Savey Spacey Kami & Ka(t)ze

CGUE 2018 Submission 2
Max Dorfmann (00926889) & Valerie Riegler (01326794)

.. is a space action game where the player has to guide a spacecraft through a tunnel full of asteroid obstacles.

Technical Description

Begin to play the game by double clicking on Savey_Spacey_Kami_Ka(t)ze.exe.

Adjustable parameters (as in Submission 1)
You can set the screen resolution (*width* & *height* in px), fullscreen-mode (*fullscreen* [true,false]), refresh-rate (*targetfrefreshrate* in frames per second) and brightness (*brightnessmultiplier* as float value) in assets/settings.in before starting the game.

Gameplay
The player has to maneuver a spacecraft through a tunnel full of asteroids in order to save the cats that are trapped in a billboard on a mystical planet by the other end of the tunnel. But careful, if the spacecraft collides with the asteroids, they would explode and shower the spacecraft in scary confetti particles :(
The spacecraft can only survive the confetti showers 5 times.

Controls / Debug Options
A, D - to move the spacecraft left or right
W, S - to move the spacecraft forward or backward
mouse - to control the orientation (yaw & pitch) of the camera / player
O - switch to freely movable debug camera (without following the spacecraft, no collisions)
P - switch to play mode with momentum in z-direction (intended for playing)
I - switch to play mode without momentum (default at start, no stress when testing e.g confetti explosions)
(F1 - help if available - not needed here)
F2 - Frame Time / Performance Output on/off
F3 - Wire Frame on/off
F4 - HUD on/off
F5 - Simple normal mapping on/off
F8 - View-frustum Culling on/off
Esc - to quit the game

Freely movable camera
Using the controls for moving the spacecraft (WASD and mouse) the player can freely move the spacecraft (followed by the camera) in “I”-mode or the debug camera (“O”-mode on its own) in all directions. The play mode (“P”) restricts the freedom of movement by adding a momentum in z-direction (forwards).

Complex 3D Models from Files

We used the assimp-library to import complex 3D models from .obj files with their corresponding texture in .dds files following the tutorials listed in our References.
We obtained the free 3D models from:

Meteor: https://www.turbosquid.com/3d-models/obj-free/927712
Spaceship: https://www.turbosquid.com/3d-models/free-space---3d-model/531813

Moving objects / Hierarchical animation using PhysX

The spacecraft moves controlled by the player with the camera following it (attached as a child with an offset position of the spacecraft).
Some of the asteroids move automatically back and forth to make the gameplay harder. When they collide with each other they would bounce apart depending on the direction of the impact momentum. The mystical planet and the billboard spin hierarchically together.
Movement and collision detection is handled by the PhysX-API.

View Frustum Culling

We implemented simple view frustum culling using the geometric approach. The implementation can be found Frustum.cpp. We use bounding spheres around each object. Before the rendering of our nodes, our program checks whether the bounding sphere of an object is either completely inside the view frustum or intersects with it. If so, the object is getting rendered.

Texture mapping (as in Submission 1)

All objects in our game are either created using the provided GeometryData-methods in *Geometry.cpp* or are imported using the assimp-library (see Objectloader.cpp for implementation details). In both cases the geometric shapes are created with respective texture coordinates. Furthermore we use the *Texture* class to load and attach textures to our objects.

Lighting and materials

We use the provided light implementation from the ECG-Framework. We only have one point-light-source which is positioned on x=-500,y=0,z=0 and should simulate a nearby star.
Each object has a material as well as normal vectors, which are either created in the Geometry class or are directly imported using assimp.

Effects

(Compulsory: Light Mapping)

in our first feedback we were advised to choose a different effect worth 1 point instead of light mapping as there are not many static objects in our scene. So our total of effect points is 5.

Compulsory: Heads-Up-Display

for our HUD we used this tutorial https://learnopengl.com/In-Practice/Text-Rendering We included a free font called Atarian https://www.dafont.com/atarian-system.font and added a second font to illustrate our hearts https://www.dafont.com/heart-shapes.font. The HUD showes the remaining lives (hearts) and a message of what the goal of the game is.

Compulsory: Physics engine

The Physics engine has already been included for Submission 1 and is used for movement and collision detection of the tunnel, spacecraft and asteroids. It was implemented following this manual https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Index.html.

1 P GPU-Particle System (+Compute Shader, Instancing)

The particle system is used to illustrate asteroids that burst into confetti particles when destroyed. We used the tutorials provided by the LVA for the Compute Shader and the Vert → Geom → Frag - Shaders that draw the quads. (https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/ComputeShader_SS18.pdf and https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/GPU_Particles_SS18.pdf) Instancing was added as described in https://learnopengl.com/Advanced-OpenGL/Instancing

1 P Procedural Textures

We applied a procedural texture on the 'mystical planet' on the other side of the tunnel. The planet texture is randomly generated at runtime. Basically we use a simple brownian motion approach using perlin noise (see https://github.com/sol-prog/Perlin_Noise/blob/master/PerlinNoise.cpp). The implementation is a potpourri of several sources like https://thebookofshaders.com/13/, http://flafla2.github.io/2014/08/09/perlinnoise.html or https://lodev.org/cgtutor/randomnoise.html, to name a few.

1 P Video Textures

On the above-mentioned 'mystical planet' there is a giant billboard that shows our trapped cats in this video: https://www.youtube.com/watch?v=I7UwSVsiwzI (it was also inspiration for our game). The video is getting processed using ffmpeg/libav (https://ffmpeg.zeranoe.com/builds/). Furthermore we ported a wrapper found on github (https://github.com/datenwolf/aveasy) to work with a newer version of ffmpeg/libav. The wrapper facilitates the reading of single consecutive video frames, which then are used as texture-images.

