

[**Faculty of computer science**](https://www.facebook.com/groups/ACUCS2014/?ref=group_header)

**Software Engineering**

**Software Requirements Specification**

**For**

**GAME ENGINE**

**Prepared by**

**MOSTAFA GAMAL ABD EL NASSER**

**MOHAMED ABD EL HAFIZ YOUSSIF**

**Under Supervision**

**DR. SHERIF EL SHAFEI**

**Table of Contents**

**1. Introduction 1**

1.1 Purpose

1.2 Problem definition

1.3 Intended Audience and Reading Suggestions

**2. Overall Description**

2.1 Operating Environment

2.2 Design and Implementation Constraints

2.3 Assumptions and Dependencies

**3. External Interface Requirements**

3.1 Hardware Interfaces

3.2 Software Interfaces

**4. Nonfunctional Requirements**

**5. Other Requirements**

# ABSTRACT

An OpenGl environment or game environment that demonstrates modern computer graphics skills (ex: creating and loading buildings, characters ,shapes and applying textures and light and all properties needed to make the objects looks natural

Apply animation to the objects), plus a GUI that controls the environment properties using antTweakBar or self-made GUI.

**Description:**

First of all we would like to mention that we have already mastered some rendering skills and we can start implementing those skills, and we working on some and more skills, so first we will be listing those skills below and then we start describing the project.

Basic and the essential:

This part is the game environment part where we start to render primitives and load objects and put them all together and form a natural based scene, this scene components are (streets, cars, buildings, characters, trees ,,etc ).

then we apply all the rendering skills listed above for the sake of (realism), we will be exploring the scene using FPS camera and keyboard inputs to move, (optional idea: related skills or skills under the same category are to be demonstrated inside its own building ex: lighting and shadow building, and another building for particles and water simulations and so on), we will be able to manipulate the scene properties using GUI like light color or light intensity, adjusting objects positions, enable and disable some properties, rotate or scale objects and it should give us almost full control over the scene without being that complex.

Progress skills:

This part is the intro part where we play a short animated scene for the sake of (entertainment) and also this scene is all about a creative idea or something special about OpenGL.

SKILLS:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HDR effect.   |  | | --- | | Drawing primitives | | Drawing primitives | | Applying textures | | Applying transformation (rotation, translation and scaling) | | Applying transformation (rotation, translation and scaling) | | Using mouse and keyboard as input actions to interact with the program | | Applying different types of light(point, directional, and spotlight) | | Model loading using the assimp library | | Writing shader programs (vertex, fragment and geometry shaders) | | Depth testing | | Simple water surface simulation | | Applying shadows.blending (transparency) | | Text rendering | | Applying shadows | | Sound implementation.-text rendering | | Applying shadows | | Cubemaps( skybox ) | | Normal mapping | | Gamma correction | | Reflection and refraction | | Instancing | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### 1. Introduction

The project to develop a game highlighting the concepts taught in the classroom lectures.

From the onset the team had high ambitions to develop eye catching attributes packaged in a game environment. The initial discussion centered on a skimmer based racer but gradually shifted to simpler setup where the emphasis would lay on complex graphics rather than game play mechanics.

### 1.1 Purpose

## 

The objective of the project is to come up with a software that visualizes and test some aspects of life in an entertaining manner like games or animated objects.

**1.2 Problem Definition**

A) Intensive programming effort:

**Suggested solutions:**

1. Use a tool to create objects (blender).

2. Use pre-made object files.

3. Use ASSIMP API to load and store the object info from the object’s file to ASSIMP data structure.

4. Construct a class that extracts the model vertices info from ASSIMP to be rendered properly.

B) Maintainability burden:

The fact that thousands of lines of code are required to render an object properly then it is difficult to edit or get that object modified.

**Suggested solutions:**

1. Use (blender) to create and modify objects.

2. Reuse pre-made libraries as long as it functions efficiently.

C) Performance issues:

Lots of factors might affect the object loading process speed (ex: Hardware capabilities, unorganized code, libraries linking and inefficient APIs).

**Suggested solutions:**

1. Use better hard drive technologies alongside the HDD like SSD or M.2 SSD.

2. Use boosted APIs versions.

3. Prevent using ASSIMP unnecessary flags that might cause a lot of memory allocation and deallocation.

### 2. Overall Description

## 2.1 Operating System Environment

Operating System: Windows 10.

## 2.2 Design and Implementation Constraints

Simple objects loads fast, but complex objects takes time to load for example:

Paris city wave front object (1.788 million vertices)

Object size: 82.4 MB

Textures size: 29.9 MB

Calculated loading time after testing: from 76 to 79 seconds.

Physical memory utilization: 850MB.

