Advanced Game Engine

Ian Lilley and Sean Lilley

Advisor: Norman I. Badler

University of Pennsylvania

ABSTRACT

Game engines are complicated. They require up-to-date knowledge of many areas of real-time rendering as well as physics, sound, and animation. Although it is usually straightforward to implement individual components of an engine, putting everything together elegantly can be a challenge. Our project will focus primarily on implementing the newest techniques in several areas of real-time rendering, including cascaded shadow maps, advanced frustum and occlusion culling, forward plus rendering, and more. Finally, we hope to engage with the graphics community by making this project open source.

Project Blog: http://gameengineers.blogspot.com/

1. INTRODUCTION

Even the simplest of games needs a functional game engine. There are three components to a good game engine: speed, adaptability, and simplicity. A game engine must be fast. It must be optimized at all levels, especially for graphics and scene management. If a game has too much lag, users will stop playing it. Next, a game engine must be adaptable. It should be able to gracefully handle many different objects, scenes, and inputs. Finally, it should be simple. And artist or designer should be able to add things to the game without worrying about the low level details. Our goal is to make a game engine that incorporates these three qualities.

Making a game engine is an interesting challenge because it forces us to combine the best techniques from many areas of real-time rendering into one package. Although individual components of an engine make for good tech demos, there is value in putting them all together while maintaining high quality and performance. Finally, the source code for most popular game engines is not publicly available, so it would be good to expose these techniques to the graphics community in our own public repository.

The first step towards accomplishing these goals is to research the newest advancements in real-time rendering, specifically forward plus rendering (discussed later). We must design a system that makes it easy to add these new features to ones we have implemented in previous projects. By the end we hope to have a robust game engine that performs well and incorporates the best aspects of game rendering.

1.1 Design Goals

This project is geared towards the game development community. Game programmers benefit because they will be able to read our code and see how we organize the different structures as well as get a glimpse of some interesting graphics techniques. Game artists and designers will benefit because we will develop tools for fast content creation.

1.2 Projects Proposed Features and Functionality

A game engine is composed of many parts. Some of the features we would like to target (in no particular order): character animation, transparency, cascaded shadow maps [DIM07][EIS11], rigid body physics, asset loading pipeline, materials and BRDFs, object instancing, frustum and occlusion culling [RAK11], level of detail, reflections and refractions, bump mapping [LEN11], post processing effects (like motion blur, depth of field, and ambient occlusion) [MOL08], game event system, level editor, user interfaces, audio, multiplayer/networking, and forward plus rendering [HIR12]. In general, these features do not depend on each other so we have the freedom to decide the order in which we implement them. A parallel goal of this project is to always render above 60 fps for complex scenes on our target hardware.

2. RELATED WORK

We are largely inspired by the amazing results of modern game engines like Unreal Engine 4 and CryEngine 3. Although we do not have the manpower to match the output of those companies, we have access to a lot of their techniques through books and papers. We will be consulting books like *Real Time Rendering* [MOL08] and *Mathematics for 3D Game Programming and Computer Graphics* [LEN11] as well as SIGGRAPH demos and papers. Also, companies like AMD and NVIDIA are constantly creating demos for their new research. We plan take from all these different sources.

3. PROJECT PROPOSAL

We will be creating a game engine from scratch. In doing so we hope to accomplish two main goals: first, to explore the newest techniques that real-time rendering has to offer, and second, to make a game. This means not only writing a lot of code, but also thinking about game design and creating art assets.

3.1 Anticipated Approach

Here we discuss some of more interesting features of our game engine:

GPU accelerated object instancing, frustum and occlusion culling, and level of detail selection – Typical game environments are quite large and contain many instances of common objects like trees, rocks, enemies, etc. We propose a completely dynamic system that updates the number of instances to draw and their level of detail based on occlusion and frustum culling tests that are performed in a compute shader. Most engines perform these tests on a per-object basis on the CPU, with constant round trips to the GPU to get results from occlusion queries. Our GPU-only technique uses some of the newest features of OpenGL including compute shaders, atomic counters, and draw indirect buffers.

Forward plus rendering [HIR12] has the performance benefits of deferred shading without the high memory cost. In addition, forward plus rendering easily supports multisampling and transparency where deferred rendering does not. The steps for doing forward plus rendering are as follows: first do a Z-prepass of the scene. Next, use a compute shader to perform light culling and create light linked lists for each pixel or tile of the screen. Finally, render the scene again and compute the color of each pixel based on the lights in that tile and the material properties of the object.

