Development of a 3D Graphics Engine

Kevin Kassing

September 13, 2006

For this project, I would like to build a well-structured core graphics engine that utilizes OpenGL, and add on features tailored to, but not limited to, games and physical simulations. I would like to implement the following into the core engine:

- Ability to load common model formats
- Flexible, simplified lighting
- An input paradigm similar to Java's input paradigm (e.g., onKeyPressed)
- Animated skyboxes and/or skydomes
- Non-Uniform Rational B-Splines (NURBS) surfaces
- Interpolated animation
- Occlusion culling via scenegraphs (BSP trees and octrees, for example)
- Composited texturing (also known as multitexturing)
- Level of detail (LOD) rendering
- Skinning of skeletal models
- Fog and particle effects
- Environment mapping for use in reflections
- Bump mapping and normal mapping for increased realism in lighting

Features that I would like to add to the engine to extend it from a simplified interface to OpenGL into a multipurpose engine include:

- Simple physics simulation based on the Runge-Kutta 4 integration algorithm, modeling linear and angular kinematics
- Collision detection and response through the use of multiresolution maps

- Management of AI agents through a unified, state-machine based interface with message routing
- Terrain generation using fractal methods
- Navigation meshing for use in pathfinding algorithms
- Debugging methods, including flexible profiling, logging, and assert macros
- A simple scripting language for easy modification of simulated events
- A user interface using the GTK+ toolkit and/or an in-engine console

The job of a 3D graphics engine is to keep track of a set of polygons and render them through the view of a camera, often utilizing perspective. Interactive lighting adds to the "suspension of disbelief." Because of the immense amount of data required, optimization and efficiency are especially important. Scenegraph techniques utilizing predetermined tree structures aid in the efficiency of occlusion culling, or the removal of offscreen polygons. Further speed increases are available by streamlining vertex submission, using techniques that interact with video hardware to manipulate cached vertex data.

I believe that this engine would be written completely in C++, perhaps with some sections written in assembly language for efficiency. I would need to develop a simulation in parallel to be able to test the features of the engine. For the engine, there is definitely a well-defined progression that could show results in short intervals. I expect to need only the following things: a computer with hardware accelerated 3D graphics, and the libraries for OpenGL, SDL, and perhaps the SDL image library.

The result I would like to achieve is to make the process of writing interactive 3D simulations and games easy and fun, especially for those who do not wish to make the large time investment involved in learning OpenGL. I would like to present the final results in the form of some visual simulation, perhaps a walkthrough of a house which demonstrates the advanced visual effects, or a Rube Goldberg contraption that shows off the physics capabilities.