We need to reduce this loading time (7x seconds) by at least x5 to be 5 times faster, using APIs or programming tricks or even new hardware tech.

## 2.3Assumptions and Dependencies

The project plan contains 4 stages.

The development approach we will be following in this project will be plan-driven and agile combined together, where come up with a general plan and start implementing a prototype, since the first prototype for sure will not be 100% complete plus satisfying then we will have to keep producing more prototypes with the required updates and modifications until we settle on the final satisfying prototype then we will have to do the finishing and make it up and ready as the final project program.

**Stage 1: window And Interactions**

Description:

the first stage is actually the stage where we set up the environment for us to start the implementation, where we create the project files ,link the required libraries to the project and create needed folders in the project (shaders, objects and texture folders ... etc).

After setting up the environment comes the window initialization and the interactions part where we start creating our program window using the windowing library, and write the functions required to take keyboard input from the user and detect mouse movement for us to be able to move around in the scene.

A stage testing process is to be done as a final step to make sure everything is up and running as it should be, where we draw any primitive object in the scene with a simple testing vertex and fragment shaders (that makes us see if we are actually moving or not and helps in debugging) and check the output of the program.

Stage classes:

1- Shader.h: contains the shader class that creates, link and compile shaders program.

2- Camera.h: contains the camera class that handles the mouse and keyboard inputs and change 3- Camera coordinates to move properly.

4- GlfwWindow.h: contains the main window class that initializes glfw window.

**Stage 2: Scene Components**

Description:

In this stage, we simply choose our objects/scene elements and start loading, setting these objects to have our scene built without applying any effects or animations, the main purpose of this stage is to build the main objects of the scene or at least 70% of it since we will be adding more as we move on.

Stage classes:

- ) mesh.h and model.h: where we extract and draw the object's vertices and textures stored in the assimp library data structure.

**Stage 3: Scene Properties**

Description:

This stage where we do all the fancy/entertaining stuff, applying our rendering skills and animations to specific objects in the scene (ex: skybox, sunlight, shadows, and sound) and then demonstrate some of our tricks using textures, geometry shader and all the skills mentioned in the project description. We have two testing levels, the first level when we test each and every skill functionality and the second level when we are finally done applying all skills we test if they are working properly with each other and make sure that the final output of the project is satisfying, meets the desired requirements and have no errors.

**Stage 4: GUI**

Description:

This is the final stage where we set up our GUI (self-made GUI or AntTweakBar) where we control and change our scene properties, since we are doing a game engine then it is all about controlling the scene properties which make this stage the most important stage in this project.

**Note:** Implementing the GUI should be the last step to avoid chaos, because a lot of changes will occur during the implementation then we will have to change the GUI structure several times, on the other hand, if it is to be done as the last step then we will be able to see the whole picture clear and avoid these (changes) headache.

## 3. External Interface Requirements

## 3.1 Hardware Interfaces

RAM: 8GB DDR3.

GPU: NVIDIA GeForce 840m with 4 GB memory

CPU: Intel Core i7-4510U 2.00GHz

## 3.2 Software Interfaces

Visual studio 2017.

C++ and GLSL (shader programming language).

Using assimp library and visual studio 2017

Modern OpenGL.

Windowing Libraries (GLFW, GLUT, SDL ..etc).

Blender or any alternative application for creating 3D objects.

## 4.1.2 Functional Requirements

1- Group of primitive and none primitive shapes and objects, forming a decent game scene (with respect to relevancy and dimensions ratio).

2- Natural phenomenon (light, shadows, gravity and objects materials ... etc) are required.

3- The user should be able to explore the scene using keyboard and mouse.

4- The user should be able to do changes and modifications to the scene properties using a GUI.

5- The GUI must not be complicated, the user should easily understand how to use it.

6- An acceptable execution time is required (including the object loading time).

7- The objects should load properly without allocation errors.

# 4. Nonfunctional Requirements

# 5. Other Requirements

*Program environment description*

**This part** is the game environment part where we start to render primitives and load objects and put them all together and form a natural based scene, this scene components are ( streets, cars, buildings, characters, trees ... etc ), then we apply all the rendering skills listed above for the sake of (realism), we will be exploring the scene using FPS camera and keyboard inputs to move, (optional idea: related skills or skills under the same category are to be demonstrated inside its own building ex: lighting and shadow building, and another building for particles and water simulations and so on), we will be able to manipulate the scene properties using our GUI like light color or light intensity, adjusting objects positions, enable and disable some properties, rotate or scale objects and it should give us almost full control over the scene without being that complex.

**Part Two** (an extra part highly dependent on the under progress skills):

This part is the intro part where we play a short animated scene for the sake of (entertainment) and also this scene is all about a creative idea or something special about OpenGL.