



## Submission 2 documentation

# Stage Fighter

The Lost Warrior

Raphael Ludwig 01526280

Daniel Fangl 01526097

## Controls

The game does use intuitive controls which are used by most of the first person games available. The character can be navigated around the map with W,A,S,D. To damage enemies the Left Mouse Button is used. The enemies only take damage if the player does touch them.

Key	Effect
Mouse	Look around
W,A,S,D	Movement
Space	Jump
Left Mouse Button	Attack
Right Mouse Button	Block projectiles
ESC	Game Menu
F1	Ingame Help
F2	FPS Counter
F11	Reload Shaders

## Development Status

### Requirements

#### Freely movable camera

With the help of GLM the perspective and view matrices are calculated each time the player does move the mouse and therefore set new values to the yaw and pitch of the camera. These matrices are then send to the shader as P and V and for each object they do get multiplied with the model matrix and the position of the model which produces a freely movable camera. Since the camera is attached to a Bullet rigid body (the player) movements with W,A,S,D are restricted to the rules of the physical bullet world.

#### Moving objects

Each object which is rendered in OpenGL is also represented in Bullet, therefore the moveable objects, which are rigid bodies in bullet, do set their coordinates from the bullet coordinates each render tick. Besides the enemies the projectiles which are shoot from them

## Submission 1 documentation

### Stage Fighter - The Lost Warrior

are also rigid bodies and do fly to a position on which the player stood when they were spawned into the world.

## Texture Mapping

Besides two hardcoded objects (Cube and Triangle) which also have hardcoded texture coordinates all models are loaded from the .glTF files. All these files which are used in the game do contain UV coordinates which are exported by Blender. The texture coordinates are stored into a VBO after the file has been loaded and are used in the shader as `texcoord_0` since that is the name under which the glTF file format does save them.

## Simple lighting and materials

All objects in the current game do have textures and have a material assigned which is imported from the .glTF file. Since the glTF format specifies the material as PBR material some properties of the material are hardcoded into the phong shader.

A single point light source is used which is placed in the middle of the map. This light source does define the ambient, diffuse, specular light color and the power of the light. All normal vectors are taken from the .glTF file since Blender does export these for each model.

## Controls

For the Controls polling and callbacks are used. Inputs like for the character movement which are applied every frame are polled with `glfwGetKey(...)` to ensure that the exact state is captured with the frame. Other inputs like pressing *ESC*, *F1* or *F2* do use the callbacks since it is not important when the key was pressed but that they state change of the key did occur. The Window class does interface with GLFW and does provide functionality for key polling and registering callback listeners which will be executed at the beginning of each frame.

## Basic Gameplay

The goal of the game is to eliminate all opponents, while facing (at the moment not so heavy) fire of them. You have to evade those bullets (or they will slowly eat your health, these bastards), and hit the turrets with your sword (which is at the moment suspect to your imagination). Some controls and a little gameplay do get explained in the "Tutorial" Level.

## Additional Features

### Complex Model loading

With the library “tinyglTF” the .glTF model files are loaded into the game engine when they are needed by a Level. These models can contain complex meshes as for example the Coliseum or Houses which are used through the levels, but also can contain with keyframe animated meshes. Animation is supported by specifying a start frame time and a end frame time. In favor for CPU Particles very complex .glTF files which could contain multiple meshes or even Scene are no longer supported and used as in the first Submission.

### Map loading with Scripting language

The map file and basic entity and object properties are saved as .lua files and can be parsed by the Lua Scripting Engine. Therefore the map can be modified without recompiling the executable. Nearly every object or entity which does exist in game is scripted. These scripts are stored in a separate folder to hide the complexity of such an object from the level file. With the lua function “dofile(…)” these files are included in the map file and can be used. Besides object and entity definitions also AI behavior and particle generation are scripted in lua.

