Antigone Engine

Kevin Kassing – Period 5
2006-07

Introduction

- Antigone = “Counter Generation”
- Library of functions for simplifying 3D application development
- Written in C for speed (compatible with C++, can make bindings for other languages)
- Wrappers for commonly used OpenGL functions
- Data structures for semantic objects

Implemented Features

- Efficient camera management
- Frontend management (GLUT, SDL)
- Input API
- Debugging tools
- Included libraries:
  - liballoy: textures and materials
  - libhemi: meshes and animations
  - libmala: matrices, vectors, and quaternions

Internal Structure

- Included in the core:
  - Debugging tools
  - Camera management
  - Basic program loop
  - Input/Output API
  - Particle Engine
  - Multipass rendering facilities

- Most essential functions are in external libraries

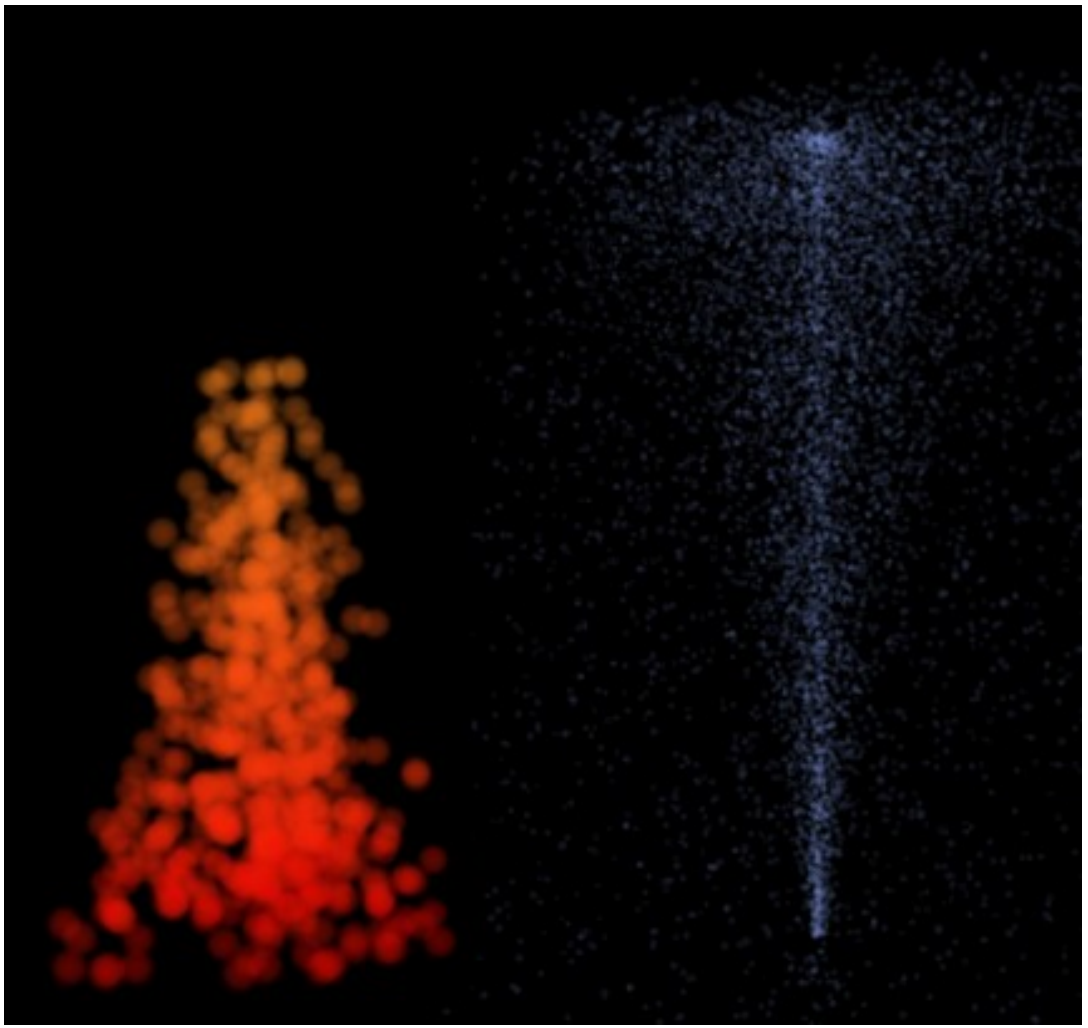
Debugging Tools

- Assert – ASSERT(a==b, “A not equal to B!”);
- Logging – wrapper for fprintf
- Profiling – measure speed of predefined blocks of code (print min, max, average ticks):

