Presented at the DLSU Research Congress 2016
De La Salle University, Manila, Philippines
March 7-9, 2016

Gesture-Based 3D Mesh Modeler

Roland Carlos[1], Clarence Dalan[1], Aaron Sanchez[1], Kevin Tolentino[1] and Florante R. Salvador[1, *]

[1]De La Salle University, Manila

[*]Corresponding Author: florante.salvador@dlsu.edu.ph

Abstract: This research is concerned with the design and implementation of a 3D modeling tool that utilizes gesture-based input. The idea is to create a system for tablet PCs, specifically an Android tablet, that accepts touch input for the creation and modeling transformation of a 3D mesh model. The current prototype supports creation, deletion and editing of 3D primitives, i.e., vertices, edges, faces and meshes. The implementation difficulties encountered and that had to be addressed are due to computing power, memory space and size of the display screen.

Keywords: Mesh modeling; 3D modeling; sketch-based interface; computer graphics

1. INTRODUCTION

3D modeling is concerned with the representation, manipulation and storage of a scene composed of entities which include objects which have geometric (such as shape) and non-geometric properties (such as material properties, eg. color and texture). 3D modeling can be in technologies such as Computer Aided Design (CAD), and 3D computer animation software.

The demand for easier interfaces for 3D modeling has urged the development of various techniques/interfaces to create an intuitive environment for 3D modeling. Sketch-based interfaces or interfaces that accept hand or pen gestures as a way of input is more intuitive for creating geometric models (Igarashi, Matsuoka, and Tanaka, 1999; Yang, Sharon and van de Panne, 2005; Kin, et. al, 2011). This is based on the premise that drawing is a basic human skill. Humans convey their ideas visually by sketching or drawing figures.

Our research is concerned with the design and implementation of a 3D modeling tool that utilizes gesture-based input. The idea is to create a system for tablet PCs, specifically an Android tablet that accepts touch input for the creation and modeling transformation of a 3D mesh model. A 3D mesh model is composed of a set of vertices, edges connecting two vertices, edge loops called faces, and faces that makes up a mesh.

In this paper we describe a 3D modeling system that accepts gesture-based input for basic 3D modeling operations. The system runs on the Android platform with following functionalities: creation, deletion and editing of geometric primitives, modeling transformations (i.e., translation, scaling and rotation) and viewing transformations. It is also able to import existing 3D models and save created and edited 3D models into a file.

2. SYSTEM DESCRIPTION

2.1 System Overview

Figure 1 illustrates the flow of a gesture-based 3D mesh modeler that we developed. It accepts touch/hand gestures as input and maps them to a library of predefined gestures to determine the operation that is being requested.

There are three categories of operations: (1) geometric operations cover creation, deletion and modeling transformations, (2) viewing operation cover viewing transformation to specify how the mesh model will be viewed from a virtual camera, and (3) file management which handles the saving of

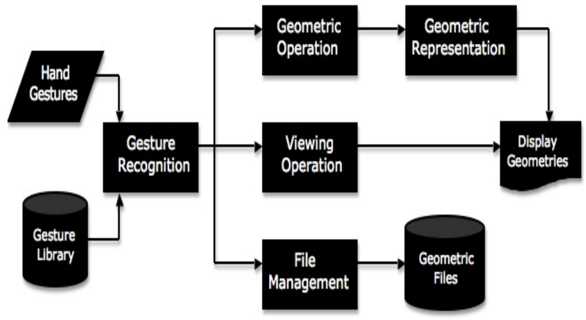

Fig. 1. Design of a gesture-based 3D mesh modeler

3D models into a file, and loading 3D models from existing files. The system updates the screen depending on the gestures and operations to be performed.

2.2 Some Implementation Details

The system is basically a simple mesh modeling tool created for the Android platform specifically for Android tablets. It accepts hand gestures, mainly by the tip of fingers or pointers, and interprets these gestures based on the mode of the system that is active. The system has four separate modes: create, model, scale, and camera. The actual user interface of the system can be separated into two layers; the first layer being the default layer provided by the Android API (Android Software Development Kit, 2015) when the activity was created and the second layer is the layer that was created using the OpenGL SurfaceView. The second layer, which is in charge of the actual rendering of the geometries into the screen, is mainly written using the OpenGL ES 1.0 API.

