**FUNCTIONAL SPECIFICATION**

**Sulfur Engine**

**2017**

**Midnight Rage**

**Dylan Norris and Maxim Kolesnik**

Contents

[**1.** **Introduction** 3](#_Toc464759451)

[1.1 Outcomes 3](#_Toc464759452)

[1.2 Project Requirements 3](#_Toc464759453)

[1.3 Development Requirements 3](#_Toc464759454)

[**2. Functional Description** 4](#_Toc464759455)

[2.1 Graphics 4](#_Toc464759456)

[2.2 Physics 4](#_Toc464759457)

[2.3 Editor 4](#_Toc464759458)

# **Introduction**

Sulfur Engine is a tech heavy project with focus on graphics and physics. The game editor will be developed on top of the engine to provide programmers an ability to create content and, most importantly, to test different graphics or physics programming techniques with ease.

The purpose of this project is for programmers to learn and implement advanced techniques within a working 3D game engine. Since the game editor is being developed, this project will allow programmers to benchmark different techniques on various types of game worlds.

Demo of the project will demonstrate different advanced techniques used to implement graphics and physics.

## Outcomes

The demo will demonstrate following features of the engine:

* Complex rendering and physics simulation at 60 fps
* Several levels featuring different aspects of graphics and physics engines
* Depending on a techniques are being showcased the level structure may vary:

1. The level can be developed as mini-game, for example to show rigid body physics.
2. The level can be as point-click interaction, for example to show cloth simulation.

## Project Requirements

* The final demo should be installable on Windows 7 or better.
* The project requires mouse and a keyboard

## Development Requirements

* Visual Studio 2015
* Qt 5.5
* DirectX 11

# **2. Functional Description**

Sulfur Engine utilizes component-based architecture, which means a game object is a set of components, can be attached or detached on run-time, that determines functionality of the object. To parallelize the engine it was decided to go with fiber-based task management. This approach eliminates the need for memory synchronization, since tasks are updated based on dependency graph. That means concurrent tasks should not write to the same memory location. The engine uses custom logging system, so developers can inspect logs and find out where warnings or errors have occurred.

## 2.1 Graphics

## 2.2 Physics

It is planned to implement rigid body. Rigid body is updated as follows:

* Partition the game world using dynamic AABB tree, BSP tree and possibly portals
* As a broad phase determine pairs of possible collisions
* As a broad phase determine which pair are actually colliding using GJK or optimized SAT
* Resolve collisions using sequential impulse solver
* Integrate bodies

## Editor

* Saving/Loading levels
* Game object creation
* Game object’s component modification
* Game object manipulation
* Play/Pause/Stop the game level

## 2.4 User Interaction

In the editor the user can toggle camera by pressing right mouse button. While holding it, WSAD can be used to move the camera, and mouse movement can be used as rotation

Depending on a level the user should have different ways to interact with the world.

For the level where rigid body is showcased, the user will be to spawn different objects with F1 and F2.