Development of a 3D Graphics Engine

Kevin Kassing – Computer Systems Lab 2006-07 – Period 5

Abstract

As computer technology moves into the new era of commonplace hardware-accelerated rendering of 3D graphics, it is becoming important to have toolkits for rendering 3D graphics in a cross-platform, library-independent manner. Most current 3D engines are specialized or have a level of complexity daunting to the beginning 3D programmer. The goal of this project is to reduce the amount of low-level management the programmer must do in order to get a program up and running. Aside from camera and matrix management, common to all OpenGL applications, this project provides management for 3D meshes, which are currently distributed in often poorly documented formats.

Procedures and Methods

The engine is designed with speed and memory efficiency as the primary goals. Modularity is achieved by abstracting the engine from the math, mesh, and material functions.

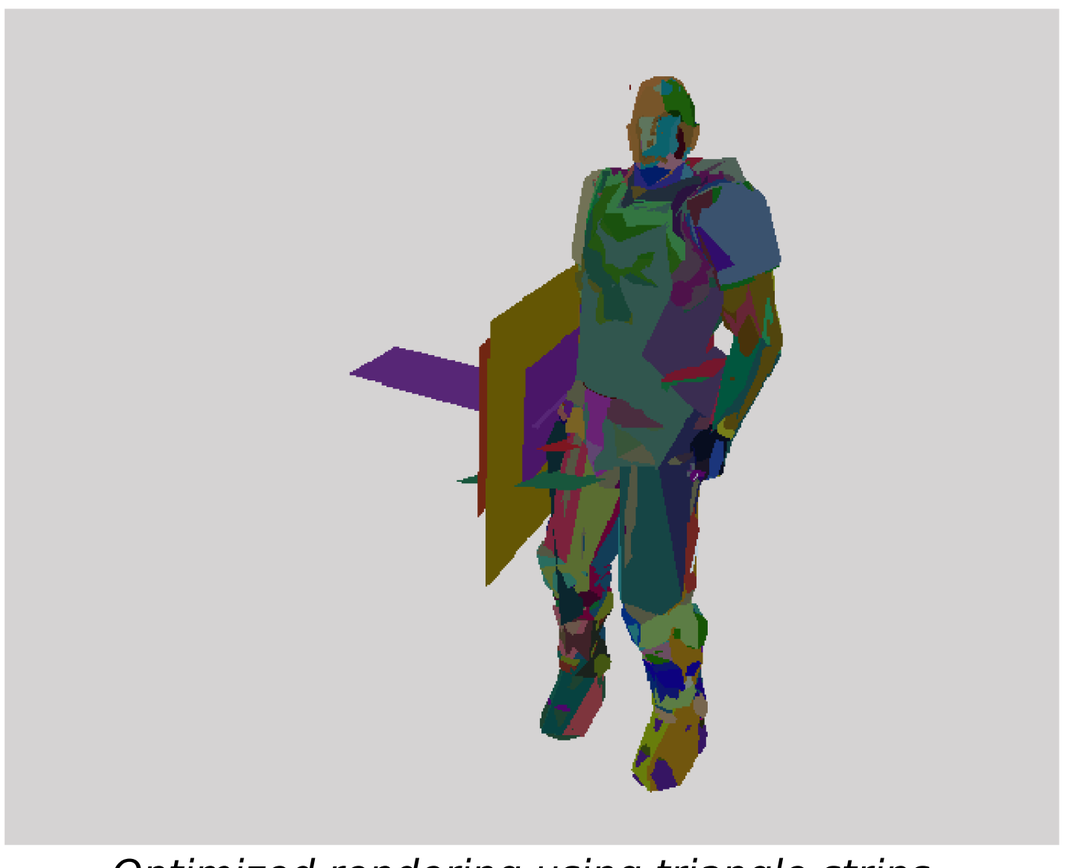
The engine consists of a basic engine framework, and accessory methods useful for 3D graphics, such as camera management. A highly optimized mathematics library provides data structures and methods for use with vectors, quaternions, matrices, and planes.

The material methods are in a separate library, which has support for multitexturing, lighting properties, and GPU shaders, which modify the fixed rendering pipeline of the graphics hardware. The library also includes image loading routines.

The mesh functions are also exported to a separate library. Support for loading from MD2, MD3, and MD5 formats is included. Vertex and skeletal animation can both be used, and methods to read and write data in the internal format are provided.

Mesh data is stored internally in the half-edge format. This allows for easier modification of the mesh topology and adjacency queries, which are useful in collision detection. Semantic data, such as triangle vertex specifications, is separated from vector data, so that the vector data may be placed in Vertex Buffer Objects for optimized transmission to the GPU.

Rendering and animating meshes in 3 formats



Optimized rendering using triangle strips

Optimizations

Included in the mesh library are functions to optimize the rendering process by exploiting the connectivity of triangles. Strips of triangles which share a common edge are algorithmically generated, creating a reduced list of vertices to be sent to the graphics hardware. After the first triangle is defined with three vertices, the next triangle is assumed to share the last two vertices of the first triangle, and the next vertex is opposite of that edge. Normally, each triangle would need all three vertices sent to the graphics card. A decrease in data sent to the graphics card results in an increased framerate.