

# Documentation

## 1. Group/Group Name

Next Destination: Earth

## 2. Brief description of implementation

Our game is an obstacle run. While not trying to fall off the parkour the player has to get to the earth, which is located at the finish line. On his way to earth, he has to overcome multiple obstacles. For example, with some he must try not to run into, others will try to push the player off the course. All this has to be done under a time limit. However, if the player loses a game there's the option to immediately restart the game by pressing the Enter-key.

## 3. Additional Libraries

- OpenGL Helper
  - GLFW: <https://www.glfw.org/download>
  - GLEW: <https://glew.sourceforge.net/>
- Image Loading
  - stb\_image library: <https://github.com/nothings/stb>
- Sound
  - sfml: <https://www.sfml-dev.org/>
- Collision Detection and Physics
  - PhysX: <https://developer.nvidia.com/physx-sdk>
- Model Loader
  - Assimp: <https://assimp-docs.readthedocs.io/en/v5.1.0/>
- 2D Text
  - Freetype: <https://freetype.org/>

## 4. Gameplay

*Mandatory*

3D Geometry

We used the import-library Assimp to import the objects via obj.-files. We created 3D objects ourselves with the use of Blender like the parkour or the asteroids. We also downloaded objects from the the internet (<https://www.cgtrader.com/> ) like the UFO or the rocket. The models we imported can be found in asset\models.

Playable and Advanced Gameplay

- Engaging First-Person Perspective: With a first-person camera view, players can fully immerse themselves in the obstacle run (see "Intuitive camera").
- Intuitive Controls: The game utilizes familiar controls (see "Intuitive controls").

- Clear Objectives: The goal of the game is to complete the obstacle run within a specific time limit (for more see "Win/Lose Condition").
- Varied and Challenging Obstacles: The obstacle run presents a range of challenges, including flying asteroids, moving carpets, slaloms, portals and spinning platforms. These obstacles require players to showcase their reflexes and strategic decision-making to overcome them successfully.
- Our submission can be started via the .exe file

Min. 60 FPS and Frame Rate Independence

Our game fulfills this requirement.

Win/Lose Condition

- The player loses when the time is up, when he falls from the parkour or runs into certain objects (asteroids at the beginning, the gray bar right after the moving platforms and the capsules after the slalom).
- The player wins the game when he completes the parkour within time and reaches the finish line with the two light bars at the end.

Intuitive Controls

For the player to move around, the WASD-keys are used to move forward, to the left/right and to move backwards. In order to jump, the player has to press the spacebar. To turn one's view more to the left or right the according left and right arrow key should be pressed. To exit the game, Esc key can be pressed.

Intuitive Camera

We implemented a first-person-camera since we decided it would fit best to our kind of game.

Illumination Model

We have two point lights (which are also visualized) at the beginning of the parkour and two at the end, indicating where the start and finish of the obstacle run is. Furthermore there are three directional lights. All our objects are illuminated by these light sources (planets, obstacles, parkour, etc.)

Textures

All of our 3D objects have a texture attached to them.

Moving Objects

Multiple objects move around in our scene like the asteroids, the UFO, the rocket and the capsules. The positions of these objects are updated right in the main loop with every frame. The other moving object is the torus/ring around one of the planets (orange one in the middle of the game). For this vertex shader animation is used and further described below.

## Documentation

This document is the documentation.

## Adjustable Parameters

Using a config file (assets\settings.ini) the screen resolution, the fullscreen-mode, the refresh rate and the brightness can be adjusted.

## Optional

### Collision Detection (Basic Physics)

The collision detection can be found when completing the slalom where the player can't move through the walls.

How it was implemented:

- Collision shapes were associated with visual objects using a CollisionComponent structure and an unordered map.
- The createCollisionShape (Main.cpp) function was implemented to create collision shapes for objects and add them to the CollisionComponent.
- The checkCollision function (camera.cpp) performed multiple raycasts and checked for collisions with the environment.
- The dot product between the displacement direction and the surface normal was used to determine collision angles.
- The processKey function in the Camera class was modified to call checkCollision and update the position accordingly for different movement directions.

## 5. Effects

### Animation: Vertex Shader Animation

[https://tuwel.tuwien.ac.at/pluginfile.php/3403666/mod\\_page/content/36/Animation\\_SS18.pdf?time=1619523068902](https://tuwel.tuwien.ac.at/pluginfile.php/3403666/mod_page/content/36/Animation_SS18.pdf?time=1619523068902)

The Vertex Shader animation is used for changing and rotating the rings around one of the planets (the orange one, in the middle of the game). The according vertex shader implements a basic animation by translating the vertex vertically and rotating it around the y-axis based on the sine of the animation time. The transformed position is then multiplied with the model matrix and the view projection matrix to obtain the final position of the vertex. This gives the object (ring) the effect of moving around the planet.

### After feedback:

The "environment.vert" shader implements a wobbling effect on a 3D model (the portals at the end before the finishing line) by modifying the vertex position based on a sine function of the x-coordinate, animation time, and user-defined wobble parameters. The wobble offset is calculated and added to the y-coordinate of each vertex position, creating the desired wobbling motion. The transformed normal is computed to ensure accurate lighting calculations, and the final vertex position is transformed into clip space for rendering.

Texturing: Environment Map

<https://learnopengl.com/Advanced-OpenGL/Cubemaps>

We created a skybox using cubemaps, which provided a realistic backdrop for the scene. Additionally, we applied environmental mapping to oval portal objects, enabling them to reflect the texture of the skybox, creating the illusion of reflection. These portals can be found pretty much at the end of the game and have the additional functionality to teleport the player back to another position in the game, once he goes through it.

Shading: Physically Based Shading

<https://learnopengl.com/PBR/Theory>

<https://learnopengl.com/PBR/Lighting>

For making some of the planets Physically Based Shading was used. The planet before the start line (visible when turning around at the beginning) and the planet on the right side of the earth consist of the following maps: albedo, normal, metallic, roughness and ao. Using these different maps gives the planets a more realistic look.

After feedback:

We added a metallic object (a cube) into our scene right after the asteroids in the first lane of the parkour.

In the fragment shader ("PBR.frag"), several textures are used to define material properties such as albedo, normal, metallic, roughness, ambient occlusion, and height maps. The shader calculates the reflectance at normal incidence based on whether the material is metallic or not and determines the final surface color through the Cook-Torrance BRDF model. It incorporates directional lights and point lights, calculating the contribution of each light source to the surface based on the material's properties and the geometry of the object. The shader also applies height map-based modifications to the albedo and performs HDR tonemapping and gamma correction for the final color output.

## 6. Walk-through

The challenge of the first lane is not to run into the asteroids. If you touch them, you lose. Jump on the metallic cube and back down. Then you have to jump up onto a box and jump down again to get to the next track. There, moving platforms await the player, which transport you off the track if you stand on them too long. You have to jump over the limbo bar in order not to lose. The Galaxy Spiral spins the player in circles. He then has to traverse a slalom between walls. On the next lane, the player must not run into the flying objects. After jumping up and down on a box again, he is almost at the finish. However, if he runs into one of the wobbly portals, he ends up a little further back in the parkour. If not, the two lights on the left and right signal the finish.