JOHANNES KEPLER UNIVERSITÄT
LINZ

LAB PROJECT: CURIOUS PENGUINS

Annabell Rößler, k12102659
Leon Schnetzer, k12104562

Computer Graphics UE

SS2023

Contents

1 Basic story of the movie

A group of penguins is walking around in a landscape which does not resemble their natural habitat. Their walk is interrupted when they notice an ancient-looking illuminated button. One particularly brave (or stupid) penguin comes forward and presses the button without hesitation. A glowing orb then ascends from the now dark button. Responding to the signal, a UFO appears in the sky and seemingly assesses the situation. From previous adventures to Earth, the alien inside the UFO notices the lack of snow, where there was plenty in the past. The visitor from space turns on the UFOs internal snow generator and starts casting a beam of ice onto the landscape, transforming it back into a frosty paradise. Now that the spirit of Antarctica is restored once again, the penguins are jumping up and down with excitement.

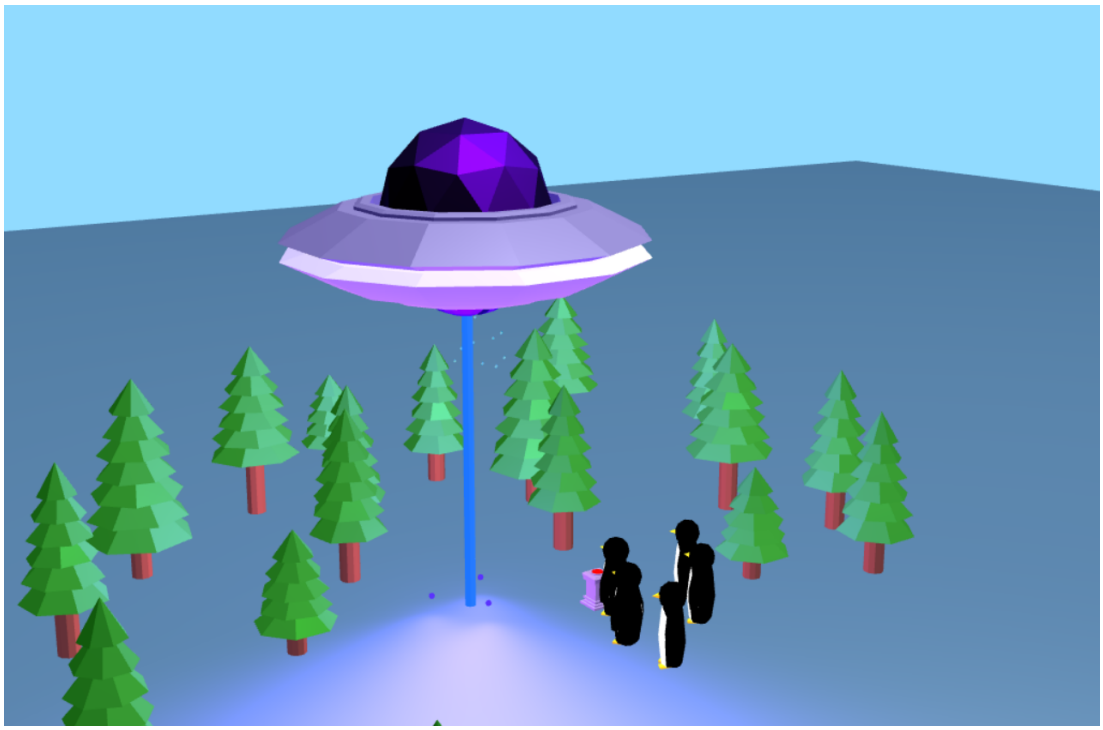

Figure 1.1: View of movie scene

2 Objects and animations

2.1 Introduction

We decided to go for a low poly look for most of our objects and created all of the base structures with the graphic software Blender. The colors/materials were added much later, and will be explained in the according section. Our project contains multiple composed objects, the most noteworthy being the UFO and our leading penguin.

2.2 Penguins

In general, we used two different types of penguins in our project: One where all parts are merged and cannot be moved separately and a second one where the body parts are separated into four groups:

- Head with beak
- Body with legs
- Right Wing
- Left Wing

This more complex model was only used for one out of the five penguins, because it needs to be able to press the button.

