Super Space Kitten (v1.0.1)

Siegfried Reinwald (1126981)
Michael Riesner (0929129)

1 Implementation of Requirements:

1.1 Required Features

1.1.1 Camera:

There are different cameras used through the game.

A free movable camera for cinematic sequences.

A chase camera used for following the player (the player's movements). Cockpit-View / Chase-View is implemented with this camera.

A object camera that sticks to an object and surrounds it. Used for rendering the current enemy on the HUD.

These are implemented via the interface Epifx3D::Scene::Camera::Camera

1.1.2 Moving Objects:

All objects in the space-sector can move around freely. Asteroids and Ice-Crystals rotate around their origin. Enemies have a basic AI which lets them fly to the player and shoot at him. Enemy pilots even try to evade collisions.
The player can move around freely too.

1.1.3 Texture Mapping

Texture Loading is done via the SDL_image[1] library. After that the texture gets stored by MemoryManager::TextureManager and uploaded to the GPU. The MemoryManager prevents loading the same texture twice.

1.1.4 Lighting and Materials

The Lighting and Material implementation is based on the tutorials of Megabyte-Softworks [2] and the wikibook OpenGL Programming[3]
Supported light-sources are directional lights, spot-lights and point-lights. Every light-source has an ambient, a diffuse and a specular part, as well as a Specular-Intensity. Specular lighting is pixel-based.
Curved-Surfaces are working.
Every Sector-Object has a material, which also has an ambient, a diffuse and a specular part.

1 https://www.libsdl.org/projects/SDL_image/ (04.05.2015)
2 http://www.mbsoftworks.sk/index.php?page=tutorials (04.05.2015)
3 http://en.wikibooks.org/wiki/OpenGL_Programming (04.05.2015)

1.1.5 Controls

Input handling is done with the SDL Library[4]. Keyboard, Joysticks and Gamepads are supported. But keep in mind, that the game is optimized to run with a joystick with integrated throttle-control.

Ship-Movement:
==========

+	Increase Throttle
-	Decrease Throttle
TAB	Full Throttle
BACKSPACE	No Throttle

Cursor-Keys	Steer (Yaw/Pitch)
Space	Shoot Tachyon Pulse Cannon
CTRL	Shoot Missile

T	Mark Object in the crosshair

Special Keys:
==========

F1: Show On-Screen Help
F2: Show FPS
F3: Render the scene in wireframe mode
F4: Cycle through texture magnification filters
F5: Cycle through texture minification filters
F6: Show On-Screen Debug-Information
F7: Change the Player Camera (Chase-Cam / Cockpit-Cam)
F8: Turn frustum-culling on/off
F9: Cycle through Bloom Implementations (No Bloom / Bloom + LDR / Bloom + HDR)
F10: Turn Glow on/off
F11: Turn Afterglow on/off (Glow has to be activated for afterglow to work)
F12: Turn Normal-Mapping on/off (Have a closer look at an asteroid and to see the effect)

1.1.6 Fonts

The Font-Loader uses the Freetype2-Library[5]
Fonts are implemented using an atlas for the glyphs. The glyphs are loaded in 5 different sizes and uploaded to GPU memory.

1.1.7 Model Loading

The mesh-data is loaded using the Assimp-Library[6]. Vertices, Indices and Normals are loaded into a mesh object. The data is uploaded to the GPU memory. MemoryManager::MeshManager ensures, that no mesh is loaded twice.

1.1.8 Sound / Music

Sound / Music is played via the SDL_mixer Library[7]. MemoryManager::SoundManager and MusicManager ensures, that no sound or music will be loaded twice. The Sound/Music-Player is implemented as a static class, so it is possible to change music / play sounds from every part of the

4 https://www.libsdl.org/
5 http://www.freetype.org/
6 http://assimp.sourceforge.net/
7 https://www.libsdl.org/projects/SDL_mixer/

game.

1.1.9 Additional Libraries

The configuration file for the game is stored in XML format which is parsed with the RapidXML Parser[8]

GLEW[9] is used to load OpenGL extensions.

1.2 Graphics-Effects

1.2.1 GPU Particle Systems

When an object explodes it spawns an explosion which is implemented as a GPU particle system using an approach similar to the approach of MBSoftworks[10]
The particle-data is created and updated on the GPU side using a transform-feedback and two buffer objects. The Exhaust-Rays from the ships are also implemented as particle-systems.

1.2.2 Spotlights & Shadow-Mapping

Every spotlight in the scene supports shadow-mapping. The spotlight as well as the shadow mapping can be seen at the gun turrets (One arrives with the second wave of enemies). The Shadow-Maps use PCF and a resolution of 512x512 pixels.

1.2.3. Bloom (Activate / Deactivate with F9)

Bloom is implemented as it was shown on the slides of the ‚Repetitorium'. We use a surface that is 1/8 the size of the screen for the Gaussian Blur. After that the luminance values of pass 1 are added to the Screen-Texture. Adding the color values didn't look that good in our space environment. That's why we use ‚white bloom'. Have a close look at an ice-crystal (gray points on the radar) when bloom is activated to see a slightly exaggerated bloom effect.

