Game Graphics Programming Project Document

Margaret Dorsey and Jesse Cooper

October 19, 2016

1 Project Documentation

1.1 Concept

The main goal of the project is to showcase the various graphical and lighting effects available within the engine, and to build an atmospheric environment for the player to explore. To this end, the actual functionality of the project is somewhat limited, and the design decisions of the project are mostly based on technical considerations, atmosphere building, cohesiveness, and exploration experience.

More specifically, the goal is the horror genre, and so the end goal of the effects we use and the environments we build is to contribute to a feeling of isolation, vulnerability, and terror.

1.2 Inspiration and Reference

1.2.1 Related Titles

The project is very similar to the Konami demo *PT*, the indie title *Gone Home*, and other similar titles, where the game is exploration based in a closed, highly scripted environment.

1.2.2 Reference Material

1.3 Scope, Behavior and Technical Requirements

The scope of the project is relatively small, given the relatively small time frame given by the class and our relative lack of development experience. The primary functionality will be simple collision detection and resolution, navigation of a 3d environment through a first person perspective, and dynamic changing of scenes and lighting (flashlight toggle) based on player behavior.

However, the technical requirements for rendering are significantly heavier - we will require at a minimum particle system generation and dynamic shadows. A list of tentative technical goals follows.

- Real Time Shadows

- Fire, Smoke, Snow (Particle Systems)
- Real Time Reflections (Glass, Metal, etc)
- Water
- Stretch Goals : Sound and Simple Animation

2 Engine Documentation

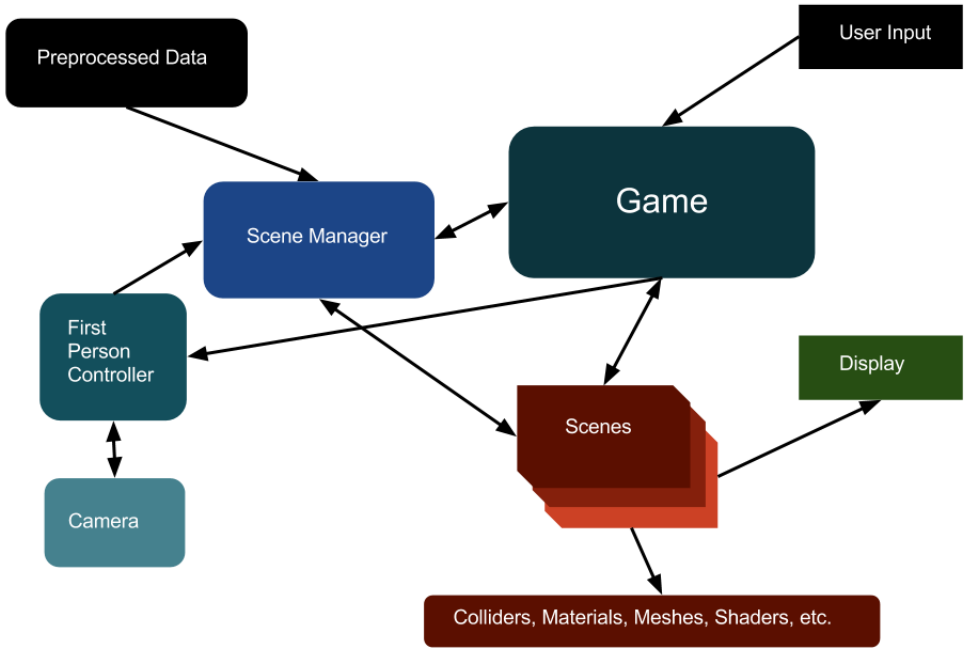
2.1 Design and Target Functionality

To suit the inherent modularity of the project, the engine is structured to dynamically switch between pre-built scenes, provided in scene files, at runtime. This modularity allows our engine to handle mid to high poly models and mid to high resolution textures, as well as a host of graphical effects, without the need for advanced optimization methods or a much deeper engine architecture.

With this in mind, the scenes are constructed as parallel arrays of object information, in order to keep object data modular and expedite operations like collision checking, shader changes, or movement without costly cache misses. Additionally, scenes are loaded into a scene manager at runtime, where they can be dynamically switched between with virtually no load time - replacement is available without reindexing in case memory use becomes a concern.

Also thanks to the modularity of the architecture, it should be relatively simple to add new data to all objects for future graphics considerations without having to re-architect.

2.2 Architecture Diagram



2.3 Notes on Architecture

Memory management is done through a reference system, so that Meshes, Materials, etc do not need to be duplicated to be reused, and 'ownership' of them need not be determined. The First Person Controller is ''owned', built, and released by the Scene Manager. While in a more robust engine there would certainly be more dependencies, but luckily our structure can be kept relatively simple.

2.4 External Tools

To reduce initial load times and the time cost of designing and implementing scenes, we will be building external tools that generate scene files, to be read by the scene manager to build the scene in engine, and rewrite the standard obj files to include binormal and tangent vectors, as well as a new set of indices to account for this extra data. As the project develops, this modified obj file may be expanded to include more data that can be precomputed outside of runtime.

2.5 File Formats

2.5.1 Model File Format

The model file format, .model, is a modified .obj format that includes binormal and tangent vectors per vertex. A sample .model file can be found in the Documentation directory of the project.

2.5.2 Scene File Format

The scene file format, .scene, contains all the meshes, textures, shaders, lights, and object and material information necessary to construct a scene. A sample .scene file can be found in the Documentation directory of the project.

3 Task List and Timetable

3.1 Task List

End Goals

- Working Collision
- Working First Person Controller
- Seamless Scene Swaps through Doors & Staircases
- External Tools
- Particle Systems (Snow and Fire)
- Real Time Shadows

- House Layout/Design
- Culling
- Advanced Graphics Effects (to be added as they become feasible)

Implementation Dependency

The external tools must be done before we can meaningfully build scenes, as writing scene files by hand is neither efficient nor intuitive. The house layout must also be at least roughed out before we can build the scenes. After the basic framework of the environment, as well as the FPC and the collision system are all implemented in engine, we actually have tremendous freedom to modularly add functionality as we desire, which will make division of work significantly simpler.

3.2 Working Timetable

3.3 Communication and Code Sharing

Because we have a small team size, there is not much structure neccessary to facilitate communication and code sharing. The project source control through git doubles as code-sharing through github, and other files are shared through a Google Drive folder. Communication is primarily through text message and personal email.