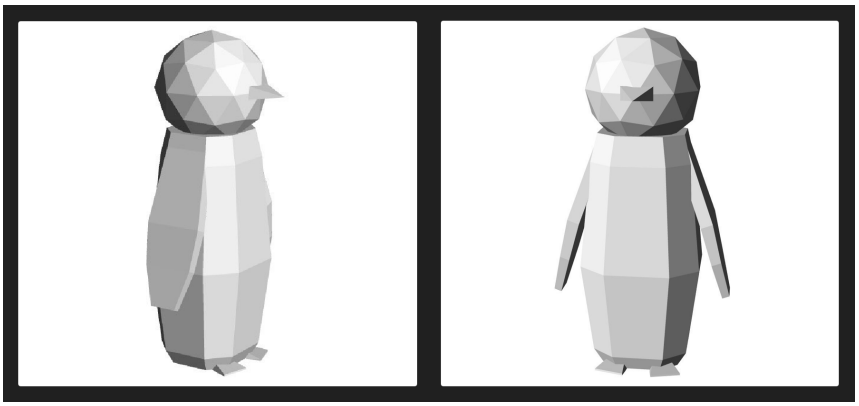

Figure 2.1: Basic penguin model

2.2.1 Waddle animation

We recreated the typical penguin waddle by alternating rotations to the left and right while moving them forwards. For each penguin a different offset is implemented in order to not make them waddle completely in sync. When choosing a final position for each penguin, the other penguins paths have to be taken into consideration to not make them collide with each other.

2.2.2 Jumping animation

The jumping animation is a simple up and down motion that is implemented with a slightly different offset for each penguin in order for them to not all jump at the same time. This animation is set to start after the UFO has finished its flight around the map.

2.2.3 Pressing the button

To make it seem like a penguin presses the button, we animated the leading penguins right arm to rotate up and down again, with the down rotation happening at the same time the button moves down into the pillar.

We first imported the right wing into Blender to the world coordinates (0,0,0) to make rotating the arm easier. It is important to be aware that Blender's axes are aligned differently than WebGL's axes. We implemented the arm rotation in our project via the addKeyFrame method, which is compromised of a translation and a rotation that was originally only meant for the camera animation. This saved us from having to do more matrix multiplications.

Figure 2.2: Penguin pressing the button

2.3 UFO

While the penguin is the object with the most complex movement set, the UFO incorporates multiple animations that only affect a single part of the whole object. According to those motions, the UFO is split into four groups:

- Dome, middle and bottom part
- Upper Disk
- Lower Disk
- Beam

The upper and lower disk both rotate around their own axis, but in separate directions. The ”static” part in the center consists of the dome on top, which also connects the two disks, and a bottom part that is designed to accommodate the ice beam. This beam is stored under the floor when it is not needed and teleported up when it should be visible.

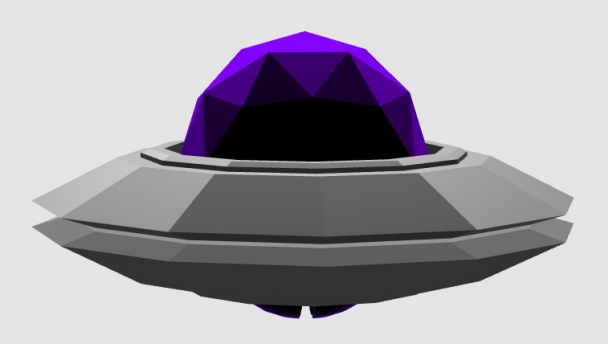

Figure 2.3: UFO basic shape

2.3.1 Disk rotations

The disk rotations are implemented with two different matrix multiplications. The upper disk is rotated by 1.7 degrees, while the lower disk is rotated by -1.7 degrees. This animation is always active.

2.3.2 Flight animation

The flight animation is split into two parts. First, the UFO flies into the area visible to the viewer. Directly afterwards, as soon as the UFO is in position, the UFO flies around the map following a route defined by multiple positions with the activated ice beam. The UFO starts to move as soon as the orb animation is finished.

2.3.3 Beam animation

The ice beam animation teleports the ice beam up from below the map and down again when it is no longer needed. The ice beam is moved along side the UFO as it makes its way around the map and disappears again afterwards.

