

# Parkour Platformer

## Submission 2

### Gameplay

The player can run around in the level and jump over or crouch under obstacles. If the player can keep up a smooth movement (e.g. no abrupt turns or stopping) the movement speed will gradually increase until the maximum movement speed is reached. Turning by more than a threshold angle, falling and crouching will reduce the movement speed again.

The goal is to reach the top of the highest cube.

### Controls

Action	Key	Type
Move Forward	W	Polling
Move Back	S	Polling
Move Left	A	Polling
Move Right	D	Polling
Jump	Spacebar	Polling
Crouching	Left Control	Polling
Exit Game	ESC	Call-back
Toggle Wireframe	F1	Call-back
Toggle V-Sync	F10	Call-back
Toggle Fullscreen	F11	Call-back

# Effects

## Head Up Display

The status of the game is displayed on the Head up Display. It is implemented with the help of <https://learnopengl.com/In-Practice/Text-Rendering> and uses the library “Freetype”.

## Procedural Texture

The level is covered by a procedural cloud texture (1 Effect-Point by Marcus Auer)

Reference: <https://thebookofshaders.com/11/>

## Video Texture

On the first obstacle is a video displayed. It is decoded via FFMPEG and the implementation is based on <https://gist.github.com/rcolinray/7552384>. (1 Effect-Point by Marcus Auer)

## CPU Particle System

Particles are generated and simulated on the CPU and drawn via Instanced Rendering.

Implementation based on

<http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing> (0.5 Effect-Points by Patrick Gantner)

## Physics Engine

Nvidia PhysX is used for the collision detection and gravity.

## Lighting

A directional and point light source are used to illuminate the whole scene. The shadows are generated via shadow mapping. Implementation loosely based on

<http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping> (1.5 Effect-Points by Patrick Gantner). The lightmap is generated on startup for all static geometry. The shadowmap is only used for dynamic objects.

## Adjustable Parameters

Parameters for our Game can be set by changing the settings.ini file in the same folder as the executable. This file is read upon starting the application and follows the default ini schema. For reading the file an external (header only) library is used. The settings are stored in a shared Class which takes care of returning default values if no settings file is loaded or some settings are missing from the settings.ini file.

Short description of implementation for these parameters:

- Resolution Width and Height are read from settings.ini and passed to the glfwCreateWindow function (please note that for Fullscreen currently the monitors default resolution is used. This may change in the future)
- Fullscreen If Fullscreen is enabled the primary monitor is passed to glfwCreateWindow function. Note: If using fullscreen the resolution setting is ignored, and the monitors native resolution is used.
- Refresh-Rate The refresh rate is set by using glfwWindowHint(GLFW\_REFRESH\_RATE, ...) and reading the value from the settings.ini file
- Brightness The brightness value in settings.ini is a factor for the output brightness. A factor of 1.0 will result in the original brightness, whereas a smaller will make the scene darker and a higher value brighter. Currently the brightness factor is simply multiplied to the resulting colour from the texture shader.

## 3rd Party Libraries

- GLFW - <http://www.glfw.org/>
- GLEW - <http://glew.sourceforge.net/>
- GLM – Math library  
<https://glm.g-truc.net/0.9.8/index.html>
- FreeImage – Image loading for textures  
<http://freeimage.sourceforge.net/>
- NVidia PhysX – Physics simulation (Collision detection, movement...)  
<https://github.com/NVIDIAGameWorks/PhysX-3.4>
- Assimp – Asset loading  
<https://github.com/assimp/assimp>
- Simpleini – Reading of settings.ini  
<https://github.com/brofield/simpleini>
- FFMPEG – Decode Video  
<https://ffmpeg.zeranoe.com/builds/>
- FreeType - Text Overlay  
<https://www.freetype.org/>