Geodesuka Engine

Alexander J. Jossart

June 15, 2022

Chapter 1

Setup

no no noo

Chapter 2

The Engine

yes yes yes In the main code directory exists several three letter folders, all being lower case since the Windows OS does not disinguish between files that are lower case or upper case, for backwards compatibility reasons. In the main repository, there exists serveral subdirectories of the main repository, for easy compiling and linking to other projects. In any repository created by Void Star Entertainment, the following directories will be used.

```
bin/
bld/
doc/
inc/
lib/
obj/
res/
src/
```

The bin/ directory is used for compiling binary executables.

The bld/ directory stands for the Build Directory. The bld/ directory is used for platform specific builds of the repository. In the case of the Geodesuka Engine, CMake has not been properly set up, and the engine only compiles out for Windows & Linux Desktop operating systems. There are future plans to incorporate mobile platforms, and MacOS, but currently the development of the engine is taking priority.

The doc/ directory, this directory contains useful documentation of the engine. Including this document which is a formated book of the engine. This document will be updated as regularly as the engine.

The inc/ directory is your average repos include directory. It contains all necessary public headers of the engine to link against the engine from any repository. Currently there is no support for dynamic library linking, the engine will be reporpused for dynamically linked libary builds later. The bin/ directory is used for compiling binary executables. The bld/ directory stands for the Build Directory. The bld/ directory is used for platform specific builds of the repository. In the case of the Geodesuka Engine, CMake has not been properly set up, and the engine only compiles out for Windows & Linux Desktop operating systems. There are future plans to incorporate mobile platforms, and MacOS,

but currently the development of the engine is taking priority.

The doc/ directory, this directory contains useful documentation of the engine. Including this document which is a formated book of the engine. This document will be updated as regularly as the engine.

The inc/ directory is your average repos include directory. It contains all necessary public headers of the engine to link against the engine from any repository. Currently there is no support for dynamic library linking, the engine will be reporpused for dynamically linked libary builds later.

2.0.1 Conventions

2.0.2 geodesuka/engine.h

In this file exists the engine code itself. The engine is at the top of the namespace "geodesuka", which encapsulates all classes, and functions inside of it. The engine uses this namespace to prevent namespace collisions with fairly common objects in other libraries. The geodesuka engine is ultimately at the end of the day, a vulkan game engine. There are no ambitions to move to other platform specific Graphics & Compute APIs to be the backend of this engine. Which is why even in the engine.h file, $\#include < vulkan/vulkan.h >$ is publically exposed front end. The engine and its core libraries rely on several platform specific libraries to interact with platform specific APIs to facilitate vulkan for generalized high performance graphics & computation.

The engine relies on several third party system abstraction libraries such as following: GLFW & PortAudio. GLFW is the Graphics Library FrameWork API. It allows the user to interact with operating system specific windowing systems that can be later used for high performance rendering. It currently supports the following windowing system APIs, WIN32, X11, Wayland, OSMesa, and Cocoa. GLFW also permits some user input and event querying with its API, aside from just system window management. This gives GLFW a wide net coverage to most Operating Systems on the Desktop. The only platforms GLFW does not support is mobile, such as Android or iOS. This is because a lot of these graphical applications take extra steps to set up, and primarily input on mobile is qualitatively different. There are other libraries out there that attempt to abstract over these platforms, like GLFW, but they require control over the applications entry point. When Geodesuka moves onto the mobile platform, it might be necessary to provide a copy and paste entry point file to the engine user to setup and modify for whatever he/she/they wish to do with it. For now, mobile support is largely neglected for the Geodesuka Engine.

The second API, which has not been included, but will be included in future releases is PortAudio, for system audio playback. As of right now, Geodesuka is completely devoid of any audio or sound management or processing libraries. In the future, PortAudio will be used to interact with platform specific audio systems.

The job of the Geodesuka Engine is to backend everything that the Engine user shouldn't, or doesn't, want to do. The Geodesuka Engine is a multithreaded engine that consists of four backend threads each tasked with unique responsibilities. The heart of the engine is the update thread. The update thread is responsible for updating all created device contexts, objects, and stages. The update thread is locked to a fixed timestep, defined on the Engine User Side in app.h. The update thread will attempt to loop over all contexts, objects, and stages within the period of time specified. If expensive operations are being performed, the update thread may run slower than the specified time step and take longer to complete requested operations.

The Render Thread is unique compared to most other synchronized engines currently out there such as Unreal Engine, Unity, Godot, and so on. The Unreal Engine utilizes a synchronous triple threaded setup where it has a GameThread, a RenderThread, and a RHI Thread. The Game thread is typically working on the frame ahead of the RenderThread. There is a level of synchronization in the Unreal Engine setup. The RenderThread typically consumes what the GameThread produces. There is backbuffering so they can work on seperate things at the same time. The Geodesuka Engine works in a slightly similar fashion such that the update thread produces while the render thread consumes. The render thread however consumes asynchronously to the update thread. On top of that, the render thread is not fixed to a particular frame rate of a single window. In the Geodesuka Engine, the job of the render thread is to honor various frame rates, or event based rendering of respective render targets in each of the stages the engine user has created over the lifetime of the engine instance. In a sort of way, the render thread is event based such that it iterates through all stages with their respective list of render targets and checks if a certain time has ellapsed since the render target has rendered a scene. Which is why it is emphasized that the Geodesuka Engine is slightly different from most other game engines out there. It is entirely possible that the Update Thread Time Step is completely different that whatever frame rate a render target is operating at. With multiple render targets, it almost becomes inevitable that the Update Thread time step and each of the render targets don't have the same processing rate. The job of the Render Thread is to query and aggregate all Graphics & Computation commands from all objects in the various stages it wishes to render for each device context.

The audio thread is incomplete, but when a full fledged audio system is created, it will also run at its own frequency.

2.0.3 geodesuka/core

The core libraries of the Geodesuka Engine are the main primitives and objects the engine interacts with. More specifically, contexts, objects, and stages.

```
math/
util/
io/
hid/
gcl/
graphics/
sound/
```

```
network/
object/
stage/
```

2.0.4 geodesuka/core/object/object.h

object_t is the base object which contains a set of qualities that all classes derived from object_t will have. Notions such as position, direction, momentum, and so on, are the qualities that all objects share. All objects must exist within a space where position and momentum make sense, namely a stage. Later in this chapter, a stage will be described in detail. It is simply a collection of objects that share the same space, physics and rendering operations. It can be considered as a base template for the world of objects in the engine. While the engine tracks the creation of all objects, objects by themselves make exist within a space for position, direction, and momentum to exists. The stage defines these things.

2.0.5 geodesuka/core/object/rendertarget.h

The rendertarget class is a child of the class object_t. It inherits many generalized object properties from its parent class. The rendertarget class is the parent class for all objects that are defined to have a collection of attachments that

obj|rt	system_display	system_window	virtual_window	camera2d	camera3d
desktop	×|✓	✓|×	×|×	×|×	×|×
canvas	✓|×	✓|✓	✓|✓	✓|×	✓|×
scene2d	×|×	×|×	✓|×	✓|✓	×|×
scene3d	×|×	×|×	✓|×	×|×	✓|✓

2.0.6 geodesuka/core/stage.h

A stage is simply defined as a collection of objects sharing the same space, physics, and rendering operations.