

Documentation Submission 2 - Polychromacy

The Game

In our Game "Polychromacy" the Player starts in a cave like- room and needs to fulfill certain tasks to enter the next level and ultimately win the game. A level is completed, when the player manages to adjust the color of the lamps of the room so that it is equal to the crystal object in the center. To achieve that, there are 3 levers, each one representing a primary color that can be changed in intensity. In our case - looking into the room from the default starting position - the left lever represents RED, the one at the back represents GREEN and the right one BLUE, also shown by some light objects positioned right above the levers. Overall the player has 5 lives to solve the 5 levels with 10 possible actions each. One action is one adjustment of a handle. It is possible to get tips to solve the level, though one tip costs the player one life.

Description

Besides having created a Win- Lose- Condition we also added two more levels and made the setup for each level increasingly harder. To implement an advanced gameplay, the player has to move around in a more complex room, move objects or avoid getting hit and do certain tasks first to be able to reach some of the handles. All our models (besides the cube and the sphere) are modelled by hand in Blender and loaded to the game using obj- Objects and the Assimp Importer. Each object has an assigned material, varying from stone to gold (wall and lamps), done by physically based shading, light for the lightobjects and the included materials of the loaded objects. The illumination of the room is done mainly by an invisible pointlight at the center of the room, which illuminates all objects, as well as the several smaller and less impactful point lights at the walls.

For an intuitive movement we used the regular WASD and arrow keyboard input, looking around the room is done by mouse- movement. Furthermore the adjustable parameters can be changed by the assigned keys to which we provide a full list in the step-by-step game instruction at the end of this document.

Features

Removable objects

In some levels, certain tasks have to be done to be able to reach some of the handles. Therefore we have implemented removable objects. In level 2 is an obstacle, blocking the way to the green lever. It can be dissolved by setting the color adjustments first to the obstacle color and will come back if the handles are switched back again. In level 5 is a transparent bridge underneath the blue lever that can be made solid with the correct light settings. The removable objects are implemented using physX, where we add or remove obstacle-geometry to the player's scene.

Collision Detection and Advanced Physics

Sometimes the player needs to jump onto platforms in order to access some of the handles. The jumping, movement and collision detection is implemented using physX. The Camera position is tied to the position of a character controller. The room and its details are all rigid static actors in the scene assigned with hitboxes. Jumping was implemented by moving the player up for a certain amount of time. Every frame the player isn't jumping up, gravity moves the player down. If the position of the player doesn't change, the player is flagged as grounded. Only when the player is grounded can they jump.

There are two kinds of moving objects. One is a dynamic light block that appears in every level and can be moved around and jumped on. Another moving object is a rolling ball that limits the player's movement in the later levels and is able to push the player away. The moving obstacle is implemented as a rigid dynamic body with a kinematic flag, it moves between two points and changes direction when it reaches one of the endpoints. The dynamic light block makes use of a collision callback via the player controller, where when the block is touched a force is applied to it in the direction of the hit. Furthermore we have introduced some pressure plates in level 4 and 5, which react to the player jumping on them.

Heads-up-display

The HUD uses the freetype library to easily read in .ftt fonts and save them as textures. We save the id of the texture, the size, the bearing (position relative to origin) and the advance (the distance between the origin and the origin of the next character). Using these values and a given x and y coordinate we calculate the positions of the quad we will render the text in. We repeat this calculation for every new character, adding in the advance to the x position after every character. The HUD can be turned off by pressing h.

Effects

Vertex Shader Animation

We use vertex shader animation as a special feature in level 5 to simulate the floor covered with water. To do that, we generate a plane geometry according to the room size with adjustable granularity of the mesh. The position's y- value and normals are getting recalculated every frame with a sinus function. To give the water surface a more natural and random look we use Fractal Brownian Motion. In the end the surface is illuminated with the centered pointlight to make the ripple effect starting from the crystal visible.

Specular Map

For the handles we use a specular map to make to golden surface look shinier. Therefore we have exported a regular diffuse map as well as a specular map from Blender. A specular map is only black and white and is used to define the specular parts in an object. The specular map is set as a uniform in the fragment shader and multiplied with the diffuse texture in the color- calculation. This results in brighter spots being highlighted.

PBR

We implemented physically based rendering using different textures for the albedo, roughness, metallic and ambient occlusion. First we use the vertex uv coordinates to get the specific positions in the textures. We also get the normal of the vertex from the corresponding point on the normal map. We calculate the base reflectivity and use it in the FresnelSchlick Approximation to calculate the ratio of light reflected over light refracted. We calculate this ratio for each light. We also calculate distance, attenuation and radiance of each light. To get the final color, we use the Cook-Torrance Reflection Equation, we use the normal distribution function to approximate the amount of microfacets aligned to the halfway vector and the geometry function to approximate the surface area where micro surface-details overshadow each other. The results of the Cook-Torrance Reflection Equation are added up for each individual light.

Bloom & HDR Framebuffer

The Bloom effect was achieved by rendering the scene to two color buffers, one for the actual scene and one for the bright areas to be affected by bloom. The two color buffers are saved in separate textures. In another framebuffer, the brightness texture is blended using a gauss filter. Lastly, the blurred brightness texture and the scene texture are combined and rendered as a texture in the default framebuffer. The used framebuffers all are high dynamic range framebuffers so that lights can be much brighter than (1,1,1).

