Documentation

Group/Game Name: Jailbreak Genius

Brief description of implementation:

All the implementation details are listed under their respective bullet point in the Gameplay section. The environment mapping is active at the bars from the window.

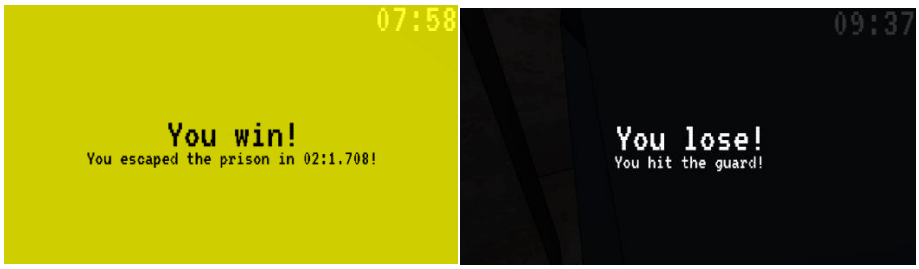
Additional libraries:

- ECG Framework Libraries
- Stb_image (loading images) (https://github.com/nothings/stb/blob/master/stb_image.h)
- Assimp (loading models) (https://github.com/assimp/assimp)
- Nvidia Physx 5.1.3 (https://github.com/NVIDIA-Omniverse/PhysX)
- Freetype (https://freetype.org/)

Gameplay:

Mandatory:

- 3D Geometry:
  - Complex modelling done in blender, exported to fbx file and then imported using assimp. Every model (guard, interior, …) used is self-made.
- Playable:
  - The player can walk around, crouch and jump.
- Advanced Gameplay:
  - Many user interactions, physically accurate collisions/movement. Challenging puzzles. Etc.
- Min 60 FPS and Framerate Independence:
  - On my PC (GTX 1080, i5 8600k), the game runs at 800 FPS with a full screen resolution of 2560x1440. Tested on a laptop the game achieves 60 FPS easily. Usually vsync is turned on, so the game will run at the monitor refresh rate. Framerate is independent by setting the physx simulation to a maximum of 60 times per second. This is done by accumulating the deltaT of the rendering and executing the simulate function only when 1/60 of a second has passed. For lower framerates (e.g., 30), the accumulated time per frame is 2/60, so physx gets simulated 2 times for the 1 frame drawn.
- Win/Lose Condition:
  - You win by escaping the cell, getting by the guard in the corridor to step on the pressure plate, which opens the last door, and then running into the darkness of the now open corridor. You lose either when your time runs out (displayed in the HUD) or when you hit / get hit by the guard.



- Intuitive controls:
  - WASD to move the player, space to jump, ctrl to crouch. E to pick up the glasses (is also displayed as text when looking at them), E to toggle the glasses (also shows in the HUD), 0-9 numbers on the keyboard (or the numpad) to enter the numbers into the door locks.
- Intuitive Camera:
  - Camera moves around with the character controller (at the fixed physics update rate). Looking around it done per frame because it is independent from the character controller. Looking around is “as you would expect” when you move your mouse.
- Illumination model:
  - Every object has material properties which are set in blender loaded by assimp. Normals are also imported from assimp. There are a few point lights in the corridors, as well as one in the starting cell.
- Textures:
  - Textures are saved as a relative file path and read using assimp. The textures are loaded from the assets folder. Basically, every object in the game has a texture assigned to it (e.g., prison walls).
- Moving Objects:
  - The guard patrolling the corridors, and the door opening after entering the correct code.
- Documentation
- Adjustable Parameters:
  - The settings.ini file in the assets can be adjusted. The new settings are loaded when the game next starts, without needing to recompile.

