context of 3D computer graphics, where the structure of a BSP tree allows spatial information about the objects in a scene that is useful in rendering, such as their ordering from front-to-back concerning a viewer at a given location, to be accessed rapidly. Other applications include performing geometrical operations with shapes (constructive solid geometry) in CAD, collision detection in robotics and 3D video games, ray tracing and other computer applications involving handling of complex spatial scenes [17].

Binary space partitioning is a generic process of recursively dividing a scene into two until the partitioning satisfies one or more requirements. It can be seen as a generalization of other spatial tree structures such as k-d trees and quad-trees, one where hyperplanes that partition space may have any orientation, rather than being aligned with the coordinate axes as they are in k-d trees or quad-trees. When used in computer graphics to render scenes composed of planar polygons, the partitioning planes are frequently chosen to coincide with the planes defined by polygons in the scene [17].

The binary space division method recursively divides a space into two spaces until a specific final purpose is satisfied. For example,

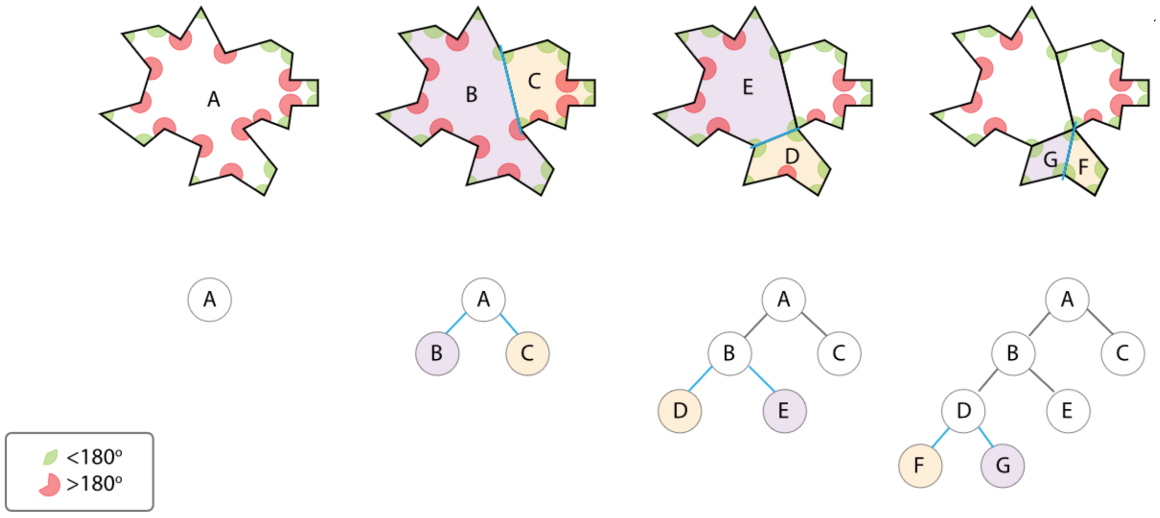


Figure 2. Generating BSP Tree [17].

in the case of collision detection, the space is divided so that the original object can be sufficiently collision-checked. In the case of rendering, the space is divided into convex shapes so that the algorithm can be used most efficiently. Do [17].

Since the plane across the splitting line must be divided into two, the number of final objects necessarily increases. Because the BSP tree must be well-balanced, creating the right and efficient BSP tree is the most difficult part of the implementation process. In 3D space, a plane is used to divide the plane of an object [17].

The Figure 2 illustrates the process of dividing an irregular polygon into several convex polygons. Note that the number of edges of the polygons divided for each step from the beginning to G and F decreases, and at the final step, the polygon becomes a completely convex polygon. In some cases, the dividing line may extend between vertices in space and non-intersecting line segments. If the splitting line intersects a line segment or plane, the intersected segment or plane is divided into two so that it is completely independent [17].