### GUI

The UI in the current game consists of the health and shield indicator in the bottom left corner of the screen, the “Victory!” and “Game Over” message displayed in white letters across the screen when the game ends are rendered with a custom font shader which uses a texture atlas generated from freetype2 glyph data. The game menu, accessible by pressing the Esc key while the game runs, which can be used to alter light settings like ambient, gamma settings and an option to end the game is created with the nuklear library. With the help of nuklear parts of the main screen and the textboxes where implemented.

### Effects

Points	Effect	Status
0,5	Cel Shading	Implemented
0,5	+ Contours (Backfaces)	Implemented
1	Procedural Textures	Implemented (Marble Texture)

## Submission 1 documentation

### Stage Fighter - The Lost Warrior

1	GPU - Particle System (Compute Shader)	Implemented
1	Scripting Language	Implemented
0,5	CPU Particles (Instancing)	Implemented
0,5	Lightmap from different Files	Implemented

- **Cel Shading / Contours:** Both implemented, but hard to spot. Backfaces shadows seem to be disconnected sometimes. Backfaces are implemented via a additional shader, resizing the backfaces and coloring them black.
- **Procedural Textures:** The marble texture is procedurally generated at game startup. Due the high startup time the generation of the texture is happening in a background to avoid various issues with non responding windows. Generated using a perlin noise with matching parameters.
- **GPU Particle System:** Flames and Smoke of Burning objects. The creation of these particles, as already mentioned are scripted in lua. These particles get generated with fixed number and velocity and than are uploaded to the GPU for computation. The corresponding shader does create a geometry from the point list and uses multiple textures to get a nice looking fading effect.
- **Scripting Language:** Lua is used to define the objects, position and their behavior. But also to define the map, and even a list of shaders the objects should be rendered with! Fancy! The lua scripts to load are all mentioned in the index.lua file, then each file is loaded, the elements are then built into c++ lua objects, which then create the final c++ objects the game operates with.
- **CPU Particles:** The projectiles which are spawned by turrets are cpu particles because they get all simulated by bullet and are needed for collisions. Are simulated by bullet.
- **Lightmap from different files:** Is possible, can also be added via a lua script which gives us the possibility to load multiple instances of a object with different lightmaps attached. The lightmap is then used in the fragment shader to create the final image.

## Tools

- **Blender:** Blender was used for the creation, painting and editing of the 3D-Models used by the game.
- **Gimp:** Gimp was used for the creation of simple placeholder graphics up to the editing of the painted textured of blender to the creation of the HUD graphics.
- **CLion / Visual Studio:** Where used as IDEs to write Code, compile & debug the Game. Visual Studio was mostly used for its compiler.

## Libraries

Name	Description	Link
GLFW	Platform-independent API for creating windows	<a href="https://github.com/glfw/glfw">https://github.com/glfw/glfw</a>
GLAD	Multi-Language GL / GLES / EGL / GLX / WGL Loader-Generator	<a href="http://glad.dav1d.de/">http://glad.dav1d.de/</a>
GLM	OpenGL Mathematics Library	<a href="https://github.com/g-truc/glm">https://github.com/g-truc/glm</a>
kaguya	Kaguya is a Lua binding library for C++	<a href="https://github.com/satoren/kaguya">https://github.com/satoren/kaguya</a>
Lua	LUA Scripting Language	<a href="https://github.com/lua/lua">https://github.com/lua/lua</a>
nuklear	Nuklear is a immediate mode graphical user interface toolkit without a renderer	<a href="https://github.com/vurtun/nuklear">https://github.com/vurtun/nuklear</a>
spdlog	Header only C++ Logging Library	<a href="https://github.com/gabime/spdlog">https://github.com/gabime/spdlog</a>
stb_image	Header only Image loading Library	<a href="https://github.com/nothings/stb/">https://github.com/nothings/stb/</a>
tinyglTF	glTF Loading Library	<a href="https://github.com/syoyo/tinyglTF">https://github.com/syoyo/tinyglTF</a>
freetype2	Font Library	<a href="https://www.freetype.org/">https://www.freetype.org/</a>
bullet3	Bullet Physics Library	<a href="https://github.com/bulletphysics/bullet3">https://github.com/bulletphysics/bullet3</a>