TJHSST Computer Systems Lab Senior Research Project
Development of a 3D Graphics Engine
2006-2007

Kevin Kassing

November 1, 2006

Abstract

The purpose of my project is to investigate the workings of a 3D graphics simulation engine, and develop an engine designed to simplify the task of coding 3D simulations, while still giving the developer control over every aspect of the rendering and simulation process. Hopefully, my engine will simplify the visualization process to the point where it is worthwhile to code a 3D representation of some problem. I also hope to make it possible for the developer to use my engine without any knowledge of OpenGL or SDL, but rather do all the necessary code behind the scenes when using the default rendering methods.

Keywords: quaternion, modelview matrix, projection matrix, managed geometry

1 Introduction - Elaboration on the problem statement, purpose, and project scope

1.1 Scope of Study

I hope to develop an engine that can interact with data in a flexible manner. I would like to be able to adapt my engine to visualize as many different

problems as possible. Since different problems call for different algorithms, and completely different code structures, I need to avoid creating a rendering architecture that works in only one way. I would like to add features for graphing data as well as for creating geometry (geometric shapes, terrain, etc.). Working with existing data formats, the MD2, MD3, and MD5 mesh formats, for example, is also a priority.

As I am developing my engine, I am designing an ontology for 3D graphics simulations. Currently, it consists of "renderables", each of which has a material (texture and reflective properties) and an orientation (scale, position, and rotation). One subset of renderable refers to specific geometric shapes. The other subset is a model, which is made up of parts. Each part of a model has a mesh, which may have tags. Links are where a separate model part is aligned to a tag. These renderables by themselves don't represent much. In the future, I will be adding an "agent" object, which is a state machine that can interact with the user's data.

2 Procedures and Methodology

The engine will be programmed in C++ utilizing OpenGL for 3D graphics support. SDL is currently used for window management, but it would not be difficult to create a version utilizing GLUT, so as to ease porting the engine to Windows. Standard frustum culling and collision detection/response algorithms will be used in the engine.

My engine does not generate data which can be shown in visual form, but instead generates visuals from data. Thus, I will have some difficulty showing the results of my work. The only meaningful statistic that I can think of is to show a relationship between the number of polygons or objects to be rendered and the rendering speed, in order to show the scalability of my rendering algorithms.

I will be using performance tuning mechanisms to fine tune my engine's performance. The ability to measure the exact number of ticks (milliseconds) each process takes is very useful to me. Because I know exactly where time was spent, I can identify problems with my algorithms and optimizie them accordingly.

The only kind of testing I can do is to give as many different types of data as possible to the engine. Because of the modular design of the engine, each component must work reliably, and thus far I have noticed no significant

problems.

I will be using standard scenegraph structures for stationary geometry, including binary space partitions (BSP trees) and octrees (which are a subset of BSP trees). In these structures, planes are used to divide the geometry into sectors. Visibility calculations become much more efficient, because when one supersector can be determined to be outside the frustum, all of its child geometry can be skipped during the rendering process. I have also been investigating structures for memory and time efficient management of resources, such as meshes and textures, utilizing queues and vectors.

2.1 Expected results

My research thus far has been as much about program design (in C++ specifically and in general) as it has been about 3D graphics. I know that I could have gotten even farther than I currently am in my project if I had just started coding without thinking about design, but eventually, the work of hacking together different paradigms becomes too much of a chore and a slowdown of the engine itself. Since my project is more of an API than an application, I have used the Java API as a model of how my engine should work. I strive to conform to certain models throughout the entire API (for example, making all string arguments the std::string type instead of const char*), to use pointers and references judiciously, and to make names as predictable as possible.

In addition to program design, I am learning about the mathematics of 3D graphics, including the transformation matrices and calculations as well as useful mathematical constructs such as splines and NURBS surfaces. The goal is to use math to create the most visually pleasing graphics with the lowest computational requirements.

I do not expect my specific research into 3D graphics to benefit anyone in the future, rather, I expect to develop an engine which will facilitate developing 3D visualizations which can be applied to a variety of problems. I believe that visualization is a benefit for many problems and applications. Perhaps in developing my engine, I will touch on new and useful techniques for the visualization of data.