In Figure 2, A is the root of the BSP Tree which includes all polygons in the world. A is divided into B and C. B is divided into D and E. Then, D is divided into convex polygons F and G which are tree leaves. An efficient BSP structure is created entirely through a good algorithm. Most BSP algorithms test several cases to obtain the optimal tree in the partitioning process. Thus, the

It is relatively easy to judge whether a part of each triangular mesh to be cut is the front side or the back side of the cutting plane, or if the plane passes through the triangular mesh, when cutting a specific polygon based on an infinite plane in the BSP-Tree have. If the cutting plane penetrates the triangular mesh, divide the triangular mesh into two polygons to complete the cutting operation. One on the front side of the cutting plane and one on the back side. Generally, when cutting a triangle, one triangle and one quadrilateral are created. The newly created triangles and quadrangles are stored in the buffer area. The triangular or tetrahedral meshes, which are stored in the buffer region, are then transformed into two separate 3D models by mesh splitting [18].

In the conventional cutting method using BSP-Tree, since the 3D model is cut into two parts based on the infinite plane, it is difficult to use it for general cutting work. [19] proposed a technique of cutting a 3D model within a finite region to solve this problem. Specifically, the cutting path plane can be defined finitely, and the 3D model is divided only within the cutting range, so that it can be usefully used in various industrial fields. [18] applied the disassembly process of the reactor 3D model to the virtual simulation process to show the usefulness of the proposed partial cut technique.

In this paper, the mesh clipping algorithm was implemented in a mobile 3D arcade game developed by ourselves. In addition, the efficiency of the process of processing complex meshes was verified through operation experiments of 3D arcade games. The metrics used for verification are rendering speed, Frame per Second (FPS).

2. BACKGROUNDS

2.1 Plane Equation

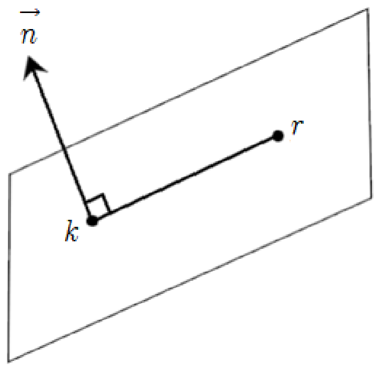


Figure 3. Plane determined by Point and Normal Vector.

As shown in Figure 3, the plane is determined by a point k and a vector $\vec{n}$ that is perpendicular to the plane to be determined. (The vector $\vec{n}$ at this time is the normal vector of the plane.) When $r-k$ is p for any point r on the plane, $\vec{n} \bullet \vec{p} = 0$ holds. Equation 1 holds for vector $\vec{n} = (a,b,c)$, point $r = (x,y,z)$, and $k = (x_0, y_0, z_0)$.

$$\begin{aligned}(a,b,c)\bullet(x-x_0,y-y_0,z-z_0)=0\\ a(x-x_0)+b(y-y_0)+c(z-z_0)=0\end{aligned} \tag{1}$$

If this is summarized and the constant term is d, Equation 2 is established. This is called the general equation of the plane [6, 7].

$$ax+by+cz+d=0 \tag{2}$$

2.2 Geometrical Relationship between Plane and Point

The distance D between the plane and the point is equal to Equation 3 [6].

$$D = |D'| = \frac{|ax_1+by_1+cz_1+d|}{\sqrt{a^2+b^2+c^2}} \tag{3}$$

If a space in the normal vector direction of the plane is referred to as 'front' and a space in the opposite direction of the normal vector is referred to as 'back', the position of the point p according to D' as shown in Table 1. If D' is greater than 0, the point p is in front of the plane and less than 0, the point p is behind the plane. If D' is 0, the point p is the point included in the plane. When mesh clipping is performed, the points are collected and stored according to the position of the point on the plane, and then the mesh is divided.

Table 1. Position of Point p by Condition D'

	Position of Point *p*
D' > 0	Point *p* is in front of the Plane
D' = 0	Point *p* is in the Plane
D' < 0	Point *p* is in back of the Plane

2.3 Intersection of Plane and Line

As shown in Figure 4, the general way to find the intersection when a straight line passing through two points p_1 and p_2 and a plane meet at a point is as follows.

If $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$, the straight line intersecting the plane and the point p can be expressed as Equation 4.

$$p = p_1 + u(p_2 - p_1) \tag{4}$$

Where u is the slope of the line. Equation 4 is assigned to plane equation $ax+by+cx+d=0$ and u is summarized as Equation 5.

$$u = \frac{ax_1+by_1+cz_1+d}{a(x_1-x_2)+b(y_1-y_2)+c(z_1-z_2)} \tag{5}$$

Substituting u in Equation 5 into Equation 4 yields the intersection point [8].