Usual applications (such as games) on touch screen devices are defined on a planar space. These touch screen gestures map one-to-one with 2D space representation. The challenge for our application is how to map gestures on a planar screen that create and manipulate geometric primitives defined in 3D space (i.e., with x, y, z coordinates).

We designed and tested different gestures for user-interaction, and finally decided to adopt a multi-touch approach as described above. Multi-touch requires the use of one, two or three fingers depending on the desired action (Jiao, Deng, and Wang, 2010).

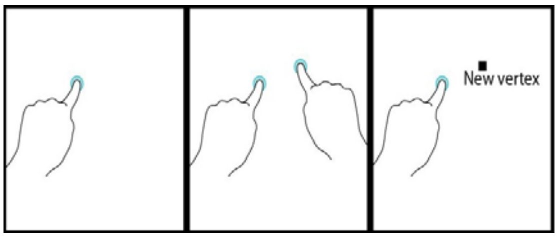

Fig. 2. Gesture for creating a vertex

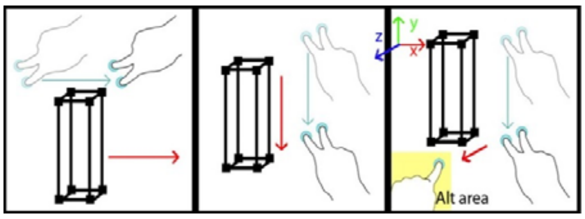

Fig. 3. Gesture for translating a model

When the system is in create mode, the only gestures that can be recognized are the *tap*, which is for selection of vertices, and the *hold-then-tap gestures*, which are for creation of geometries. Figure 2 shows the idea on how a vertex is created – by a single finger tap on the display screen. The implementation of the actual recognition of these two gestures does not make use of any library from the Android API due to the fact that the gesture library of Android does not return the raw values of the pointer when the gesture is made.

When the system is in model mode and in scale mode, it will still accept the single tap for vertex selection along with other gestures which include two fingers for translation, three fingers for rotation, and a combination of a finger on the alt region with the previously mentioned gestures for modeling transformations about the z axis since the normal gestures for translation and rotation are limited to the x and y axes. Figure 3 shows how a model is translated by moving two fingers along a specified direction, and Fig. 4 shows how a model is rotated using three fingers.

For scaling, the gesture designated is the *pinching* action done with two fingers, see Fig. 5. However, there is a need to be in scale mode to be able to perform actual scaling. To be able to perform these modeling transformations, the system stores all the vertices that have been created, and it then manually performs the matrix multiplications to the

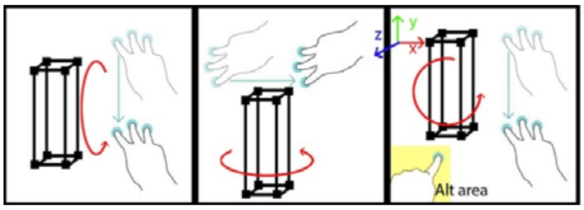

Fig. 4. Gesture for rotating a model

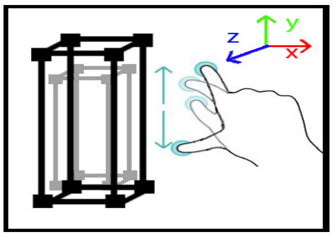

Fig. 5. Gesture for scaling a model

vertices, and renders all the vertices into the viewport using OpenGL ES.

When the system is in camera mode, the user can change the view on the mesh into seven different views: default front view, back view, top view, bottom view, left view, right view, and isometric view. The user may freely change from the different views by simply tapping the screen. This mode only accepts the tap gesture for switching views. In the implementation of the actual changing of views, the matrices that are needed for the changing of the position of the eye are already predefined and are just fed into the predefined OpenGL utility function **`gluLookAt()`**.

