

# Dokumentation

## Implementations

### Gameplay

#### 3D Geometry (6 Points)

3D Geometry is either generated at run time or read from a OBJ-File using the OBJ-Loader.

#### Playable (3 Points)

The game consists of a player (the ball) and levels with obstacles.

The player has to maneuver through the level until he reaches the goal, where he will be notified (via text on screen) that the level is over and he can proceed to the next level. If the player touches an obstacle or falls off the edge he will be notified that the level is over (via text on screen) and he has to restart the level.

If the player completes all 5 levels a victory screen appears.

#### Advanced Gameplay (3 Points)

The game consists of collision detection and dynamic objects controlled by our physics engine (Physx) as well as our finalised game logic. Handling the events when a player hits an obstacle / ground / accelerator / falls off the edge / finishes the game / ... .

#### Min. 60 FPS and Frame-rate Independence (3 Points)

We implemented our own time system and tried to consistently include it in animations and physics calculations.

The game runs on ours Systems with over 60 FPS.

We try to manage our resources: Reusing meshes and textures; Removing them from the GPU if they are not needed anymore and so on.

#### Win/Lose Condition (3 Points)

The game has win - lose conditions. The first condition is level specific hitting the finish line of each level means the player won said level, if the player hits an obstacle or falls off the edge the player loses this level and has to try again.

The second condition is game specific where the player can "win" if all 5 levels are completed, where the player will be notified (via text on screen) that all levels are completed and that he finished the game.

#### Intuitive Controls (2 Points)

The Player can steer the ball with A (Left) and D (Right) and can Jump with the Spacebar.

Additionally the mouse is used to interact (e.g. clicking) with certain obstacles.

For this submission we also implemented a "DEBUG" - mode letting the player control the camera freely and also switch through all levels on demand.

#### Debug-Mode:

By pressing the **F4** key you can enter the debug mode where additional information is shown to you, the game is stopped and you can move the camera but not rotate it.

Using **WASD** for planar movement (X and Z) and **SHIFT** and **SPACE** for vertical movement (Y).

### Level-Control:

**Arrow-Left:** Previous Level

**Arrow-Right:** Next Level

### Intuitive Camera (2 Points)

The camera in the game follows the player at a certain distance and stays at the same angle to the player making it easy to understand where the player has to move to proceed.

### Illumination Model (2 Points)

The game has an illumination model. The game has depending on the level 1 or more light sources and every object is rendered via phong shading.

### Textures (2 Points)

Theoretically every object can be created with a texture, however we decided that only the player should have a texture while everything else is kept rather simple design-wise.

### Moving Objects (2 Points)

The game has moving obstacles. (e.g. Level 1)

### Documentation (1 Point)

Right here

### Collision Detection (Basic Physics) (4 Points)

We are using Physx in our game, which handles all the collisions in our game. Noticeable when the player hits any object. (e.g. Ground, Obstacle,...)

### Advanced Physics (6 Points)

We also implemented advanced physics (via Physx) giving every object their own dynamic body to redirect the player if they hit a wall or obstacle.

### Heads-Up Display (4 Points)

We implemented on screen-text rendering using FreeType notifying the player when and how he can interact with an obstacle (when hovered with the mouse) and also show the player the current gamestate (e.g.: GameOver, Level completed)

## Effects

### Shadow Map with PCF (16 Points)

One directional light is being used throughout all levels.

Point lights also work but have no use in our game right now.

### CPU Particle System (8 Points)

Implemented when player dies or lands on ground (see: Effects implementation)

### Hierarchical Animation (4 Points)

We have objects in our game, which have to be swiped using the mouse.

Above them is an arrow, pointing\* and **moving** in the direction to swipe.

\* does not actually work atm, currently always points up but moves in the right direction.