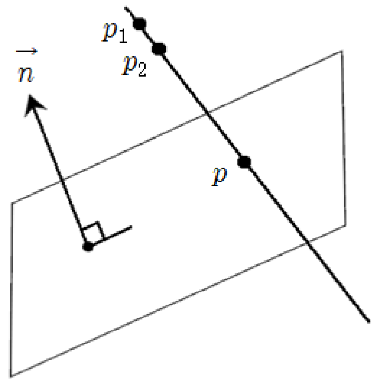


Figure 4. Plane determined by Point and Normal Vector.

3. Mobile 3D Arcade Game Design

As described above, we implemented and experimented with Mobile 3D Arcade game with mesh clipping. The game of this paper was produced using Unity 3D game engine (version: 2018.2.0f2, 64-bit) [14]. Figure 5 shows the flow of the game in this paper to implement mesh clipping and execute it.

When you start the game, the first-person player moves along the spline curve. As the game progresses, objects appear at regular intervals, and the player can ring the object. When the object approaches a certain distance from the player, the color changes, and the player can ring the object. When the player plays the object, the mesh clipping is done and the mesh is cut and the score is obtained. When the player can not cut the object, it bumps all the life, or when the player reaches the end of the curve, the score is printed and the game ends.

As shown in Figure 6, there are a total of eight stages in which the game can be played, and scores can be recorded for each stage to compare this record with others.

4. MESH CLIPPING DESIGN IN THE 3D GAME

4.1 Mesh Clipping Control Flow

Figure 7 shows the flow of mesh clipping applied in the game. The user creates a plane through a straight line entered to cut the object. Mesh clipping proceeds to the generated plane.

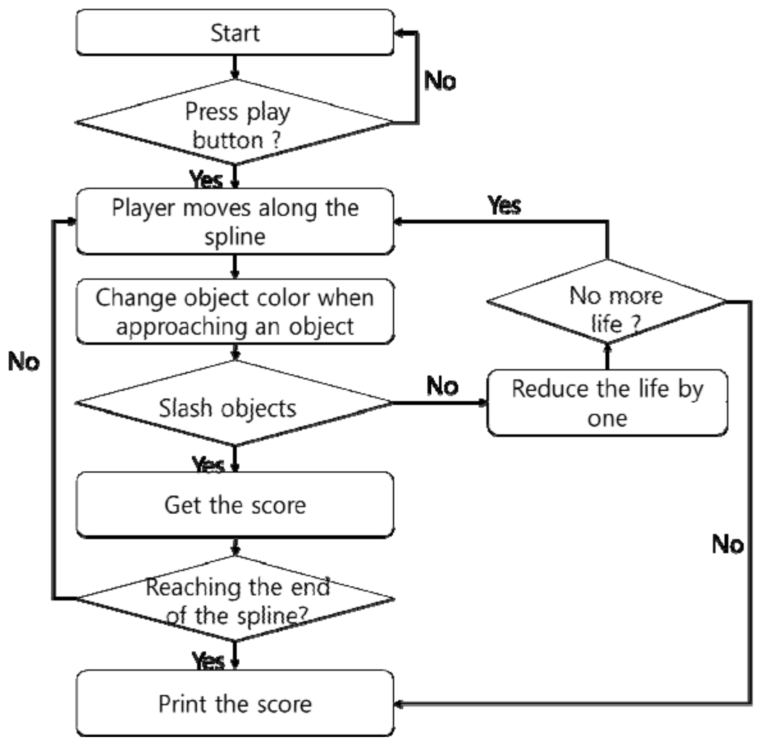
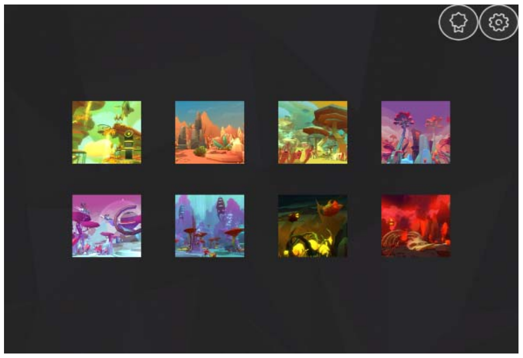
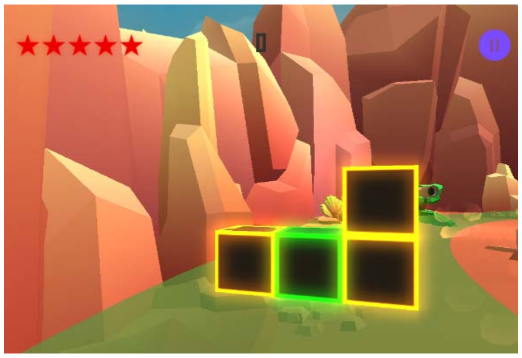