2.4 Button

The button placed on top of a pillar is our third composed object, even if it is a very simple one. It only consists of two parts:

- Pillar
- Button

Figure 2.4: Pillar with unpressed Button

2.4.1 Button animation

The button animation implements that the button sinks down into the pillar. The timing is set in a way that the button starts its descent as soon as the moving up animation of the penguins wing is finished. The moving down animation of the penguins wing is timed to overlap with the button animation to make it look like the button is pressed.

2.5 Orb

The orb is a low poly sphere that ascends upwards after the button is pressed. We remove it from our root node after its animation is finished to make it disappear.

Figure 2.5: Orb shape

2.5.1 Orb animation

The orb animation consists of two animations, one that makes the orb move from its original position to its last position up in the sky and a second one, which adds a wobble movement to this ascend by using an bit of a special effect. This special effect is the result of using negative degrees in the deg2rad function for a rotation.

2.6 Trees

In total there are four different tree models, which not only vary in size but also in other design choices, like the trunk-to-top ratio or the angle of each separate row of branches. Regardless of which model is used, all of them are composed of two parts:

- Tree top
- Tree trunk

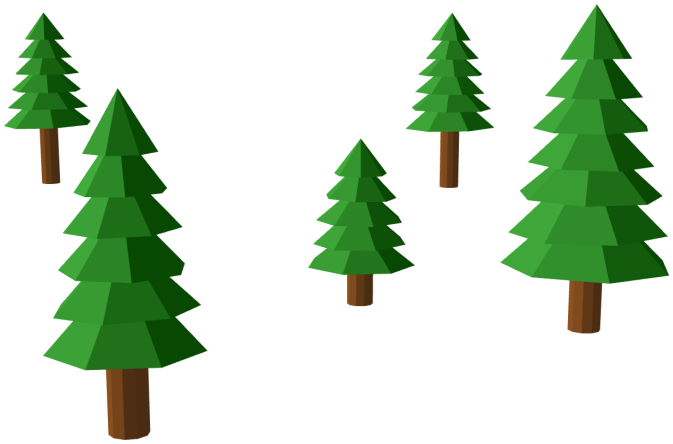

Figure 2.6: Tree Variations

3 Camera Animation

After a 30 seconds long automatic camera flight, the user is able to freely control the camera using the mouse and keyboard to move through the scene. The camera navigation was implemented in the framework already.

3.1 Automatic camera flight

For the implementation of our camera animation we used two functions.

- addKeyFrame(position, xAngle, yAngle, zAngle)
- addCameraAnimation(camera)

The addKeyFrame function is used for rotating the camera based on the given angles and then translating it to the wanted location. The addCameraAnimation function creates the complete camera animation for the video by defining suitable keyframes with the addKeyFrame function. For our camera flight we used 14 keyframes in total.

3.2 Controls for user

Users can press the WASD-keys as well as the E- and Q-key to manually control the camera along the viewing direction.

- W-key: forward movement
- S-key: backward movement
- A-key: leftward movement
- D-key: rightward movement
- E-key: upward movement
- Q-key: downward movement

Users can also utilize the mouse to control the heading and pitch of the camera relative to the ground.

- mouse-x: rotation around y-axis
- mouse-y: rotation around x-axis
- no roll

4 Textures and materials

4.1 Colors and materials

The materials and colors for our objects were chosen with the WebGL Playground tool provided in the lecture (https://jku-icg.github.io/cg_demo/00_shading/).

In contrast to this tool, WebGL uses values between 0 and 1 for defining colors. Which for us meant, to also be able to use RGB values we had to write a function (rgbToPercent) which takes the RGB values and maps them to the according percent value. This works by simply dividing the RGB values by 255.

4.2 Penguin texture map

As the penguin is the only object that includes parts that do not all have the same color, we needed to add a UV map which we created in Blender.

