Group “Cheese Thief”
Jonathan Eder 12023984
Jonathan Prölß 1912403

Documentation

Group/Game Name: Cheese Thief

Brief description of implementation: The Goal of the game is to push the cheese off the trap without letting the mouse get too close to it. To do that stones that are laying around can be pushed around to get the cheese.

The mouse in the game can be controlled with WASD (movement) space (jump) and your mouse (direction). When you want to reload the level you can press 1, and zoom with the scroll wheel.

Additional libraries:

- PhysX Library from Nvidia
- JSON for Modern C++, https://json.nlohmann.me/

Gameplay:

Mandatory:

- 3D Geometry:
  The mouse, the stones, the trap and the cheese

- Playable:
  The game is playable.

- Min 60 FPS and Framerate Independence:
  It is Frame Independent, movement, physics and the video texture are framerate independent at 60 fps.

- Win/Lose Condition:
  The win condition is to move a stone on the trap, the lose condition is when the mouse is falling off the platform or is moving too close to the trap

- Intuitive controls:
  The Controls are intuitive, WASD, Space and the mouse like in many other games on the computer

- Intuitive Camera:
  The camera is focused on the ‘ingame’ mouse but can be adjusted with zooming, strafing and dragging.

- Illumination model:
  There is one directional light. Every object has a material, changing the parameters shows a different final output. All objects have normal vectors.

- Textures:
  All Objects are textured.

- Moving Objects:
The mouse, the stones and the cheese are movable objects.

- Documentation

- Adjustable Parameters:
There is a settings file in the assets folder to adjust some basic parameters

Optional:

- Collision Detection (Basic Physics):
There Is Collision Detection Implemented as the mouse can push the stones and does not fall through the floor.

- Advanced Physics
As almost all objects (except the snow) are controlled by the physics engine, (and you mentioned in your feedback we could claim for it :)) the game contains advanced physics.

Effects:

Animation:

- Hierarchical Animation:
an Hierarchical animation for the tail of the mouse that swings from left to right the hole time

Texturing:

- Video Texture:
the snowy background is an inverted sphere with a snow video mapped to it, the video (“https://www.vecteezy.com/video/11321105-snow-falling-overlay-loop-black-background-animation”) was converted to single dds frames (and downsized to 480x270 to save storage and reduce loading times) that get loaded at the start of the game, then the texture of the sphere material gets changed to the next frame at a constant 60 fps (the video is 300 frames so at 60fps its 5 seconds long)

Shading:

- Cel Shading:
A simple cel shader was implemented with only one color band. There is one directional light source and the shader uses the material coefficients for ambient and specular lighting and the specular alpha values.

Advanced Modeling:

- CPU Particle System:
  simple particle system that moves with the mouse, it draws instanced Spheres with a white texture but with the same shader as the environment to get rid of the contours
  (the tutorial i used is https://learnopengl.com/In-Practice/2D-Game/Particles)

Post Processing:

- Contours via Edge Detection:
  The Contours are made with a Sobel Edge Detection in a fragment shader. First two framebuffer objects are generated, one for the default scene named FBO, one for the contours named contoursFBO. Due to the contour shader working with normals, there are two color buffers in the framebuffer called FBO. The cel shader writes the color values and the normal values into these two color buffers. The contour shader then performs the edge detection with the normals and writes it in the color buffer of the contoursFBO. Then they are combined in another shader and rendered to the default framebuffer. (https://learnopengl.com/Advanced-OpenGL/Framebuffers, https://learnopengl.com/Advanced-Lighting/Bloom, used the tutorial for bloom, because it's somewhat similar and could not find resources for contours via framebuffers)

Other special features:
The level gets loaded from the levels.json file, you can just edit the json file, to change the levels. The types define which objects to render at the position. (1 for the floor and walls, 2 for the mouse, 3 for the stones, 4 for the trap). Currently the level can only be changed in the settings.ini file.

Walk-through:
When starting the game you will spawn on a map, there are stones laying around. When moving the mouse with the WASD keys, you can push the stones around. To win the level, you need to push one stone on top of the trap, to get to the cheese. If you fail by falling off the platform or pushing all stones off it, you can restart the level by pressing the 1 key.

Group “Cheese Thief”
Jonathan Eder 12023984
Jonathan Prölß 1912403

Cheese Thief Documentation

The Goal of the game is to push the cheese off the trap without letting the mouse get too close to it. To do that stones that are laying around can be pushed around to get the cheese.
The mouse in the game can be controlled with WASD (movement) space (jump) and your mouse (direction). When you want to reload the level you can press 1, and zoom with the scroll wheel.

The Implementation is based on the ECG_UE Project that was given. On the Effect side we Implemented

- an Hirarchical animation for the tail of the mouse that swings from left to right the hole time a Video Texture the snowy background is an inverted sphere with a snow video mapped to it, the video (“https://www.vecteezy.com/video/11321105-snow-falling-overlay-loop-black-background-animation”) was converted to single dds frames (and downsized to 480x270 to save storage and reduce loading times) that get loaded at the start of the game, then the texture of the sphere material gets changed to the next frame at a constant 60 fps (the video is 300 frames so at 60fps its 5 seconds long)
- A simple cel shader was implemented with only one color band. There is one directional light source and the shader uses the material coefficients for ambient and specular lighting and the specular alpha values.
- The Contours are made with a Sobel Edge Detection in a fragment shader. First two framebuffer objects are generated, one for the default scene named FBO, one for the contours named contoursFBO. Due to the contour shader working with normals, there are two color buffers in the framebuffer called FBO. The cel shader writes the color values and the normal values into these two color buffers. The contour shader then performs the edge detection with the normals and writes it in the color buffer of the contoursFBO. Then they are combined in another shader and rendered to the default framebuffer.

On the Gameplay side we implemented:

- 3D Geometry: The Stones, mouse, trap and the cheese
- The game is Playable
- It is Frame Independent, movement, physics and the video texture are framerate independent at 60 fps
- There is a Win/Loose Condition you lose when you fall off the cliff or get too close to the trap, and you win when you place a stone on the trap
- The Controls are Intuitive, WASD,Space and the mouse like in many other games on the computer
- All Objects have an Material assigned and (almost) all objects are lit, except for the environment sphere, because it looks better
- All Objects are textured
- The stones, cheese and the mouse are movable objects
- There is a documentation, you are reding it right now
- There is a settings file in the assets folder to adjust some basic parameters
- There Is Collision Detection Implemented as the mouse can push the stones and does not fall through the floor

Group “Cheese Thief”
Jonathan Eder 12023984
Jonathan Prölß 1912403

Additional Library that were used are:

- PhysiX Library from Nvidia
- JSON for Modern C++, https://json.nlohmann.me/