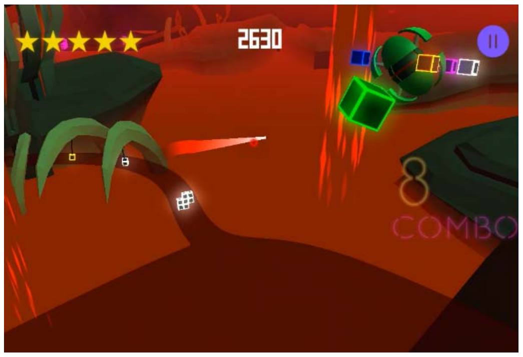
Figure 5. The Control Flow of the Mobile 3D Arcade Game.



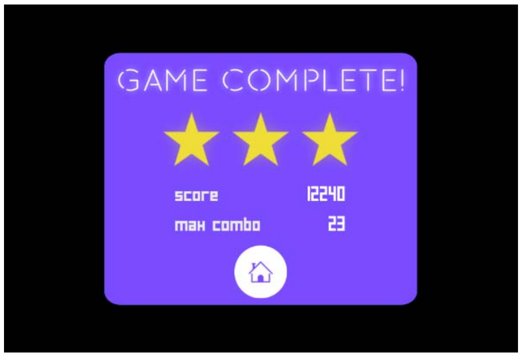
(a) Game Lobby



(b) Cube in the Game



(c) Monster in the Game



(d) Game Score

Figure 6. The Screen Shots of the Mobile 3D Arcade.

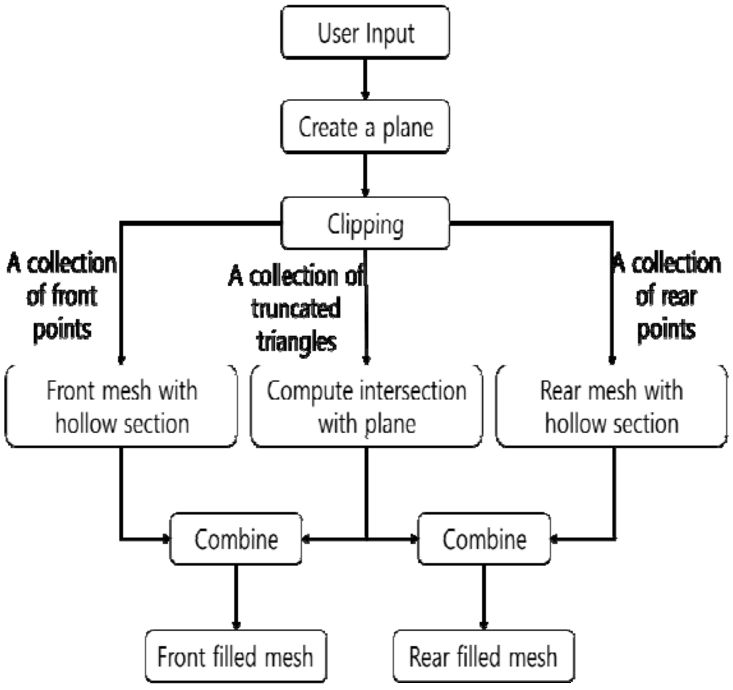

User Input
Create a plane
Clipping
A collection of front points
A collection of truncated triangles
A collection of rear points
Front mesh with hollow section
Compute intersection with plane
Rear mesh with hollow section
Combine
Combine
Front filled mesh
Rear filled mesh

Figure 7. Mesh Clipping Control Flow.

Mesh is divided into three cases as shown above. First, it can be divided into a group of points preceding the plane and a group of points behind the plane. These points are used to create a front mesh and a back mesh that are not filled in each section.

In the triangle cut by the plane, the intersection with the plane is searched to create the section, and it is possible to create the mesh filled with the front mesh and the rear mesh created above. The above procedure will be used to perform mesh clipping and explain how the above process is done.

