Game Engines

Documentation of Engine

Written by:

Emil Erik Hansen	14-06-85	`emha@itu.dk`
Julian Møller	18-03-87	`jumo@itu.dk`
Mads Johansen	22-05-85	`madj@itu.dk`
René Korsgaard	28-05-84	`reko@itu.dk`
Steen Nordsmark Pedersen	15-02-87	`snop@itu.dk`

Group: "GROUPNAME"

IT University of Copenhagen

December 2011

Contents

Formalities 1

The engine was the product of the course *Game Engines*, at the *IT University of Copenhagen*, Fall of 2011.

In this documentation, the engine is described according to the requirements specified in the two following sections.

1.1 Documentation Requirements

The following were required to be included in the documentation. The various chapters and sections are structured in such a way that it should be fairly easy to quickly find the needed information, based on the requirements.

- **Engine Scope**: Does it target a specific genre? What range of games do you have in mind that the engine would be suitable for?
  (section 2.1, page 4)
- **Major Features**: What are the main features your engine supports?
  (section 2.3, page 4)
- **Implementation Overview**: Give a high-level explanation of how the engine is structured (perhaps with an architecture diagram). Then, give at least brief explanations of how the major subsystems work, delving into more details for components that are particularly interesting, advanced, or unusual. *(section 3, page 5)*
- **Design Rationales**: Were some of the design decisions made after considering alternatives, and do you have rationales for why you made the decisions you did? If so, tell us! *(section 2.2, page 4)*
- **Examples**: Give some examples illustrating interesting features of your engine and how they'd be used. Depending on the features, these can take the form of screenshots, code snippets, flow charts, and/or prose explanation. *(section 4, page 6)*

1.2 Engine Requirements

The following describes the actual requirements of the engine. For each, a reference to the section and page in which it is described is given.

A Unique/Advanced Element

The feature "Performance Logging" is described in section 3.6, page 5.

- Pick either one area of the engine to add advanced functionality to, or at least one optional feature to add. For example: a user-programmable shader interface, procedural terrain, advanced physics, an audio subsystem, etc.

Dynamic Elements

Described in section 3.1, page 5.

- *Physics and animation*: What makes sense for your engine and why? Decide on a physics/animation split for your engine, considering its target. Do you want forces, manual animations, parametric curves, something else?
- *Collision detection*: have collider surfaces for your objects, with at least basic collision detection. If you need performance (lots of objects), GJK may be best.
- *Collision response*, if it makes sense for your genre of game: things like objects bouncing off other objects when they collide.
- Be able to *add/remove objects from the game world dynamically*, and update associate data structures.

Resource Management

Described in section 3.2, page 5.

- *Singleton class* (or several) managing global engine state.
- *Memory-management system*. Design on paper first. Implement at least one kind of custom allocator, e.g. a stack allocator used for per-frame allocations.
- Design a *resource/dependency-management system*, both runtime and offline portion (offline includes things like asset conditioning, and level format).
- Quickest way to get some resources loading: wrap a library like `http://assimp.sourceforge.net`.

Rendering

Described in section 3.3, page 5.

- *Object-oriented, scenegraph-based rendering*, that recursively renders objects in the scene.
- A *camera component* should be present.
- Use a method of *speeding up rendering* by retaining data on the GPU. For now display lists are okay, but vertex buffers are better (and will be useful later).
- *Lighting manager* (or data structures) storing information on lights in the scene.
- *Material properties* on objects.
- *Textures* on objects.
- *Shaders*: Optional. A user-programmable shader API for your engine could be a nice enhancement. But, everyone should understand how shaders work, even if your engine doesn't use them.

Game Loop

Described in section 3.4, page 5.

- Support *framerate-independent game-world updates* (e.g. using deltaT timings).
- Implement an *event system* (may be useful to split into several event systems, e.g. an InputManager handling all input events).

Input

Described in section 3.5, page 5.

- Handle *keyboard* input, with at least a few options. For example: event-based input that queues multiple keypresses, and polling-based input that does something while the key is held down.
- Handle *mouse* (or joystick/controller) input, with the option for nonlinear mapping.
- Connect input to scene in a basic way, e.g. wasd moves an object, mouse moves camera.

Engine Description 2

More of an actual light introduction. Description of the engine and what it can be used for. Covers the ideas and design. See the sections below.

2.1 Engine Scope

The engine targets the “Hero RTS[1]”-genre. This includes games like *Heroes of Newerth* and *League of Legends*, based on the *WarCraft III* modification *Defense of the Ancients*.

A more broad and loose description would be multi-player games, in which each player only has one unit, that requires precision and quick reflexes.

2.2 Design Rationales

Describe various design choices in detail.

2.3 Major Features

The things that makes the engine special, the primary forces and so on. There is no further explanations of what this part should contain.

[1]Real-Time Strategy

3 Implementation Overview

More detailed about the structure of the various parts of the engine. The sections below are taken from the requirements of each of the major parts, but might need to be re-structured after that.

3.1 Dynamic Elements

3.2 Resource Management

3.3 Rendering

3.4 Game Loop

3.5 Input

3.6 Performance Logging

Examples 4

Giving examples is one of the requirements of the documentation. It could either be done throughout the document, or simply be here in a chapter for itself. Most likely a better idea, to keep it from the more in-depth technical facts.

Concluding Comments 5

Might be a good idea with a minor wrap-up at the end.

Also, a better name could be used for this section - but a fitting alternative is eluding me at this point.