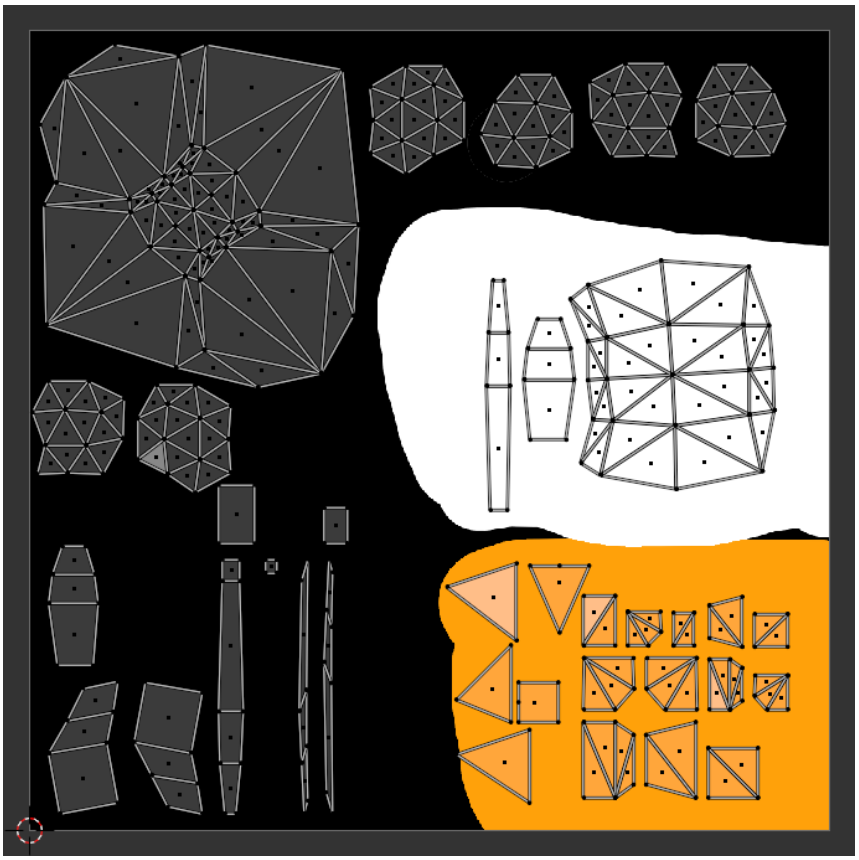

Figure 4.1: UV map for penguin

4.3 Particle texture

Instead of choosing a material for our particles, we mapped a texture we created ourselves onto it.

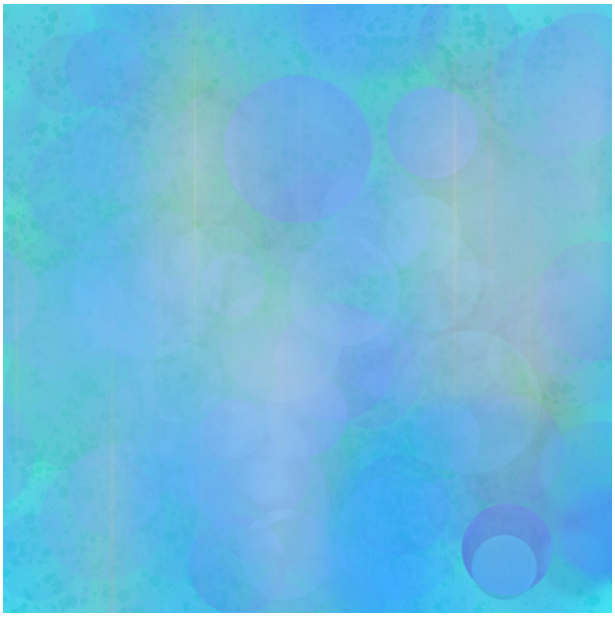

Figure 4.2: Texture for particle

5 Lights and illumination

5.1 Point lights

The majority of the code used for implementing lights can be found in the lightCreations.js file. After looking at the provided shading/illumation demo (https://github.com/JKU-ICG/cg_demo/tree/master/00_shading), we also figured out how to get colored light spheres.

5.1.1 Button

The light around the pillar has a red hue and is activated from the start. It gets disabled after the button is pressed. It is appended to the root node because it does not move.

5.1.2 Orb

The orb light is placed inside our orb object to make it glow. It emits a greenish/blueish color.This light is appended to the orb because they move completely in sync. After reaching its final destination we disable the orb light by setting all of its values to zero before the orb is removed from the scene.