4.2 The Proposed Mesh Clipping

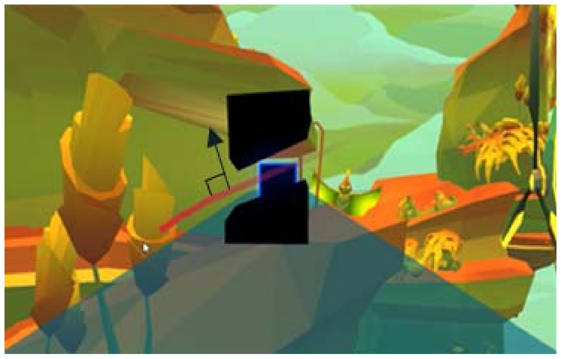


Figure 8. Normal Vector for Making Plane.

The proposed method consists of (1) plane generation, (2) point separation according to the distance between vertex and plane of polygon, (3) intersection of plane and polygon, (4) separation of triangle using intersection, and (5) creation of cross section.

To cut a mesh, we first need to create a plane to truncate the mesh. As described in Section 2.1, a point and a normal vector are required to create a plane (see Figure 8). After the straight line you entered touches the object you want to clip, you get two points and get the center point. This is the point that is necessary for generating the plane. Also, the two imported points are used as the normal vectors necessary to generate the plane by subtracting one point from one point out of two points rotated 90 degrees from the center point.

If you have created a plane, you must locate the point based on the plane and separate it. Use Equation 3 in section 2.2 to divide the position of the point. The position of the point is divided into three cases, but if it is not before the plane, it is judged to be behind the plane and proceed. That is, if the point is included in the plane, it is assumed to be behind the plane.

It repeats the number of triangles of the object to be cut and finds the three points of the triangle in the space. If three points of a triangle are in the same space, the triangle is not divided by the plane created by the straight line you entered. The points of the unpartitioned triangle are stored according to the space. Conversely, if one of the three points is in another space, the triangle is divided. In this case, triangles must be separated by intersection [7,9,11]. Figure 9 illustrates the vertex separation pseudo-algorithm.

Pseudo Program: Vertex Separation

```
foreach Triangle in Mesh
    foreach Vertex in Triangle
        D = Distance between Vertex and Plane
        (not taking the absolute value)

    if D > 0
        Side = Front
    else
        Side = Back

    if All on the Same Side
        if Side == Front
            Add Vertices in Front Vertex Array
        else
            Add Vertices in Back Vertex Array
    else
        Clipping the Triangle
```

Figure 9. The Pseudo Algorithm of Vertex Separation by Plane.

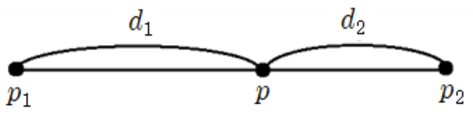
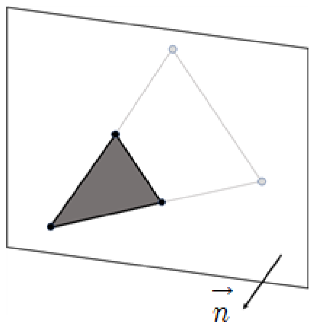


Figure 10. Linear Interpolation.

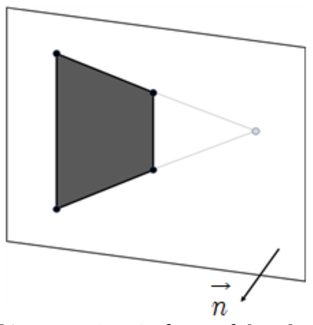
First, store the points of the triangle to be separated according to the space and obtain the intersection point. The intersection point is generally obtained through the procedure in Section 3.3, but it can be easily obtained by using the fact that the intersection of the triangle and the plane is always between two points in another space of the triangle. The linear interpolation method shown in Figure 10 is used.

Equation 6 is established if the intersection is p, the distance between point p_1 and point p is d_1, and the distance from point p_2 to point p is d_2.

$$p = \frac{d_2}{d_1 + d_2} p_1 + \frac{d_1}{d_1 + d_2} p_2 \qquad (6)$$



(a) A single point in front of the plane



(b) Two points in front of the plane

Figure 11. Triangles Clipped by Plane.