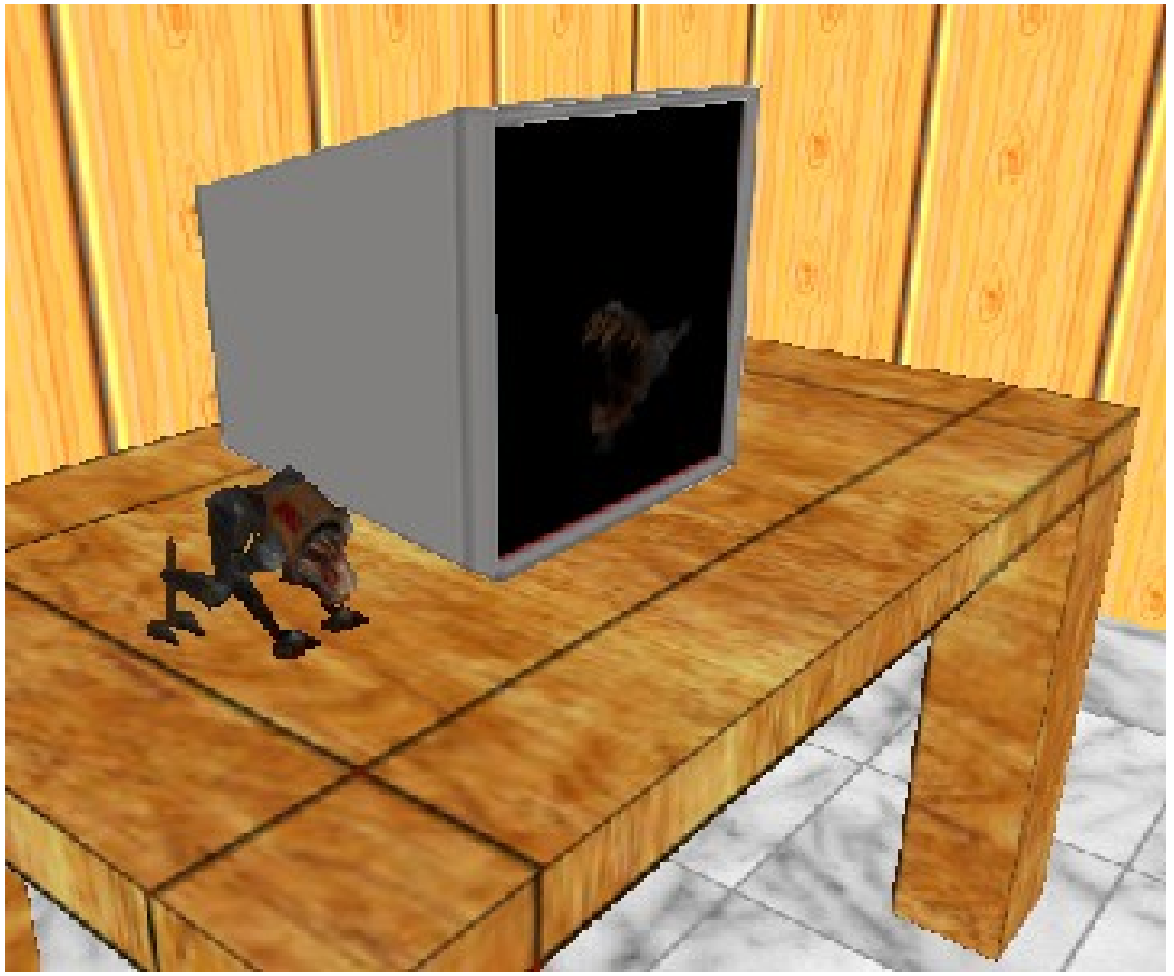
```
startSample(“Loop A”);
    for (int i=0;i<100;i++)
    {
        x=x*sqrt(y);
    }
endSample();
```

Particle Engine

- Uses lots of “billboarded” triangles (oriented toward the camera)
- Simple gravity effects and customizable colors
- Cheap way to emulate liquids and gases – smoke, fire, water, etc.



Multipass Rendering



- Render to texture can be used to dynamically animate textures
- Common use is to render something on a monitor or screen within the simulation

libmala

- Highly optimized math functions
- No dependencies
- Vector structs with 2, 3, or 4 components
- 3x3 and 4x4 matrices, square or flat
- Quaternions (w+x*i*+y*j*+z*k*)
- Basic support for common operations on transform matrices and rotations
- Limited collision detection routines

liballoy

- Depends on libmala
- Handles lighting properties, textures, multitexturing, and shaders
- Routines to read/write material definitions embedded within another file
- Support for loading from PNG, BMP, JPG, PCX, WAL, TGA
- Can optionally be compiled with other image loading libraries (SDL_image)

