

Prototype Document

Urine Trouble, Little Fly

Aschl Bernhard (1529509)

Böswirth Klemens (1429367)

“Urine Trouble, Little Fly” is a two-player action game. One of the players controls the fly trying to survive. The other player controls the jet of liquid and tries to hit the fly. The game is over when the fly has no life left or there is no liquid left.

Controls

Key	Effect
Mouse-Movement	Control jet direction (except in FreeCam)
Arrow Keys	Control fly's walking direction
F2	Toggle FPS-Display
F3	Toggle Wire-Frame
F4	Toggle HUD
F5	Toggle Normal-Mapping
F8	Toggle View-Frustum Culling
F9	Toggle DebugCam
F10	Toggle DebugLight
F11	Toggle Fullscreen
ESC	Menu/quit the game
PageUp/Down	Alter brightness-modifier
FreeCam Controls	
Mouse-Click	Catch cursor
Mouse-Movement	Change view direction
WASD	Move camera

Effects

- ✓ Physics Engine (mandatory)
- ✓ Heads-Up-Display (mandatory)
- ✓ Lightmap (mandatory) in separate textures [0.5] and calculated in game [1.0]
- ✓ Blobby Objects [1.5]
- ✓ Simple normal mapping [0.5]
- ✓ GPU-Particle System [1.0]

Features

- Model (+Material) loading via AssImp
- Collision Detection via Physx (Urine-Balls, Urinal, Fly)
- Baked complex model of urinal for Physx
- Texture loading via FreeImage
- Automatic saving of window settings (Fullscreen/Resolution)
- Window resizing on the fly
- Immersive Bathroom

Implementation

Our code is structured in a maincontroller holding all the relevant information. It holds references to the scene graph, window, camera(s), shader, important gameobjects and modifiers. Furthermore it handles the gameloop, keycallbacks and so on.

Camera

There is a Camera supertype, which holds a projection and a viewmatrix. The FreeCam-Class is derived from the Camera-Class. In addition to the cameras attributes It holds variables for turning and orientation.

The DebugCams variables are altered in the Maincontroller. The other Cameras are only moved by their parent objects.

Moving Objects

Movement is mainly handled via Physx.

There are 2 types of moving Physx objects in the game. The balls of urine, which are given a velocity in the viewing direction when spawned and the Fly which is a player-controller.

Furthermore the penis can be rotated via user controls.

Texture Mapping

Our game-objects are loaded with AssImp. Some of those objects have paths to textures on them. These are loaded with FreeImage and then loaded to the graphics card. Loaded objects contain uv-coordinates which are assigned to the Objects and when drawn are

loaded to the graphics card, interpolated in the vertex shader and then the color is determined by the texture and the interpolated coordinates in the fragment shader.

Lighting and materials

Material attributes are loaded with AssImp. Currently there are 3 types of lights implemented, but only 1 is used.

A Phong shader is used. The fragment shader gets interpolated normals. It iterates over the lights in the scene and uses the normals (and normal maps) and the material properties to calculate the ambient, diffuse and specular portion of the light and adds them all together to get the intensity of the fragment. Said properties are coefficients of the different kinds of light and the shininess of the material.

Normalmaps

Normalmaps are loaded into the game and into the shader like normal textures.

Before using normalmaps a matrix for transforming the normals in the map from tangent space to model space has to be calculated. The matrix consists of the vertex' normal, tangent and bitangent and those are calculated on object load.

In the fragment shader the normal from the normalmap is multiplied by 2 and 1 is subtracted, to map the normalmap's value (zero to one) to real normals' values (-1 to 1). The resulting normal is then transformed to model space by the matrix mentioned above.

Source used:

<http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/>

Dynamic lightmaps

At start-up lightmaps are created. For every static object in the game a framebuffer is generated and a texture applied.

The vertex shader of the lightmapper uses the uv-coordinates for `gl_position` and sends the vertex' position in world coordinates to the fragment shader. The fragment shader calculates the diffuse portion of the light and sets the output color to the calculated value.

Source used: https://www.joshbeam.com/articles/dynamic_lightmaps_in_opengl/

The edges of the lighted areas of the resulting texture are black resulting in black lines on the object. Because of this a dillute filter has to be applied to the texture. A similar shader to the lightmapper calculates the mean of the pixels around a pixel in the texture ignoring black pixels. This way the border-pixels are colored.

View Frustum Culling

At creation of an object an oriented bounding box is created using the maximum and minimum vertex of the object in relation to 2 axis.

Before drawing a Frustum View Box is created using the parameters of the camera consisting of 6 planes. Every object is then checked for each plane if the relevant point (the point leastlikely to be beyond the plane) is beyond the plane and therefore outside the box.

Blobby Objects

To render the urine the effect of Blobby Objects (also called Metaballs) is used. The points used in the Metaball-calculation are simulated using PhysX spheres. The metaball-effect is created during rendering in the fragment shader using ray casting.

The vertex shader used for the Blobby Object is called with the axis-aligned bounding box of the metaball-points (extended by the maximum radius of a blobby sphere). The corners of the AABB are converted to clip-space using the Model-View-Projection matrix and passed on to the fragment shader. There the actual ray casting happens. Using the fragment coordinates and the near-plane rays are created and intersected with the AABB. If a ray intersects the bounding box, the intersection segment is checked using ray-marching. Starting with the entry point the segment is stepped through using a fixed step-size until the metaball-density-value exceeds the isolevel threshold or the end of the intersection segment is reached. Fragments for which the ray hits the blobby object are shaded as always, others are discarded.

Metaballs: 1) <http://www.geisswerks.com/ryan/BLOBS/blobs.html>, 2) <https://en.wikipedia.org/wiki/Metaballs>

Ray Casting: <http://prideout.net/blog/?p=64>

Particles

The particles used to render small drops of urine, spawned randomly when the blobby-object urine hits something. Particles positions and velocities are calculated using a Compute Shader similar to the example shown in the repetitorium presentation.

Controls

Camera controls and player controls are handled in the update of the gameloop. It polls the status of relevant keys and the adjusts the corresponding variables with regard to deltaTime (time since last frame).

Adjustable Parameters

There is a settings.ini in the assets. It can be changed manually and also indirectly by changing the size of the window or toggling fullscreen.

Illumination and Textures

Every object in the scene has a material. Fly, towel, walls and floor are textured.

There are 2 pointlights in the scene. One at the lamp on the ceiling and one above the sink. They illuminate all the gameobjects (walls, urinal, fly, urine-balls, penis, wc, sink,...)

Additional Libraries

- FreeImage - <http://freeimage.sourceforge.net/>
- Physx - <https://developer.nvidia.com/>
- AssImp - <http://www.assimp.org/>
- glew - <http://glew.sourceforge.net/>
- glfw - <http://www.glfw.org/>
- glm - <https://glm.g-truc.net/0.9.8/index.html>
- freetype - <https://www.freetype.org/>
- loguru - <https://github.com/emilk/loguru>

3D Models

Simple models as the rug in front of the toilet, the walls, the toilet paper rolls, the magazine and the lamp on the ceiling were modeled in blender. The rest of them is from the Internet.