3.2 Target Platforms

Languages: C++ and OpenGL 4.3

Operating Systems: Windows and Linux. No OSX because it is unlikely that they will have OpenGL 4.3 drivers by the time we do this project

Hardware: AMD and Nvidia OpenGL 4.3 compliant graphics cards

Code Libraries:

- SFML (Simple and Fast Multimedia Library) for handling viewport, audio, networking, and user input
- Bullet Physics for physics
- TinyXML for XML parsing
- GLM - linear algebra math library
- GLI - texture loading utilities

Software:

- Blender for 3D model creation, rigging, and animation
- Gimp for texture editing

3.3 Evaluation Criteria

We will compare the visual quality of our results against those of popular game engines. We will also be benchmarking our engine, timing different areas of code and fixing bottlenecks. Our goal is to stay above 60 fps for complex scenes.

4. RESEARCH TIMELINE

In order to complete most of the features in section 1.2, we will implement smaller features on a weekly basis and larger features on a monthly basis.

Project Milestone Report (Alpha Version)

- Complete all background reading
- Complete forward plus rendering and object instancing pipeline (essentially the core graphics engine)

Project Final Deliverables

- Implement as many features as possible from the list in section 1.2. In order to do this we will first complete the core rendering engine. With the remaining time we will implement some of the less critical features like UI, networking, and level editor.
- Tech demos showing off the various features separately and together.
- Open source repository.

Project Future Tasks

- Continue to add more features. A game engine is never complete because there is always new research on the horizon.

(remove line)

You will fill in the following sections as you make progress on your project, particularly for the alpha review and the final deliverable. In these sections, list pseudo-code, charts, images, examples, etc. to show what you've done over the course of the semester.

5. Method

6. RESULTS

7. CONCLUSIONS and FUTURE WORK

APPENDIX

A. Optional Appendix

Some text here. Some text here. Some text here.

(remove line)

References

[HIR12] Takahiro Hirada, Jay McKee, Jason C. Yang: *Technology Behind AMD's Leo Demo* (GDC 2012).

[MOL08] Tomas Akenine-Moller, Eric Haines, Naty Hoffman: *Real-Time Rendering*. AK Peters, 2008.

[LEN11] Eric Lengyel: *Mathematics for 3D Game Programming and Computer Graphics*. Course Technology PTR, 2011.

[DIM07] Rouslan Dimitrov: *Cascaded Shadow Maps*. NVIDIA Corporation, 2007.

[EIS11] Elmar Eisemann, Michael Schwartz, Ulf Assarsson, Michael Wimmer: *Real-Time Shadows*. CRC Press, 2011.

[RAK11] Daniel Rakos: *Hierarchical-Z Map Based Occlusion Culling*, http://rastergrid.com/blog/2010/10/hier-archical-z-map-based-occlusion-culling/

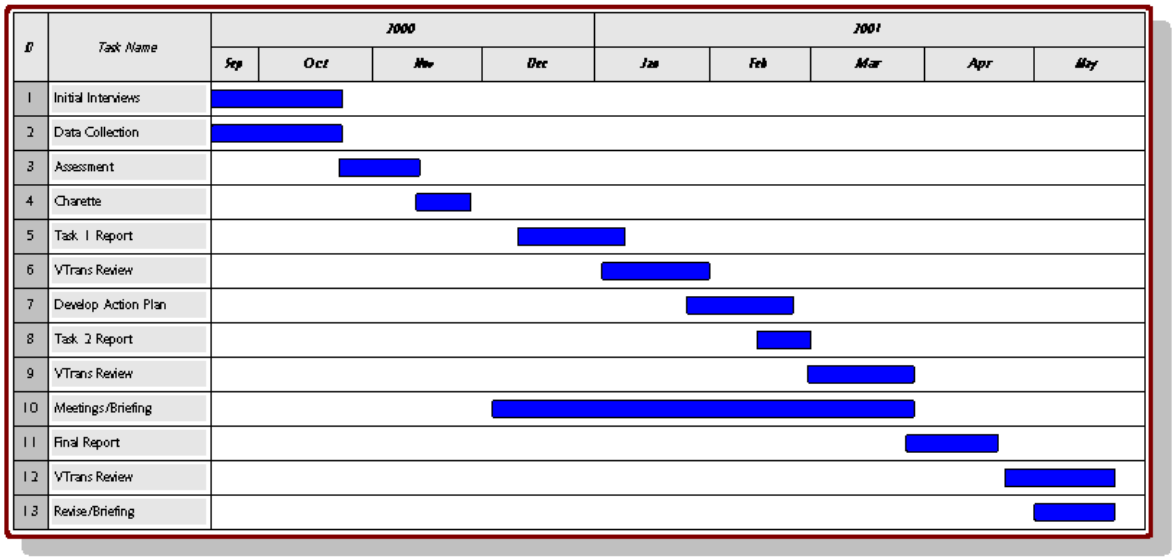

Figure 2: *For publications with color tables and figures that span two columns like your gant chart or results will.*