1 P (Normal) Normal Mapping

As our objects are not planar, and normal mapping therefore becomes more complex because of the calculation of the normals in object space, we were offered 1P instead of 0.5P for this effect. We had to introduce two further VBO's per object - the so called tangents and bitangents - which then can be used to calculate the normals in object space. Tangents and bitangents are either generated by the objectloader, or - in case of primitive objects created at runtime - are calculated during the creation of a Geometry-object. The implementation was inspired by

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/ and https://learnopengl.com/Advanced-Lighting/Normal-Mapping. The effect can be seen on the asteroids and the space ship as they received realistically shaped indentations.

0.5 P **Cel Shading**

Cel shading was used on the spaceship and the meteors and was implemented following https://www.youtube.com/watch?v=dzItGHyteng

0.5 P **+ Contours (backfaces)**

The contours were analogue to the cel shading also applied to the spaceship and the meteors to achieve a comic style look to our not so serious surreal game. This tutorial guided the implementation for the backface contours http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=15&topic=Cel-Shading.

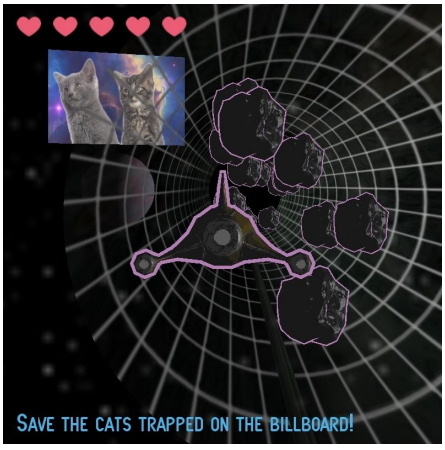
(Cel Shading and backfaces were not applied to the planet and the video as this would mask/obscure the procedural textures and the video)

---

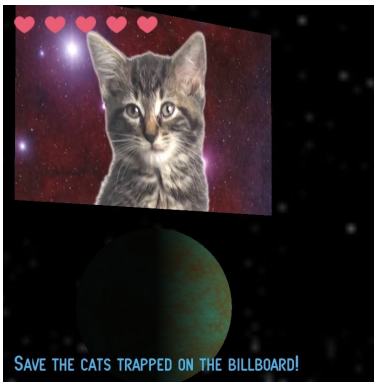
5 P

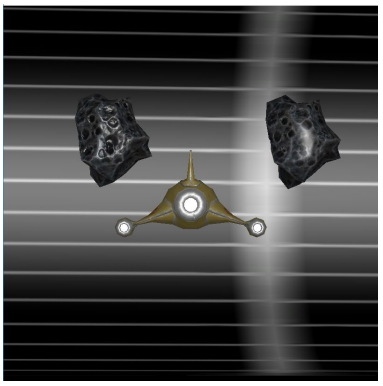
Some screenshots to illustrate our effects:

(first row from left to right)
The first image gives a general overview of what our game looks like. You can see the spaceship and the tunnel with the asteroids inside. The asteroids and the spaceship have a purple contour. Through the transparent tunnel the mystical planet with the video billboard is visible.
The second image shows an exploding asteroid with it's colorful confetti particles.



(second row from left to right)
The first image shows the planet with the procedural texture and the video billboard.
In the second image shows the close-up of an asteroids displaying the cel shading effect.
Last but not least the third image shows two asteroids side by side, one with normal mapping one without (without cel shading).





Additional libraries including references - fehlt noch was?

In addition to the libraries that were already used in the ECG Framework we included

- assimp to import models
- PhysX to handle collisions, movement and triggering.
- freetype
- FFMPEG/Libav
- stb_image

See references below.

References

We extended the ECG Framework, using amongst others it's texture mapping, lighting and material implementations

Frustum Culling:

https://github.com/gametutorials/tutorials/blob/master/OpenGL/Frustum%20Culling/Frustum.cpp

Camera:

https://learnopengl.com/Getting-started/Camera guided us for implementing the camera

Assimp model loading:

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/

Spaceship model: https://www.turbosquid.com/3d-models/free-space---3d-model/531813

Meteor model: https://www.turbosquid.com/3d-models/obj-free/927712

PhysX:

https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Index.html and Krishna Kumar (2013), *Learning Physics Modeling with PhysX* was used to implement functionality using PhysX.

HUD:

https://learnopengl.com/In-Practice/Text-Rendering

Font Atarian https://www.dafont.com/atarian-system.font

Font for hearts https://www.dafont.com/heart-shapes.font

GPU Particle System:

LVA Folien: https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/ComputeShader_SS18.pdf and https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/GPU_Particles_SS18.pdf)

+Instancing https://learnopengl.com/Advanced-OpenGL/Instancing

Procedural textures:

https://github.com/sol-prog/Perlin_Noise/blob/master/PerlinNoise.cpp
https://thebookofshaders.com/13/
http://flafla2.github.io/2014/08/09/perlinnoise.html
https://lodev.org/cgtutor/randomnoise.html

Video texture:

https://github.com/datenwolf/aveasy

FFMPEG/Libav

https://ffmpeg.zeranoe.com/builds/

stb_image

https://github.com/nothings/stb/blob/master/stb_image.h

(Normal) normal mapping:

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/
https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Cel shading:

https://www.youtube.com/watch?v=dzItGHyteng

+ Contours (backfaces):

http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=15&topic=Cel-Shading