

Space Sweeper

Sebastian Schrammel, 1526197

Hana Salihodzic, 1320206

Description of the implementation:

Freely movable camera

All transformations from model to perspective are handled by the camera class. The player-controlled spaceship automatically moves forward and the camera is attached to it. The camera thus follows each movement of the ship which results in a third person view.



Game Objects

We use Assimp to load the models that we created in blender (spaceship, asteroids, sun, trashcans, path rings) and store them using our model class, where each model is composed of all its meshes and textures. Using the collada format to export the created level allows us to retain mesh references, so that identical meshes, such as asteroids, don't have to be loaded multiple times.

All objects that have 3D coordinates (such as Model or Camera) inherit from the class SceneObject, which handles all 3D positions, orientations and movements.

Texture Mapping

We also load the texture coordinates and vertex normals from the 3D objects. For texture loading itself we use stb_image. If the same texture is used multiple times (e.g. for multiple asteroids) it is only loaded once.

Simple lighting and materials

We implemented a simple phong lighting model in our fragment shader. For each object we defined material constants (ambient, diffuse and specular) that describe how a mesh should be shaded. We have a single centered light source at the sun's position that illuminates the level.

Controls

The spaceship can be controlled with the four arrow keys or WASD. Forward movement happens automatically. The F9 key can be used to toggle Debug mode, in which the camera can be moved independently from the spaceship using mouse dragging and W/A/S/D or the arrow keys.

- W/UP: Pitch down
- S/DOWN: Pitch up
- A/LEFT: Roll left
- D/RIGHT: Roll right
- Q: Yaw left
- E: Yaw right
- P: Pause/unpause
- F5: Toggle Bloom
- F6: Toggle subdivision level
- F7: Toggle normal mapping
- F8: Toggle HUD
- F9: Toggle debug mode
- F10: Toggle wireframe mode

Basic Gameplay

The player can maneuver the ship through the demo level that is available. The goal is to finish the track which is designated by rings. The rings must be passed in the order that they appear in. On the way, collectibles (trash cans) can be picked up to reach a higher score. The state of the game is printed as the window title in windowed mode. If the spaceship collides with an obstacle, the game is over. As soon as all rings have been passed, the game is finished. The goal is to collect as many trash cans as possible.

When the player reaches the end of the track, they are awarded points depending on the number of trash cans collected on the way, and the time it took them to finish the track.

Features of the game

The game offers intuitive 3D flight controls and a somewhat realistic space atmosphere. Smooth motion is taken care of by the Physics engine.

How and which objects are illuminated/textured?

All objects are textured and illuminated, most of them with the complete phong model. However, certain objects such as the sun and background skybox are only illuminated with ambient light, which is achieved by setting the respective material constants. All objects have a diffuse texture that provides color data. For this purpose we implemented two shaders, one for the phong-shaded models in the scene and one specifically for the skybox.

Adjustable parameters

Certain parameters can be adjusted via a .ini file, which can be found in bin/res/config. These are FOV, brightness, draw distance, framerate cap, fullscreen mode, screen resolution and blur iterations for bloom.

Physics engine

Nvidia PhysX handles collision detection and player movement. The physics simulation uses a fixed timestep. Obstacles are modelled as RigidStatics, the spaceship is a RigidDynamic, collectibles and path rings use trigger shapes.

HUD Overlay

We implemented a simple HUD to show the current game time and number of trash collected. This text rendering utility is also used to notify the player when the game has ended.

Effects

- **Bloom**

To improve the atmosphere and make the path rings more visible, we integrated bloom into our rendering pipeline. The main shader stage writes its output to a framebuffer, which is then blurred around the brighter areas. After that, the blurred bright spots are blended over the existing scene, leading to a light-bleed effect around bright areas. The number of iterations of blur can be adjusted via the respective parameter in the .ini config file.

- **Subdivision Surfaces**

Since the geometric detail of the asteroid base meshes that are loaded from our blender interface is kept fairly basic (in order to keep vertex count acceptable), we implemented a subdivision algorithm. The three subdivision levels are computed while the game loads and can be toggled seamlessly while the game is running.

- **Normal Mapping**

To enhance the details of the asteroids, we have implemented normal mapping. A TBN matrix is used to transform normal vectors read from an RGB normal texture, and apply those to the asteroid models on a per-fragment basis. The tangents needed for this process are calculated manually and passed to the shaders as vertex attributes.

Resources

- Model loading: assimp: <http://www.assimp.org/>
- Image loading: stb_image: https://github.com/nothings/stb/blob/master/stb_image.h
- ini reading: ini.h: <https://github.com/benhoyt/inih>
- Subdivision Surfaces: http://www.pbr-book.org/3ed-2018/Shapes/Subdivision_Surfaces.html
- Tangent calculation for normal mapping: <http://ogldev.atspace.co.uk/www/tutorial26/tutorial26.html>
- Font loading: FreeType: <https://www.freetype.org/>
- Font: <https://www.1001fonts.com/>
- Textures: Total textures repository
- Other inspiration: <https://learnopengl.com/>