1.2.4. HDR (Activate / Deactivate with F9)

If HDR is activated, the scene gets drawn to a half-float surface which doesn't clip the color values between [0,1]. The Tone mapping algorithm used to map the values to LDR is the same as in the paper „High Dynamic Range Rendering in OpenGL" by the authors Fabien Houlmann and Stephane Metz[11]. It uses the maximum luminance value of the picture to change the illumination adaptively. The maximum luminance is calculated by scaling down the surface (like mipmaps) while preserving the maximum luminance value. If HDR is activated, then bloom is automatically activated.

1.2.5 Glow (Activate / Deactivate with F10)

Glow is implemented like bloom. But instead of using a threshold we use the alpha value of the pixels for the Gaussian blur.

1.2.6 Afterglow (Activate / Deactivate with F11)

If afterglow is activated, not only the glow pass1-texture from the current frame is used for glow, but the glow pass1-texture from the last frame as well.

---

8 http://rapidxml.sourceforge.net/
9 http://glew.sourceforge.net/
10 http://www.mbsoftworks.sk/index.php?page=tutorials&series=1&tutorial=26
11 https://transporter-game.googlecode.com/files/HDRRenderingInOpenGL.pdf

1.2.7 Normal-Mapping (Activate / Deactivate with F12)

Normal-Mapping uses tangent space to calculate the normals of the objects from a normal-map. The Asteroids use heavy amounts of normal mapping. The ice-crystals also, but not as much as the asteroids.

1.2.8. Hierarchical Animations

We've implemented hierarchical animations. They can be seen at the gun turrets (which arrive with the second wave of enemies). The Gun Turrets consist of 2 Meshes (Plane and Turret), where the turret can rotate independently of the plane (Turret is a child of the plane)

1.3 Other Effects

1.3.1 Joystick / Gamepad support

Joystick / Gamepad support was implemented by using the SDL_Joystick functions. We recommend using a joystick to play the game, because it was optimized for a joystick with throttle control. Using a gamepad is possible, but the Triggers won't work to set the speed of the ship. You'll have to use buttons for that. Full Gamepad support will be implemented in a future version ;)

1.3.2 Force Feedback Support

If you use a force-feedback enabled device, you can activate Force Feedback in the Launcher.exe application.

2 Features

- Galaxy consisting of a space-sector (multiple sectors can be read from XML Files)
- Collision-detection using bounding-spheres
- Enemy-Ships that attack the player and try to evade collisions
- Static-Objects (Asteroids, Ice-Crystals)
- Multiple Weapons (Missiles and Energy-Weapons)
- Realistic Space-Flight-Avionics Simulation (Radar, Status-Monitors, Hud)
- XML Reader to read the galaxy-definition from XML Files

2.1 Illumination

All objects in the space-sector are illuminated by one or more sun objects. A sun holds a directional light-source that is considered by the shader. Additionally Spotlights can be used. Up to 8 Light-Sources per sector are possible. As well as 8 shadow-maps per object.

2.2 Textures

All objects in the space-sector are textured. If a mesh has no texture, a default texture (white.png) will be used for the mesh.

2.3 Launcher / Configuration File

Screen-Resolution, Refresh-Rate and Fullscreen (yes/no) can be set in the Launcher.exe Application. In that application you can as well set your preferred input device. We recommend using a joystick with throttle control. Gamepads are supported, but the game is mainly optimized for joysticks.

2.4 Space-Flight-Avionics

We've implemented a system that allows writing avionics-simulations for the ships. We've implemented a simple radar as a proof of concept, but much more complex implementations

are possible. The HUD then uses the data from the radar to draw the red marks around the enemies.

2.5 XML Reader

The XML Reader was implemented to parse the configuration file for the 3D Engine as well as for parsing the galaxy-definition(Ships, Weapons, Factions, etc...) from XML files. The implementation isn’t complete, but works for the config-file, factions and space-sectors.

3 Future Work

There are many things in the game that could be extended in the future:

*) Implement more mission types. Currently there is only 1 mission type implemented. (Kill all enemy waves)
Future mission types could include Escort Missions, Reconaissance Missions, Capital Ship Attacks etc.

*) Implement a LUA Stack and read AI-Definitions as well as Mission Definitions from LUA Scripts.

*) Improve Vessel implementation, so that capital vessels can have different modules (Bridge, Engines, Shield-Generators, etc.) and more than one weapon as well as built-in weapon turrets.

*) Drop implementation of different Vessels and different Weapons and define them solely in XML-Files

*) Implement a full featured GUI. The current implementation is very minimalistic

*) Improve the user feedback. For example the player-vessel could shake when it gets hit, Missiles could have Exhaust-Rays to make them better visible.

*) Improve the Cockpit View. Design a nice cockpit and integrate the monitors into it.

*) Add additional effects. Anti-Aliasing, SSAO, LOD, etc...