We can use Equation 6 to find the intersection because we know the distance from the point in front of the plane to the plane and the distance from the point behind the plane to the plane for two points in another space of the triangle.

When the triangle is divided by plane, there are two cases as shown in Figure 11 (a) and (b). In Figure 11 (a), one point of the triangle is placed in front of the plane, and two points of the triangle are located behind the plane. Conversely, in Figure 11 (b), two points of the triangle are placed before the plane, and one point of the triangle is located behind the plane.

Among the divided surfaces, the square surface is divided into two triangles.

Pseudo Algorithm: Clipping the Triangle

```
if Number of Vertex on Front Side = 1
    F1 ⟵Vertex on Front Side
    B1,2 ⟵Vertex on Back Side
    N1,2 ⟵Intersection of Plane and Triangle
    Add N1, N2 in New Vertex Array

    if Direction of F1, N1 and N2 is not CCW
        Change Direction of Vertices
    Add Vertices in Front Vertex Array

    if Direction of B1, N1 and N2 is not CCW
        Change Direction of Vertices
    Add Vertices in Back Vertex Array

    if Direction of B1, B2 and N2 is not CCW
        Change Direction of Vertices
    Add Vertices in Back Vertex Array
else
    F1,2 ⟵Vertex on Front Side
    B1 ⟵Vertex on Back Side
    N1,2 ⟵Intersection of Plane and Triangle
    Add N1, N2 in New Vertex Array

    if Direction of F1, N1 and N2 is not CCW
        Change Direction of Vertices
    Add Vertices in Front Vertex Array

    if Direction of F1, F2 and N2 is not CCW
        Change Direction of Vertices
    Add Vertices in Front Vertex Array

    if Direction of B1, N1 and N2 is not CCW
        Change Direction of Vertices
    Add Vertices in Back Vertex Array
```

Figure 12. The Pseudo Algorithm of Clipping the Triangle.

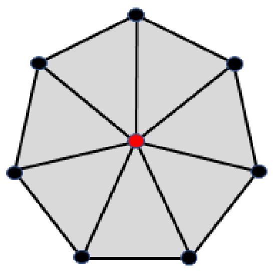


Figure 13. Section Created by Intersections.

Note the order when storing new triangles created with intersections. The mesh is visible only in the counterclockwise direction based on the normal vector of the triangle. If the order of the three points is not counterclockwise, the order of the points must be reversed to make them counterclockwise [9]. Figure 12 shows the pseudo-algorithm of triangulation.

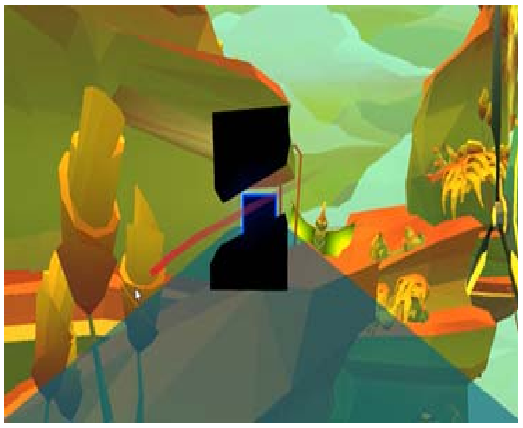
Through the above process, the mesh is divided by using the separated points. Before creating the mesh, we must process the cut section.

We can create a triangle with a set of intersections found above (see Figure 13).

As shown in Figure 13, you can create a cross-section by creating a triangle after finding the midpoints of the intersections. The midpoint is simply obtained by dividing the sum of the intersections by the number of intersections. We create the frontal mesh by combining the created triangles with the set of front triangles obtained in the previous procedure. Likewise, we create the back mesh by combining the created triangles with the back triangles we obtained in the previous procedure.