The other functionalities of the system that are not covered by the modes are deletion, load, and save. There are two types of deletion: delete vertex and delete edge. For delete vertex, the system does not truly remove the vertex but rather, flags that vertex as deleted and does not redraw it on the screen. For delete edge, however, there is a need to manipulate the actual data structure. First, the system checks whether the two vertices are actually on an edge. This is done by checking the data structure if the two vertices both have each other in their respective neighbors list. If that is the case, the system will then remove the vertices from the neighbor list of the other.

For the load function, the system makes use of an open source activity called Filechooser. There was no need to edit the actual activity since it only provided a medium for the system to open .obj files present on the machine. When the system has opened the file that was returned by the Filechooser activity, it reads the file then stores each vertex and face into the data structure and then renders the loaded mesh into the screen.

For the save function, the system is limited to only saving triangular meshes. After all the faces are identified, the system then writes the .obj file containing all the vertices and all the faces on the mesh into the external memory of the machine and is named based on the input of the user.

Figures 6 to 8 show some screenshots of the actual system in use. In the screen shots, the horizontal line in red color represents the x axis, the green line represents the y axis. The z-axis, which is not seen in the screenshots, is represented by a line in blue color. It cannot be seen because the x-y plane is coplanar with the plane corresponding the tablet screen. The modeler uses a right-handed coordinate system.

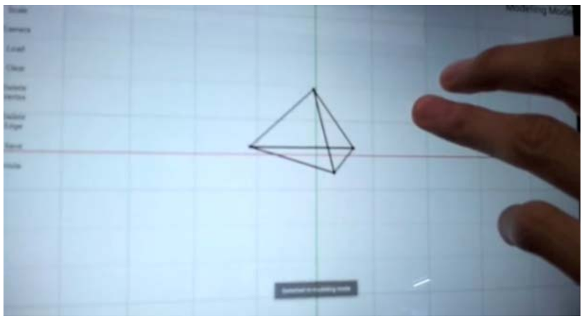

Fig. 6. Screenshot of the system with a sample pyramid mesh model

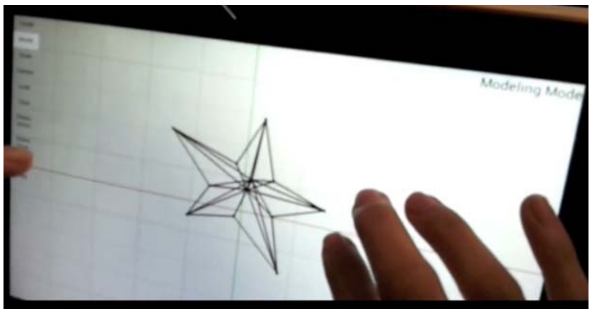

Fig. 7. Screenshot with a star shaped mesh model

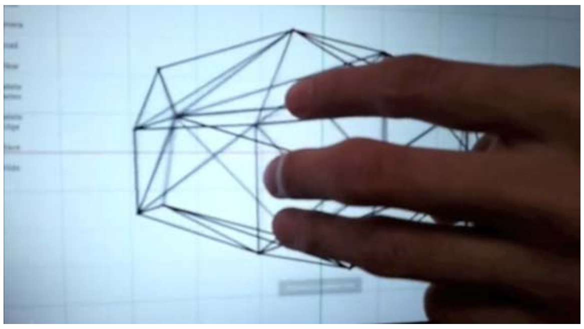

Fig. 8. Screenshot showing a user about to rotate a model using a three-finger gesture

3. TESTING, RESULTS AND ANALYSIS

The system underwent two different types of testing, i.e., a usability test and a simplified stress test.