Optional:

- Basic Physics:
  - Physx integration to give walls, floors/ceilings, and other objects collision.
- Advanced Physics:
  - This is done with the physx library. Each object has a physx collider, which is used in physx for simulating. The transform result of these simulations is then used to update the rendered object in opengl every frame. The door is a dynamic rigid body which uses joints, the guard and pressure

plate use callbacks (UserControllerHitReport for testing the pressure plate and guard collisions, and the scene SimulationEventCallback, for when the guard runs into the player)

- Heads-up Display:
  - After rendering the scene onto the plane, depth testing is enabled to render the HUD scenes. The HUD is used to display a crosshair, images for when the user picked up the glasses, and a timer which counts down using text from the freetype library.

Effects:

Advanced Modelling:

- CPU Particle System:
  - When the lock is unlocked, and the door of the prison cells opens a particle system spawns to symbolize a success. The particles are visible as small cubes to fit the chosen aesthetic of the game. This is done using instancing of a cube object. The actual calculations to update positions and sizes etc. happen on the CPU side. To archive the visuals each particles gets a different velocity vector to start with. The color also varies from particle to particle. After a given amount of time to life, the particles get disabled again to not distract from the gameplay.

Animation:

- GPU Vertex Skinning:
  - When importing the fbx file with assimp, the bones from the animation are also loaded into the mesh. The bones need to have a hierarchy, to calculate the total transform matrix for each bone. This is done by recursively multiplying all the model transform matrices of the parents together with the current bone, and then multiplying the result with the offset matrix of the current bone. In the mesh, each vertex can save up to 4 bones which affect it, by storing the bone ids and their corresponding weights (which are normalized). In the vertex shader, the vertices position gets multiplied by the sum of the products of the bone weight and the bone transform. This now allows the mesh to be properly affected by the bones, all that is left to do is to lead the animation we need with assimp and load the animation's model matrix for each frame (and interpolate them if needed) and send the updated data to the vertex shader. https://www.ogldev.org/www/tutorial38/tutorial38.html

Texturing:

- Environment Map:
  - The six sides of the cube map are loaded onto inwards facing faces of a 1 by 1 cube. The cube's vertex shader sets the z coordinate to the w component of the position, which results in the depth of the planes always being 1. After drawing everything on the scene, the skybox is drawn where there hasn't been drawn anything, by setting the depth

testing to less than or equal and drawing it. This only displays the skybox in the background, but it does not affect any objects yet. To make the reflections, the viewing ray of the camera to the fragment is reflected at the normal, and then the texture of the skybox where this reflected ray hits is added to the material, causing reflections.
https://learnopengl.com/Advanced-OpenGL/Cubemaps

Shading:

- Cel Shading:
  - The fragment color (after calculating everything like lights) is first converted into HSV, where then the V value is discretisized into a configurable number of values. These values are then converted back to RGB and set to be the new fragment color.
    https://tuwel.tuwien.ac.at/pluginfile.php/3403666/mod_page/content/36/CelShading_SS19.pdf

Post Processing:

- Contours via Edge Detection:
  - The scene is rendered into a framebuffer. In the fragmentshader, instead of just outputting the color, the shader also outputs the normals, which are calculated anyway. This normal color attachment is then convoluted with a laplace kernel, which detects the edges. This rendered image with the contours is then rendered onto a plane which covers the whole screen.
    https://tuwel.tuwien.ac.at/pluginfile.php/3403666/mod_page/content/36/CelShading_SS19.pdf

Other special features:

Walk-through:

The glasses can be picked up by crouching (holding ctrl) and walking close to them. When you look at them from the correct distance (1.25m maximum), a help text will appear and invite you to click E to pick them up. Once picked up, the HUD displays that you have them, and they can be toggled by clicking E.

The locks are color-coded to specific puzzles. The code for the green lock can be obtained by stepping on the bed and looking at the bars from a specific angle to see the green number (9). The code for the red lock can be obtained by scanning the red QR code (8). The cyan code can be obtained by getting the glasses and looking at the QR code with the glasses on, to reveal the hidden number, which is cyan (4). The last lock is half red and half cyan, so you need to take half the number of the red lock and half the number of the can lock and add them together (6).


09:24


05:06
e


„The secret number you are looking for is eight“ suchen