5. PERFORMANCE EVALUATION

We applied the mesh clipping proposed in Section 4 to the mobile 3D arcade game designed in Section 3 and measured the rendering speed when the object was cut. The specifications of the computer used in the experiment are as follows: Processor: Intel (R) Core (TM) i5-5200U CPU @


(a) Slashing Cube

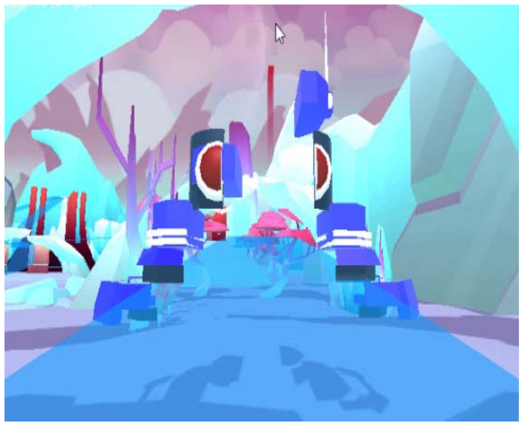

(b) Slashing Monster

Figure 14. Screen Shots for Slashing Objects.

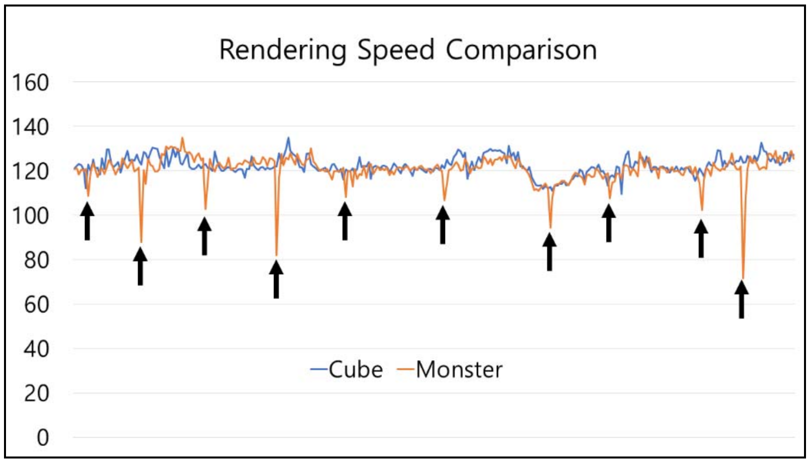


Figure 15. Rendering Speed Comparison.

2.20GHz 2.20GHz, memory: 8.00GB, graphics card: GeForce 940M and 64bit operating system.

Figures 14 (a) and (b) show screen shots of slashing Cube and Monster as the game progresses.

Table 2 describes the number of vertices and triangles of cube and monster objects used in the game.

Table 2. Number of Vertices and Triangles of Each Type of Object.

	Cube	Monster
Vertices	24	1453
Triangles	12	816

Experiments were conducted to measure the variation of the frame rate (Frame Per Second (FPS)) according to the type of objects displayed during the game. Figure 15 shows the results of measuring the change in rendering speed until the cube and monster are tilted 10 times while playing

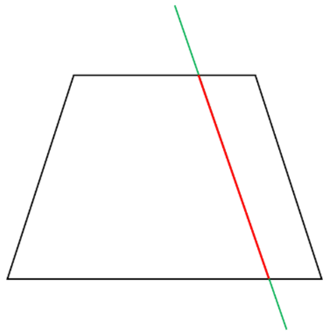
the game. In Figure 15, the black arrow points to the rendering speed when the objects are slashed.

It can be confirmed that noticeable frame deterioration does not occur when the cube is cut. On the other hand, if they have a monster with a larger number of triangles and vertices, they may notice a slight frame drop.

Table 3 shows the average rendering speed with and without mesh clipping for cube and monster. In the case of a cube, there was almost no frame change, and in the case of a complex mesh monster, the rendering performance dropped by about 19.4%. Since the mesh clipping process proceeds through triangles and vertices, rendering performance degrades as the number of triangles and vertices increases. However, since the rendering speed is very high, the effect on mobile 3D game progress is negligible.

Table 3. Average Rendering Speed with and without Mesh Clipping.

	Cube	Monster
Without Mesh Clipping	122.51FPS	120.81FPS
With Mesh Clipping	118.27FPS	97.27FPS


(a) Convex Polygon

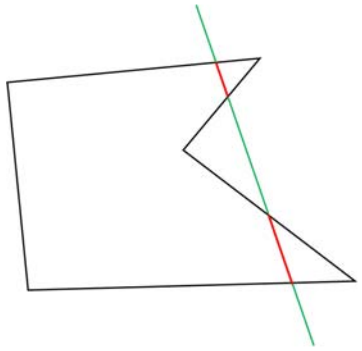

(b) Concave Polygon

Figure 16. Convex and Concave Polygons.