3.1 Usability Test

The usability test was conducted with the help and input of 20 volunteer testers. They were first shown a demonstration of the functionalities of the system, specifically, on how to create and model a 3D pyramid similar to the one shown previously in Fig. 6. They were then asked to use the system to create their own pyramid model. Afterwards, they were requested to fill-up a simple evaluation form to determine how easy or difficult it was for the users to create a 3D model using the gesture-based 3D modeling system. In particular, they were asked to provide feedback on their experience in creating a model, i.e., vertices, edges, faces, mesh, selecting geometries, and modeling transformations.

The evaluation form feedback reflected that most testers found it relatively easy to perform most of the functionalities. However, the results also show that the modeling transformations, particularly translation and scaling provided the testers with the least amount of ease compared to other functionalities. It must also be noted that the testers using the 10" Asus tablet provided more positive feedback compared to the testers that used the 7" Samsung tablet. This is most probably due to the relatively smaller size of the screen and the number of fingers that had to be used for the functionalities.

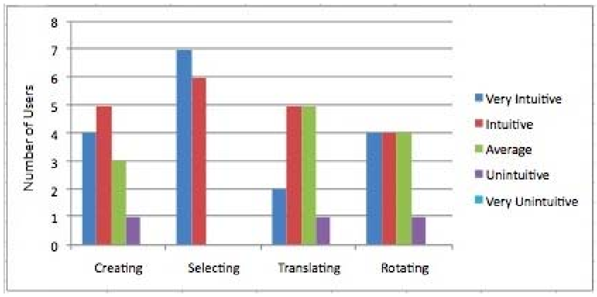

Fig. 9. Evaluation from testers with prior experience in using a 3D modeling tool

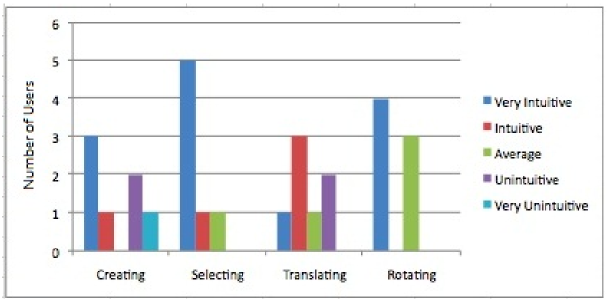

Fig. 10. Evaluation from testers without prior experience in using a 3D modeling tool

One interesting observation is that testers who had no previous experience with 3D modeling tools found the gestures intuitive compared to those who had used 3D modeling tools before. This is probably because, those who had experience in 3D modeling were able to compare their testing experience with their experience using other tools, while those without experience had no prior basis on the intuitiveness of the tool's interface. Figure 9 shows a bar graph of the feedback from testers that have prior experience in using 3D modeling tools, and Fig. 10 shows a bar graph for those without prior 3D modeling experience.

3.2 Stress Test

The stress test was used to determine how the system will perform under increasing load conditions. Load, in this case, is measured in terms of number of vertices and faces, and performance is a value based on basic observed system slowdown such as frame rate drops and application crashes. The stress test was performed on a 10" screen Asus EEepad Transformer. The test was done simply by loading 3D geometries with increasing number of vertices and faces, after which some basic 3D modeling operations were performed. Several 3D models were tested ranging from a simple cube with just 8 vertices to a pig model with the most number of geometries, i.e., 3166 vertices and 6204 faces.

The stress test performance is summarized in the graph shown in Fig. 11. It can be noted that in cases where there are less than 100 vertices, the system does not experience any performance issues. All modeling transformations are performed without any latency in refreshing and redrawing on the screen. At higher vertex counts, specifically from 150 vertices to 330 vertices, the system experiences a notable amount of latency in redrawing the 3D mesh. At even higher vertex and face counts, possibly between 400 and 600 vertices and from 600 to 1200 faces, there is a significant amount of latency in which occasionally forces the tablet to prompt the user to close the application due to unresponsiveness. At over 3000 vertices and over 6000 faces the system can no longer consistently load the mesh. It occasionally succeeds in loading the pig mesh, but more often than not, force closes the application.