5.1.3 Beam

In total there are three beam lights which circle around the beam to highlight the bottom part where the action is happening. Consequently, these lights or rather their rotations are appended to the beam after it is activated. They give off a blue to purple glow. They are teleported down below the map with the beam after deactivation, which makes them disappear for the viewer.

5.1.4 Sun

The "sun" is a big light sphere up in the air that illuminates our whole scene. It has a slightly yellow hue and is appended to the root node because it is always at the same position.

5.2 Spotlight

Our attempts to implement a spotlight failed, mainly because the illuminated part of the scene was either dependent on the viewing position or the spotlight was lighting approximately half of the scene.

6 Special effects

6.1 Particle system

We planned to implement either a particle system with snow flakes or ice crystals.

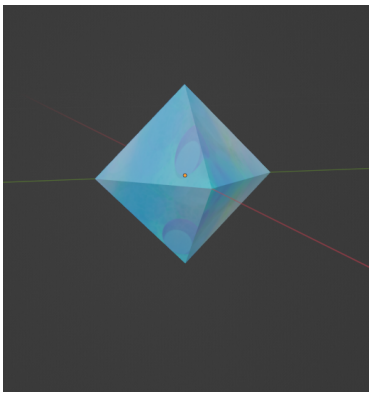

Figure 6.1: Basic shape of a particle

Unfortunately, we were not successful when it comes to the physics simulation. In its final version, our particle system can randomly spawn ice particles at the top of our ice beam directly underneath the UFO. The despawn mechanic also works as intended.

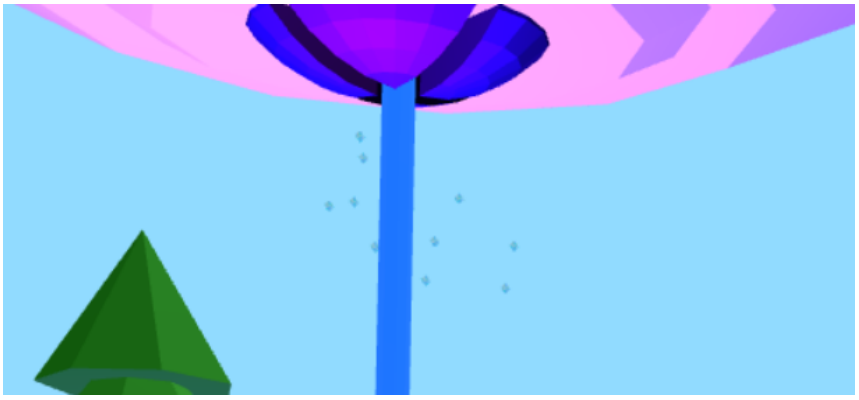

Figure 6.2: Particles around the beam

For a detailed description see README file.

6.2 Snow appearing

Our original idea to use animations to gradually turn the floor white, did not work for us. We tried to use the Animation class with matrix translations, but the floor only ever switched to grey immediately when update was called.

So instead we use our beam lights to illuminate the floor with a blue light as a transition between the grass and the snow state. To get it to look like snow covers the ground, we simply set all of the floors ambient values to 1.

