An OpenGL Implementation of Teddy

Evan Flesher and Travis Hunt
CMSC 435 Final Project Report

ABSTRACT
We have designed a working prototype of Teddy, a program originally written in JAVA, in C++ using OpenGL resources and libraries. The original Teddy system was designed by Takeo Igarashi as a 2D sketch to 3D prototype system. The original will allow the user to draw a 2D shape and Teddy will take that shape and make is semi-spherical to create a 3D mapping of the image. Also in the original, the user could manipulate the 3D object to cut or add after the 3D is rendered.

Through this project, we have made compromises, both due to time and resources available, as well as our own knowledge, to create a prototype that functions in a similar format as the original Teddy system. We focused on getting the original function of Teddy, which is the 2D sketch to 3D render. We did not include the addition or cutting functions in our project. The project also does not use the same algorithm to create a 3D render as the original teddy system. This project will create and render a 3D mesh from a 2D drawing as Teddy does, using OpenGL instead of JAVA.

General Terms: siggraph, OpenGL, Java, C++.

Key Words: Mesh.

1. INTRODUCTION

The main focus of this project was to develop a system that mirrored an already implemented system that uses JAVA, using OpenGL and C++ as the backbone. Teddy was implemented for the purpose of modeling 3D objects, similar to that of CAD software and other similar tools. The original software functions properly in its role of assisting the user to create object, as it is able to render more complex geometry, and modify that geometry once it is rendered.

Creating a user friendly interface that preformed the basic job of sketching on 2D and rendering the object in 3D was the focus of the project. We were not attempting to create an easier or simpler solution to the Teddy algorithm. The focus was to develop personal skills with C++ and OpenGL as well as deepening our understanding of computer graphics as a subject. It was also to create an implementation of a useful tool that could be reused in other OpenGL programs.

To begin the project, we had to implement our understanding of how the original teddy program works. Our interface gives the user no instructions, but provides a clean canvas to begin drawing. The user must make one continuous stroke that does not intersect itself, and upon completing the line they wish to draw, the computer will assist the user in closing the shape.

Once the shape has been closed, the user has the choice of viewing the 2D output where lines and vertices are provided for the user's inspection in a geometric layout, or a clean render in the 3D space. Both views complete the same task; however, each shows a different output view, having different uses for the user.

The main focuses while developing our implementation were on switching from 2D perspective into 3D, as well as how to render a mesh. Algorithmically we had to create ways to calculate where the vertices exist on the shape in the 3d space. We then connect those vertices together logically to form the mesh.

2. MOTIVATION

We originally discovered a Teddy implementation using JAVA on YouTube while researching possible projects to implement in OpenGL. We had a previous interest in building a program that would create a 3D object given a 2D sketch. Finding the basic implementation, we researched the original Teddy program. We felt that the project would provide enough challenge to be interesting; however, we already understood the basics of how Teddy functioned. Applying the knowledge and realizing the project in OpenGL were difficult but helped us to learn the basics of computer graphics.

These basics include OpenGL, C++, and how computer graphics programming is structured. The project provided us with the proper output and algorithm, and the challenge was making it function within OpenGL. The secondary goal was to provide an OpenGL implementation for future use. Given that the program could be utilized in many different cases. The output is simple to apply compared the number of calculations that must be done, and the setup to view the output. A programmer could utilize the code and change the output viewing method to work with their configuration. The algorithm simply outputs a list of vertices and a list of indexes ordered to represent the triangles on the mesh.

3. LITERATURE UTILIZED

The resources we used were the original SIGGRAPH paper written by the creator of Teddy, Takeo Igarashi, as well as several other articles, and the class text book, "Fundamentals of Computer Graphics", 3rd edition, by Peter Shirley.

We used the original SIGGRAPH paper as a basis for our algorithms. Though some of the algorithms we utilize do not follow the same method as the original Teddy, we do utilize some of the original ideas and designs. Major differences in our design versus the Teddy program were using "Pruning" which takes some of the edges with fewer vertices and rounds them out, bringing the number of inner triangles up. We also do not work with smoothing the shape once it is drawn. We utilize a tight triangle mesh which limits the number of extreme peaks in the surface. This limits the need for the smoothing method. We also do not utilize Delaunay Triangulation as the original Teddy program does.

Several of the articles we read through were looking at Delaunay Triangulation. Looking into the process, we ultimately made the decision that we would not include it in our implementation. There were several reasons, including that we were looking for a simpler way of providing the 3D mesh and calculating the base vertices. We decided to simply rotate points on the outer edge of the shape around midpoints between them. This would create the points in 3D space for the mesh.

Other articles that we looked at were relating to building a mesh. Ultimately going with a method in the textbook, we looked extensively at the options available for building our mesh and what our outputs could be.

4. METHOD AND IMPLEMENTATION

As stated previously, our implementation is not entirely the same as the original Teddy algorithm. We still utilize the main processes; however we modified it to be simpler. Our process is this:

First the user draws a line, and the line is automatically connected back to its other end. The line must not cross itself and therefore can be any shape defined by the user. The program saves the points on the line as it is drawn. The algorithm then starts at the front and back and connects these points across the shape from each other. This is completed until the iteration meets itself in the middle of the list of points.

There is now a list of connected points on she shape. Taking these points, we can calculate the midpoint between them, and rotate the points around that to give circular pattern of points around the shape. Logically, the points are then "connected" into triangles, and this list of triangles is passed to the mesh making portion of the program.

The viewing portion of the program is where we spent a large amount of time. We worked to get lighting working as well as the transition from 2D to 3D. Getting user commands and the mesh to work also took a great deal of effort. Overall, a large portion of time was spent on the front end development where graphics were concerned and user interaction was key.

The calculations for the point generations still have bugs that need to be worked out in the future. This is in addition to the other ideas we had such as a save and load file system, tablet integration, and complete Teddy code, including the cut and add functions.

5. TAKE-HOME MESSAGE

What we learned through this project is there exist many subtleties in OpenGL and computer graphics. We had several issues where the code will work on mac and not with linux. We also experienced that there were many different ways to solve each problem we encountered and sometimes the solution was to start over. We also found that if the program is not outputting correctly, solving by hand with a test case is an easy solution to find the issue. Our major take away is that programming for user input, large compensations must be taken for unexpected input. Many times our program would break because of a test case that does not match what is expected.

Upon reflection, we discovered that programming in computer graphics takes a lot of patience and time. Expecting specific results was sometimes more that the program could handle, and there were times where the development process would halt to wait for a specific part of the program to be finished. Learning how to program as a group was somewhat challenging, but far less because there were only two of us. It was a good experience, and a project worth the time to complete.

6. CONCLUSIONS

This project was exciting and interesting. Many pieces of the project function in ways different from the original teddy program; this however is not a huge issue. This is an implementation of Teddy in OpenGL, and it is an implementation using nothing more than the SIGGRAPH written for Teddy. Given our time constraints, the outcome is not perfect, but it does function in a way similar to Teddy. We never had access to the Teddy code; however, this is a fair design despite the persistent issues. This was a learning experience and helped to develop our understanding of computer graphics, including how the user can interact with it.

Most importantly, it helped us gain an understanding of C++ and OpenGL we did not have previously. Gaining new insight and experience is very important. Completing the development of a complex system like this was worth the effort; to learn about Teddy and apply our own changes to the algorithm, this was a difficult but an interesting process.