# ALBA

## Documentation

Maximilian Deutsch 1327587

June 22, 2016

# 1 Gameplay

For every restart, the player is spawned at the beginning of the first level. All levels are generated dynamically with some random parameters. The difficulty increases with each level. Difficulty is defined by the number of platforms, their shape and their behaviour. The first levels have slower and bigger platforms, whereas later levels have faster and smaller ones. The player can move from platform to platform by jumping and moving in a rotating way with the tube's center as the angle point. The player must not touch the surrounding tube, or the game is over, presenting the score of the run. The score is the distance the player reached. A moving shadow is constantly approaching from behind, wrapping everything that it passes with darkness. This forces the player to move quickly.

# 2 Effects

## 2.1 Shadow Maps (with PCF) (1.5) + Omni-Directional (0.5)

Several point light sources are spawned across the levels, with the closest one to the camera casting the shadows. The implementation is based on the *Learn OpenGL Tutorial* [2]. From the light's point of view the scene is rendered into a depth cube map in one render pass. The geometry shader projects each triangle in the 6 different directions for each side of the cube, storing the depth linearly. Afterwards the scene is rendered normally from the camera's point of view, passing the depth cube map to the shader. For each fragment the distance to the light is compared to the distance stored in the depth map in the same direction. If the stored depth is smaller than the fragments depth a shadowfactor is calculated which weakens the light's influence. To smoothen the shadow's edges the shadow values are sampled from the surrounding fragments and a bias for the depth comparison is considered to eliminate some artifacts generated from skewed light directions.

## 2.2 Something similiar to Lens Flare (0.5)

The implemented algorithm for the "lens flare" effect is roughly based on the steps explained in the Article *Lens Flares Effect in Java*[1]. The light causing the "lens flares" is positioned at the far plane from the camera in the direction to the end of the tube. This position is transformed into window space to check if the light is within the camera's view. If the corresponding depth value matches with the depth value from the light (1.0, as it is always on the farplane), it is not covert by anything. This comparison is done with a 3x3 block, where if one pixel passes, the light is visible. The "lens flare" effect itself is a partly transparent texture that is rendered on a quad with its center placed around the lights position.

## 2.3 Depth of Field (1.5)

The algorithm is based on the blog post *Depth of field reloaded*[4], with partly different implementation approaches. Blurring is done on a downsampled texture because of performance reasons. The blurring itself is done using a gaussian filter, implemented in a way described by the article *Efficient Gaussian blur with linear sampling* [3]. A fewer texel fetches are achieved by exploiting interpolated fetches with proper weight factors. The algorithm is done in 5 steps:

1. While rendering the scenen, a blur factor per vertex is calculated from the position, focal distance and focal range, interpolated per fragment and stored in a seperate texture.

2. The color texture of the scene rendering is downsampled to a texture quarter the size, by copying it into a seperate framebuffer with proper size and texture attached.

3. A gaussian filter is applied along the X-Axis of the downsampled texture.

4. A gaussian filter is applied along the Y-Axis of the X blurred texture.

5. The final fragment colors are interpolated between the full resolution texture and the blurred texture based on the blur factor calculated in the first step.

# 3 Objects

Using the Assimp library several meshes that have been created with 3dsMax are loaded at the start of the game. These meshes are the tube and pipe segment, four types of platforms and the rotor. The meshes are shared between multiple instances. All meshes are textured except the rotor, which has a plane white material. The fourth type of the platforms consists of two meshes with one being the child of the other. Both parts have different moving behaviour with the parent passing its transformation to its child representing the ability for hirarchical animation. In fact the transformation of

the whole world (rotation) is passed to all models, making it a hirarchical animation too. The collision objects for bullet are created from low-poly models with Blender and exported as *.bullet*-files.

# 4 View-Frustum-Cullling

For each Mesh a bounding box is calculated from its minima and maxima vertices. When enabled this bounding box is used for view-frustum-culling by transforming them in clipspace and comparing the respective components. A Vertex Count can be displayed to see culling affecting the number of meshes rendered.

# 5 Transparency

The pipe in the middle of the tube has an translucent material. The pipes are rendered after all other opaque models have been drawn for the correct translucency. Also the pipes are not rendered for the shadow mapping from the light's point of view, disabling shadows casted from them.
Additionally partly transparent textures are used for the Text and the Lens Flare.

# 6 OpenGL functionality

- Apart from VAO and VBO, Uniform Buffer Objects (UBO) are used to store data for the the shaders. Multiple ligths are passed to the shader via an array of structs stored in an UBO. The array lenght is fixed and the number of lights is passed as a seperate uniform. For the different effects multiple Framebuffer objects are used.
- the texture quality can be regulated by mip mapping and sampling quality.
- The wire frame mode reveals the objects complexity

# 7 Additional Features

- To visualize the collision shapes from the Bullet library, its Debug Draw is implemented and enabled with F6. The implementation is not optimized and very slow, dropping the framerate enourmously. The shapes are only visible with DoF and Lens Flare disabled.
- With F7 the Coordinate Axes are drawn in front of the camera, with Red X, Green Y, and Blue Z Axis.
- Vertex Normals can be visualized by pressing F10. This is achieved by rendering the scene with a Geometry Shader that draws a line per Vertex in the direction of the Normal.

- To visualize the Bounding boxes used for the viewfrustum culling, press F11.

# 8 assignment of keys

| | | |
|---|---|---|
| | ESC | Quit |
| | F1 | Vertex Count |
| | F2 | Frame Time on/off |
| | F3 | Wire Frame on/off |
| | F4 | Textur-Sampling-Quality: Nearest Neighbor/Bilinear |
| | F5 | Mip Mapping-Quality: Off/Nearest Neighbor/Linear |
| | F6 | Collision Shape drawing: on/off |
| | F7 | Coordinate Axes Drawing: on/off |
| | F8 | Viewfrustum-Culling on/off |
| | F9 | Transparency: on/off |
| | F10 | Vertex normal Visualization: on/off |
| | F11 | Bounding Box drawing: on/off |
| | 1 | Shadows: on/off |
| | 2 | Lens Flare: on/off |
| | 3 | Depth of Field: on/off |
| Y-Button | R | Restart |
| B-Button | P | Pause/Resume |
| X-Button | G | Godmode (no gameover) |
| left stick | W/A/S/D | Movement |
| right stick | Mouse | Look around |
| right trigger | space | jumping |

In the pause mode the left and right triggers are used to move forward and backwards. With Godmode enabled contact with the tube is allowed and the moving shadow is disabled, allowing to freely explore the levels .

# 9 known issues and bugs

- Shadows are not drawn at a certain camera position and viewing direction (facing away from the light source).
- Translsucent objects with DoF enabled are drawn wrongly at some camera positions.
- When the frame rate drops below 30fps, the applied forces for jumping are too high, causing unnormal movement.
- At high velocities, it is possible to get through collision objects.

# References

[1] Luis Cruz. Lens flares effect in java. http://www.emagix.net/academic/item/lens-flares. Accessed: 2016-06-20.

[2] Joey de Vries. Point shadows. http://learnopengl.com/#!Advanced-Lighting/Shadows/Point-Shadows. Accessed: 2016-06-20.

[3] Daniel Rkos. Efficient gaussian blur with linear sampling. http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/. Accessed: 2016-06-20.

[4] Angelo Theodorou. Depth of field reloaded. http://encelo.netsons.org/2008/04/15/depth-of-field-reloaded/. Accessed: 2016-06-20.