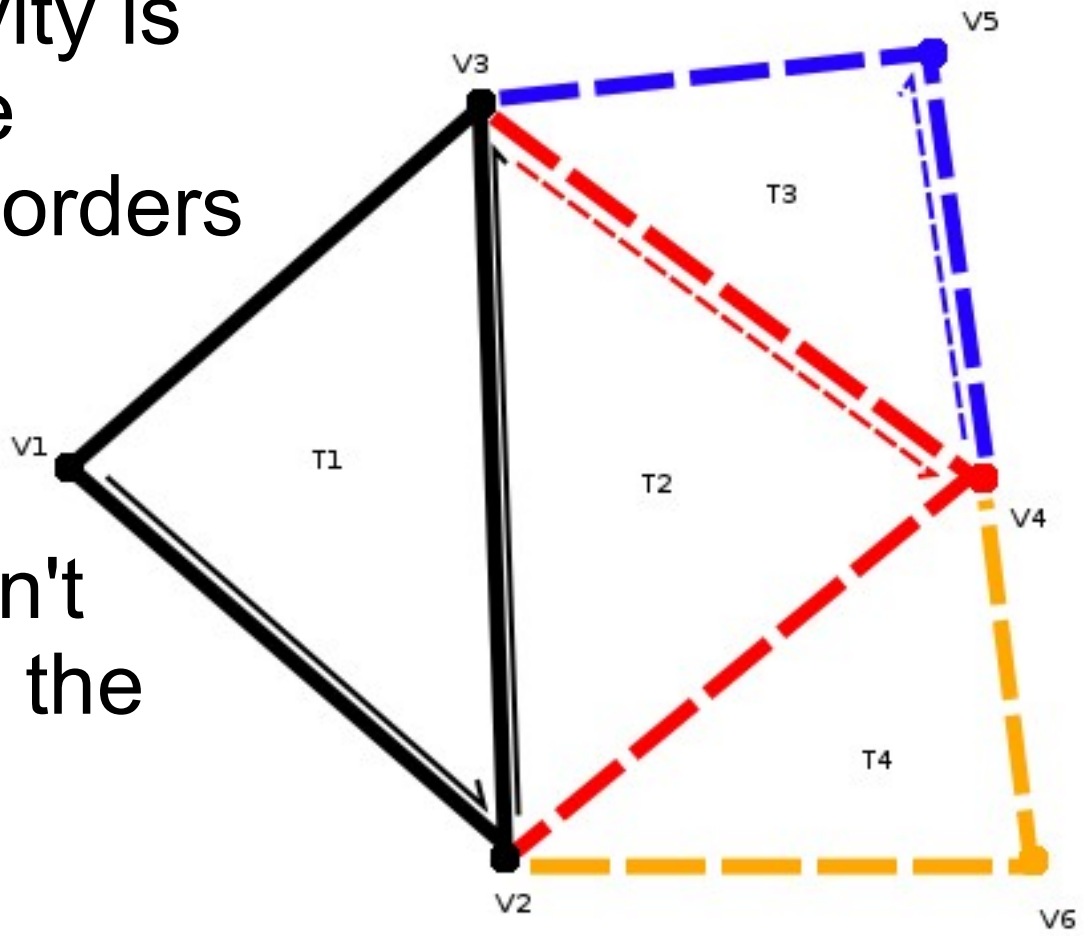
libhemi

- Depends on liballoy and libmala
- Support for loading meshes from MD2, MD3, MD5
- Support for vertex and skeletal animation
- Meshes internally represented in half-edge format
  - Each edge knows where it starts, the next edge, opposite edge
  - Allows for easy adjacency queries, mesh modifications

Mesh Optimizations

- Topology is separate from vertex position
- Triangle connectivity is exploited to create optimal rendering orders

Neighboring triangles share vertices, which don't need to be sent to the graphics card



Triangle Strips

- In skeletal models, speedup can reach 15%
- Possible to create strips averaging 30 triangles in length
- In complex models, can render using < 50% of the number of vertices as in normal rendering

Design Considerations

- Speed, Speed, Speed!
  - mesh rendering optimizations (triangle strips)
  - data structures (balance with memory constraints)
- Backwards Compatibility
  - once API reaches some point, updates may deprecate but not remove interfaces
- Modularity
  - should be able to easily add new systems (which operate efficiently), for example, sound and network

Problems

- Feature creep, especially with formats
  - Need to generalize data structures as much as possible
- Compatibility
  - Defining multiple frontends and input systems
  - OpenGL extensions / Hardware capabilities
  - Support for those who don't have library X
- Lack of documentation on formats

Conclusions

- The benefits of using C instead of C++ are marginal at best
- Texture loading functions should be handled by an existing library in most cases
- Topology should be separable from vertex positions
- Extensive consideration should be given to the future of a program which is built from libraries
- Triangle strips are well worth the amount of time they take to generate