Bloom/Glow (8 Points)

Implemented to highlight obstacles depending on level (see: Effects implementation)

## Features

### Basics

Our game consists of 5 levels, each level consists of the player (the ball) and objects on the screen, some of which are obstacles the player has to dodge others are the ground the player has to move on others are tools for the player to proceed in the level (e.g. ramps on Level 3). The player has one goal during every level which is reaching the end of said level where the player is notified that the level is completed (and the player can proceed to the next level).

### Levels

Each level consists of a variety of obstacles some of them are static others are moving and others have an interaction with the mouse (the player can hover obstacles, interactable obstacles will change color as well as display an info-text on how to interact [or what the interaction does]).

### Goal

Goal is to beat all 5 levels where a "Win"- Screen will be displayed notifying the player that he has beaten the game.

### Debug

For this submission we implemented a debug modus to move the camera / switch through levels. (see Gameplay/Intuitive Controls).

## Effects implementations

### Shadow Map with PCF

Shadow mapping with PCF works by setting up a camera where the light is and rendering the scene from this point of view and then comparing these values in the shader when the scene is rendered as usual. For point lights, because they are omnidirectional, the scene has to be rendered to a cube map. Getting the right parameters for the directional light is a lot of guesswork. Right now it is not perfect but could be adjusted in the near future.

### CPU Particle System

[2] The CPU Particle System implementation can be seen in the Folder (\src\Particle) and the basic structure can be found inside "ParticleSystem.cpp".

Every Particle-System consists of a fixed number of particles (e.g. 200 for the "Game-Over"-Explosion) and each particle contains all the information which are needed to render the particle to a certain position with a certain color.

The Particle System iterates over all these particles to first set their initial parameters and afterwards to update each particle (to make them move/spin/larger/...).

To render the Particle System we iterate again over each particle and write the transformation matrix for the current particle into a VBO, now we only have to call `glDrawElementsInstanced` with the number of particles and all particles with their given transformation will be rendered.

## Hierarchical Animation

We pass the model matrices of the parent objects recursively to the children, which in return multiply them with their own and pass them the shader.

## Bloom/Glow

[1] To implement the Bloom/Glow - Effect we first filtered the scene for specific colors (specifically we created a dot-product between the current pixels color and a filter vector, if the result is large enough it stays on the screen and bloom is applied else it gets filtered out.

see: "assets/shaders/filtler\_frag.glsl")

Then the resulting Texture will be blurred in our case multiple times to increase the blur-effect. (see: "assets/shaders/blur\_frag.glsl").

Finally the Texture is additive blended into the scene (see: "assets/shaders/bloom\_frag.glsl"), we experimented with tone mapping as well as gamma correction but decided the game looked better without them.

## Additional Libraries

Physx for ... physx

OBJ-LOADER[2] for loading Wavefront files,

Freetype for rendering text in an inefficient but simple manner

## Links

Bloom/Glow[1]

Bernhard Steiner: "Post Processing Effects" (05.08.2011), unter:

[https://www.cg.tuwien.ac.at/research/publications/2011/Steiner\\_2011/Steiner\\_2011-Thesis.pdf](https://www.cg.tuwien.ac.at/research/publications/2011/Steiner_2011/Steiner_2011-Thesis.pdf) (abgerufen 17.06.2020)

CPU Particle System[2]

Alexander Heinz, "High-Quality Rendering of Interactive Particle Systems for Real-Time Applications"

(29.4.2019), unter:

<https://www.cg.tuwien.ac.at/research/publications/2019/heinz-2019-psi/heinz-2019-psi-thesis.pdf> (abgerufen 17.06.2020)

OBJ-LOADER[2]

[https://github.com/Bly7/OBJ-Loader/blob/master/Source/OBJ\\_Loader.h](https://github.com/Bly7/OBJ-Loader/blob/master/Source/OBJ_Loader.h)

Used Objects

<https://free3d.com/de/3d-model/skatepark-launchramp-v2--441803.html>

<https://free3d.com/de/3d-model/up-arrow-v1--370749.html>