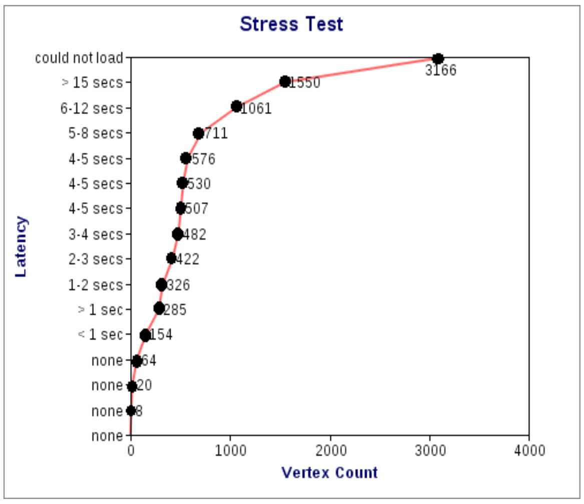

Fig. 11. Stress test result, vertex count vs. latency

These results may be caused by one factor or a combination of different factors. For one, the system is implemented on top of a number of platforms and APIs; specifically, it is implemented using OpenGL ES which is implemented on top an Android API which is implemented in Java. This causes a lot of lower level processes such as dynamic memory allocation to be abstracted during development. This may be the cause of the relatively slow performance of the system when loading hundreds to thousands of vertices and faces. Another factor could be the limitation of tablet PCs in terms of computing power.

4. CONCLUSION AND FUTURE WORK

We were able to develop a basic 3D mesh modeling tool that runs on one of the newer tablet platforms. Our project is just a groundwork for future development of 3D mesh modeling tools on tablet platforms. What we were able to develop still pales in comparison to the WIMP based 3D modeling systems available in the market today. The system we developed is also not as comprehensive as a modern 3D modeling tool should be. However, we were able to show that the concept is possible and that a 3D modeling tool on a tablet PC load 3D models from WIMP interface 3D modelers and conversely, save 3D models on a tablet PC that can be loaded onto WIMP interface 3D modelers like Blender and Wings 3D. We have also shown that the basic concepts of computer graphics such as modeling and viewing transformations can be applied on a tablet based 3D modeling tool just as it can be applied on WIMP interface 3D modelers.

There is still a lot of work that can be done with respect to the system we developed. In terms of usability, the feedback from usability testing provided some insight for future incremental improvements to the 3D modeler. The testers' feedback reflected a need for a tutorial in the application. This can be done by notifications or maybe a help option in the menu. Several testers commented that the modeling transformations are too sensitive. This implies that the calibration of the gestures and the modeling transformations need an adjustment to reduce the sensitivity. Some testers commented that certain gestures are unintuitive for its corresponding function, especially the gesture for rotation. A long term solution would be user-centric, i.e., the user will have the ability to customize his/her own controls, for example, the user can specify his own gesture for a certain action, and to set parameters such as sensitivity according to his/her personal preference.

We still need to think about how to incorporate material properties (such as color) and light sources for shading, as well as more advanced viewing and modeling transformations such as user controlled camera operations and modeling transformations such as extrusion, pushing and pulling. Inclusion of other 3D primitives such as curves and surfaces can also be grounds for future work

5. REFERENCES

Android Software Development Kit (2015). Available online http://www. developer.android.com/sdk.

Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: a sketching interface for 3D freeform design. In Proceedings of the 26th annual conference on computer graphics and interactive techniques (pp. 409-416). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Jiao, X., Deng, H., and Wang F. (2010). An investigation of two-handed manipulation and related techniques in multi-touch interaction. International Conference on Vision and Human-Machine Interface (MVHI), pp. 565-568.

Kin, K., Miller, T., Bollensdorf, B., DeRose, T., Hartmann, B., and Agrawala, M. (2011). Eden: A professional multitouch tool for constructing virtual organic environments. ACM Human Factors in Computing Systems (CHI), 1343-1352.

Yang, C., Sharon, D., and M. van de Panne. Sketch-based modeling of parameterized objects. In ACM SIGGRAPH 2005 Sketches, SIGGRAPH '05, New York, NY, USA, 2005.