Normal Mapping

The normal is retrieved from a texture and transformed to the range of $[-1,1]$. Then we calculate the Tangent, Bitangent and Normal Vector for our TBN coordinate system - the Normal is the normal vector stored in the vertex, the Tangent is calculated using the differences in the partial derivatives of uv coordinates and partial derivatives of world coordinates and the Bitangent is calculated as the negative cross product of the other two Vectors. We store them as a 3x3 Matrix and multiply that matrix with the normal vector we received from the texture, then normalize the new vector.

Shadow Mapping

We decided that it would be best if the Shadows only come from the central room light illuminating our scene. Since this light is a point light, we use a cubemap to save the shadows. This cubemap only saves the depths from the viewpoint of the light. Firstly, we define the 6 view matrices, one for every side of the cubemap and multiply them with a perspective projection matrix - we use a perspective matrix instead of an orthogonal one since we are trying to calculate the depths from the viewpoint of the lightsource. For each of the sides of the cubemap, we use the corresponding view-projection matrix to draw every element in the scene. In the Shader, we save the distance between fragment-positions and the light-source in the depthMap. After six passes of this, one for each side of the cube, we then draw the scene as normal, but with the added cubemap in the fragment shaders. The shadow value is calculated in the Function "ShadowCalculation". Here, we first get the direction vector from the light source to the fragment position. Using this vector, we get the corresponding position in the depthMap. To combat the problem of the edges of the Shadows being pixelated, we use a set of Vectors to sample from different positions of the depth Map around the Position that the Fragment actually has. We multiply those Vectors with a value calculated using the position of our player - this makes it so that shadows are sharper when close by and softer when further away. We then take the average of the shadows at the sampled positions as our actual shadow value. For the issue of Shadow

As we use a bias to offset the depth at the current spot - since our values are all in a big range (0-100) we use a rather big bias.

Additional libraries and code references

Simple Shader without lighting for model and classes for model loading (mesh and model class)

https://learnopengl.com/code_viewer_gh.php?code=src/3.model_loading/1.model_loading/1.model_loading.vs

https://learnopengl.com/code_viewer_gh.php?code=src/3.model_loading/1.model_loading/1.model_loading.fs

https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/model.h

https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/mesh.h

Assimp for model loader - <https://www.assimp.org/index.php/downloads>

stb_image

physX - <https://github.com/NVIDIAGameWorks/PhysX>

freetype - <https://www.freetype.org/>

Text Rendering: - <https://learnopengl.com/In-Practice/Text-Rendering>

https://learnopengl.com/code_viewer_gh.php?code=src/7.in_practice/2.text_rendering/text_rendering.cpp

Specular Map

<https://learnopengl.com/Lighting/Lighting-maps>

Bloom

<https://learnopengl.com/Advanced-Lighting/Bloom>

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/7.bloom/bloom.cpp

Physically Based Rendering:

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

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

https://learnopengl.com/code_viewer_gh.php?code=src/6.pbr/1.2.lighting_textured/lighting_textured.cpp

Vertex Shader Animation

https://subscription.packtpub.com/book/game_development/9781849695046/1/ch01lv1sec12/doing-a-ripple-mesh-deformer-using-the-vertex-shader

Fractal Brownian Motion for Water: <https://thebookofshaders.com/13/>

Crystal Texture - <https://www.123freevectors.com/grey-crystal-background-image-125862/>

Blender Rock Brushes

<https://blendswap.com/blend/20195>

Shadow Mapping:

<https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping>

<https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows>

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/3.2.2.point_shadows_soft/point_shadows_soft.cpp

<https://users.cg.tuwien.ac.at/husky/RTR/OmnidirShadows-whyCaps.pdf>

Video Sound

<https://www.bensound.com/royalty-free-music/track/deep-blue>

Instructions to Solve the Game

Level 1

- press ENTER to start game
- go to red handle (left wall) and press 3

Level 2

- go to blue handle (right wall) first and press 3 → removes obstacle
- go to green handle (back wall) and press 3

Level 3

- push light cube to platform to jump on it
- go to green handle (back wall), jump on platform and press 3
- push light cube to right wall, jump on it to reach blue handle and press 2

Level 4

- move up on elevated bridge (use light cube to support jump on first step of stairs)
- go to red handle (left wall) and press 2
- go to green handle (back wall) and press 2

Level 5

- avoid falling down into water (watch for rolling ball)
- go to red handle first (left wall) and press 3 (makes platform at the right side solid)
- go to blue handle (right wall, jump around the column on the front wall to reach it or jump on golden plate at the left wall to get a temporary help crossing that corner) and press 3
- make use of the pressure plates again to go back
- go to green handle (back wall) and press 2
- go to red handle (left wall) and press 1

Keyboard-input/Adjustable Parameters

- move around : WASD or arrow keys
- jump : SPACE
- adjust handle: 1,2,3
- pause / unpause : p
- show HUD / hide HUD : h
- show settings: press p for pause, then c to toggle to settings
- tipp: t
- leave tipp: ENTER
- brightness : brighter = , darker = .
- full screen on / off = F11
- framerate : f
- refresh rate : 8, 9
- toggle shadows on/off: o
- exit game : ESC