6. CONCLUSION

In this paper, we study mesh clipping by applying plane equations and experiment on 3D game. Experimental results show that the user can create a plane using the line and then cut the mesh into the plane. The result shows that the performance degradation is more complicated for mesh with many vertices and triangles.

The study and experiment in this paper is about the convex polyhedron described in Figure 16 (a). The concave polygon described in Figure 16 (b) is a polygon with at least one internal shape of the figure, and the concave polygon with a face is a concave polygon. In the case of concave polyhedra, there are many other variables to consider.

For example, in the case of a concave polyhedron, there is no guarantee that one cross section will occur in the process of creating a cross section. Future research will be to increase efficiency in processing complex meshes and to study more accurate mesh clipping to handle concave meshes and apply them to game.

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] Robert Gruen, et al., "Measuring system visual latency through cognitive latency on video see-through AR devices", *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, IEEE, 2020.

[2] Ysabelle Coutu, et al., "Immersiveness and Usability in VR: AComparative Study of Monstrum and Fruit Ninja", *Game User Experience and Player-Centered Design*, Springer, Cham, 2020, pp. 437-448.

[3] Ancret Szpak, Stefan Carlo Michalski, and Tobias Loetscher, "Exergaming With Beat Saber: An Investigation of Virtual Reality Aftereffects", *Journal of Medical Internet Research,* 22.10:e19840, 2020.

[4] Sung-Ho Kim, "Development of Simulator for Rockfall and Landslide using Physical Engine", *The Journal of the Korea Contents Association*, Vol. 9, No. 9, pp. 60-67, 2009.

[5] B. Kapralos, C. Johnston, K. Finney, and A. Dubrowski, “A Serious Game for Training Health Care Providers in Interprofessional Care of Critically-Ill and Chronic Care Patients”, *Journal of Emerging Technologies in Web Intelligence*, Vol. 3, No. 4, pp. 273-281, 2011.

[6] Wikipedia, "Plane (geometry)", https://en.wikipedia.org/wiki/Plane_(geometry).

[7] David Eberly, "Clipping a Mesh Against a Plane", *Geometric Tools,* 2008.

[8] Paul Bourke, "Intersection of a plane and a line", University of Western Australia, http://paulbourke.net/geometry/pointlineplane/, August, 1991.

[9] Aaron Scherzinger, Tobias Brix and Klaus H. Hinriches, "An Efficient Geometric Algorithm for Clipping and Capping Solid Triangle Meshes", *In Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pp. 187-194, 2017.

[10] Wikipedia, "Linear interpolation", https://en.wikipedia.org/wiki/Linear_interpolation.

[11] James F. Blinn and Martin E. Newell. "Clipping using homogeneous coordinates." *ACM SIGGRAPH Computer Graphics*, Vol. 12, No. 3, ACM, pp.245-251, 1978.

[12] Ivan E. Sutherland and Gary W. Hodgman, "Reentrant polygon clipping." *Communications of the ACM*, Vol.17, No.1, pp.32-42, 1974.

[13] David Robert Baldwin, "Triangle clipping for 3D graphics", U.S. Patent No. 7,215,344, May 8, 2007.

[14] Unity3D Game Engine, http://unity3d.com/

[15] Turbo Slicer Guide, http://www.noblemuffins.com/files/turboSlicerGuide.pdf

[16] Xuhui Fan, Bin Li, and Scott Sisson, "The binary space partitioning-tree process." International Conference on Artificial Intelligence and Statistics. PMLR, 2018.

[17] Binary Space Partitioning Tree, https://en.wikipedia.org/wiki/Binary_space_partitioning

[18] Viet, H.Q.H., T. Kamada, and H.T. Tanaka, “An Algorithm for Cutting 3D Surface Meshes”, *Proceedings of the 18th International Conference on Pattern Recognition* (ICPR'06), Vol. 4, pp. 762-765, 2006.

[19] Wan-Bok Lee, Wen-Yuan Hao, Byung-Pyo Kyung, and Seuc-Ho Ryu, "Dismantling Simulation of Nuclear Reactor Using Partial Mesh Cutting Method for 3D Model", *Journal of Digital Convergence*, Vol. 13, No. 4, pp. 303-310, 2015.