Technical Design Document

DigiPen Institute of Technology

April 19th, 2019

NERDHERD

Sai Sindhu Jannali

Producer | Audio Programmer

Sidhant Tumma

Engine Programmer | Graphics Programmer

Shantanu Chauhan

Physics Programmer | Gameplay Programmer

Vidhi Soni

Graphics Programmer

Sairaj Padghe

Tools Programmer | Gameplay Programmer

VY Pham

Artist (Voluntary)

Aight Engine

TECHNICAL DESIGN DOCUMENT

TECHNICAL OVERVIEW

SUMMARY

The Aight engine is component-based architecture with 2D rendering written in C++. OpenGL is used for hardware-accelerated graphics and SDL for windowing and input handling. The engine is data-driven and uses FMOD API for audio. The engine has custom physics engine with SAT and impulse-based collision resolution.

TECHNICAL SPECIFICATION

Language: C++, Lua

IDE: Visual Studio 2017

Libraries: Freetype, ImGui, LuaBridge, GLEW, SDL, Rapidjson,GLM,

API: FMOD, OpenGL(4.2 and above)

Hardware: Nvidia Graphics card, Windows 10

OTHER

Source Control: DigiPen Git, BitBucket

Planning , Scheduling and Bug tracking: Microsoft Teams

Software Development Model: Agile Development Model

Coding Convention: google.github.io/styleguide/cppguide.html

ENGINE SPECIFICATION

ARCHITECTURE

Our engine is built using component based architecture. Game object manager will be responsible to keep their components updated.
To make runtime more efficient, we load all the required resources when the engine initializes. Pre-compiled header are used to make compilation faster. Only most commonly used includes are pre-compiled.

MANAGERS & COMPONENTS

Being a component based engine, the managers are used to handle the components for different game objects. The managers have singleton design pattern.

The core systems implemented are:

Input Manager

Resource Manager

Frame Rate Controller

Audio Manager

Memory Manager

Physics Manager

Collision Manager

GameState Manager

Render Manager

Event Manager

GameObject Factory

GameObject Factory

The game object manager is used to create, delete and modify the objects in the game while the game state manager is responsible for preparing the managers for every level. The object factory is used to read the JSON files that has the level and game object information using rapidJSON. All the creation and deletion uses a dedicated memory manager.

All in-game objects creation and deletion are handled by memory manager, enhancing the performance of the game. Object pooling of memory for game objects and components are maintained.

Memory Manager

All in-game objects creation and deletion are handled by memory manager, enhancing the performance of the game. Object pooling of memory for game objects and components are maintained.

Graphics

Our engine uses OpenGL for hardware-accelerated graphics. The rendering engine is capable of rendering 2D textures of png, jpegs file format. TextureCache is implemented to refer textures that are already loaded on GPU for fast rendering.

Rendering manager creates primitive shapes like triangle, quad and circle using VBOs, VAOs and IBOs for each shape. It uses a single instance of a shape to create multiple objects of that type.

Renderer stores struct of data required for drawing objects in deque optimizing rendering flow.

Rendering manager is capable of rendering texts on screen for the UI. The engine supports Phong lighting model. Our renderer supports particle system to make it visually appealing. We also have incorporated bloom. This post processing effect will be used to retain the theme of the game. Portal and lights are implemented to enhance the look and feel of the game.

Physics

The physics manager handles all the interactions between all the objects. What happens to the player when it hits a wall or an enemy, how far the player will be pushed back by a wall, how will the gravity act upon the player. All of these interactions will be handled by the physics manager. At the current moment the physics manager handles all the interactions using “Impulse Based Collision Model”, this model is perfect for a 2D game as it prevents interpenetration issues(with some additional work arounds). Coulomb's law is used to model friction, we can now create static and dynamic friction interactions with all the objects. Combining both of these is helping us to control the interactions between the bodies in minute

detail.
All the player based movement is handled using forces, Semi Implicit Euler is used to propagate the bodies. The collision detection system used for now is simple AABB.

So for now the physics handles all of these things:-

- Player/object movement using forces.
- Collision detection using simple AABB
- Collision resolution using Impulses.
- Coulomb's law to handle static and dynamic friction.
- Basic interpenetration solver.
- The physics engine is works on deterministic time.
- Dynamic AABB tree for the broad phase.

AI

The game engine is designed to have prompters and linear patrolling enemies so that the game play is interesting for the player. The prompting acts as a Non-Player Companion which aids the player in movement and for better understanding of the game.

Audio

The audio manager is responsible to order the songs and sfx being played during the game. The current scope is restricted to 2D audio. The manager is capable of changing the volume, pause, change the songs being played that belong to objects with audio component. The audio component specifies the parameters such as volume, isBGM, isLoopable,etc. FMOD api is used for the audio engine. The audio manager is also capable of fading-in and fading-out the background music. There are different

Input

InputManager makes use of the SDL API. All the input is drawn from SDL's event system. Our engine supports keyboard for input. Functions are provided checking the current key and button states. The input manager does not send events.

Basic Event states supported are :

- Button Up
- Button Hold
- Button Down

Event Manager

All messaging between game objects is handled using event manager. Game objects can subscribe to events for getting notified on occurance of particular event. Event manager is capable of handling timed events. Events created by event manager can hold values related to the event for use by the subscribed gameobjects.

Level Editor

To make the level creation process easier, an in-game level editor is integrated into the engine. Imgui is used for the editor. The parser for writing and reading the json files were coded from scratch. Also in level editor we could create prefabs or used previously created prefabs. It supports creating and editing existing levels.

Debugging

Imgui windows are being used for displaying debug information such as frame rate, physics values of gameobjects and camera info. Various imgui buttons are used for enabling and disabling debugging options.

Scripting

Lua is used for scripting and LuaBridge for mapping data, functions back and forth between C++ and Lua. Currently we have implemented a Lua_Update function inside PlayerState class which read values from script which is called whenever player input is triggered.

Currently script contains playerState values like forceX,forceY and animaState. So we are able to change these values at runtime.

Future Scope:

Inspired by Unity, Aight Engine is planned to implement script as a separate component that will be used by all the game objects and gameplay logic for the game objects will be written in the script.