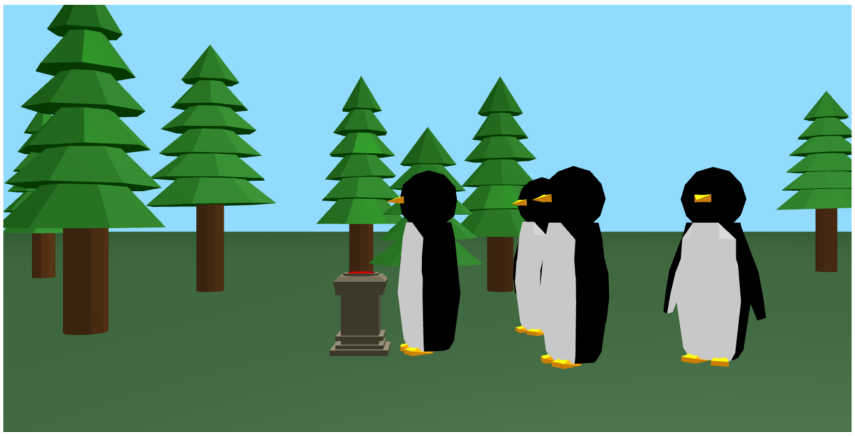

Figure 6.3: Grassy floor

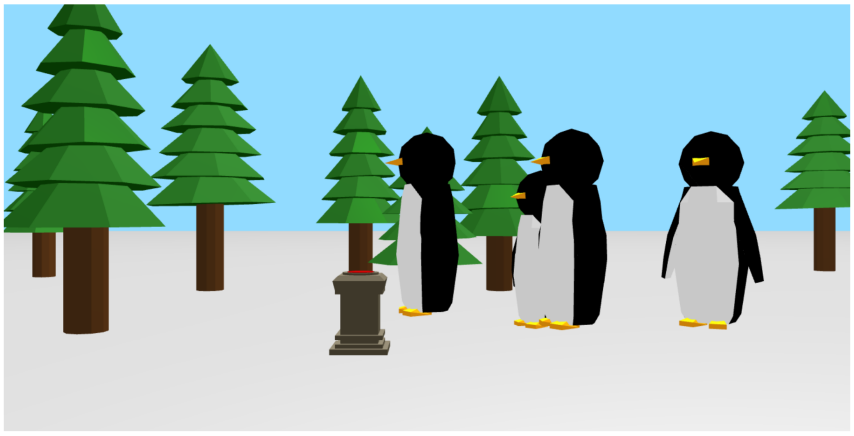

Figure 6.4: Snowy floor

6.3 README file

CG Lab Project

Submission template for the CG lab project at the Johannes Kepler University Linz.

Explanation

This `README.md` needs to be pushed to Github for each of the 3 delivery dates.
For every submission change/extend the corresponding sections by replacing the *TODO* markers. Make sure that you push everything to your Github repository before the respective deadlines. For more details, see the Moodle page.

Concept submission due on 31.03.2023

Group Members

	Student ID	First Name	Last Name	E-Mail
Student 1	k12102659	Annabell	Rößler	annabell.roessler@outlook.com
Student 2	k12104562	Leon	Schnetzer	leon.schnetzer@gmail.com

Concept Submission due on 31.03.2023

Basic story of the movie

A group of penguins is walking around in a landscape which does not resemble their natural habitat. Their walk is interrupted when they notice an ancient-looking illuminated button.
One particularly brave (or stupid) penguin comes forward and presses the button without hesitation. A glowing orb then ascends from the now dark button.
Responding to the signal, a UFO appears in the sky and seemingly assesses the situation. From previous adventures to Earth, the alien inside the UFO notices the lack of snow, where there was plenty in the past.
The visitor from space turns on the UFOs internal snow generator and starts casting a beam of ice onto the landscape, transforming it back into a frosty paradise. Now that the spirit of Antarctica is restored once again, the penguins are jumping up and down with excitement.

Special Effects

Selected special effects must add up to exactly 30 points. Replace yes/no with either yes or no.

Selected	ID	Name	Points
no	S1	Multi texturing	15
no	S2	Level of detail	15
no	S3	Billboarding	15
no	S4	Terrain from heightmap	30
no	S5	Postprocessing shader	30
no	S6	Animated water surface	30
no	S7	Minimap	30
yes	S8	Particle system (rain, smoke, fire)	30
no	S9	Motion blur	30
no	SO	Own suggestion (preapproved by email)	TODO

Intermediate Submission due on 29.04.2023

Prepare a first version of your movie that:

- is 30 seconds long,
- contains animated objects, and
- has an animated camera movement.

Push your code on the day of the submission deadline.
The repository needs to contain:

- code/ Intermediate code + resources + libs
- video/ A screen recording of the intermediate result

Nothing to change here in `README` file.

Note: You don’t need to use any lighting, materials, or textures yet. This will be discussed in later labs and can be added to the project afterwards!

Final Submission due on 20.06.2023

The repository needs to contain:

- code/ Documented code + resources + libs
- video/ A screen recording of the movie
- README.md

Workload

Student ID	Workload (in %)
k12102659	50
k12104562	50

Workload has to sum up to 100%.

Effects

Select which effects you have implemented in the table below. Replace yes/no/partial with one of the options.
Mention in the comments column of the table where you have implemented the code and where it is visible (e.g., spotlight is the lamp post shining on the street).

Implemented	ID	Name	Max. Points	Issues/Comments
yes	1a	Add at least one manually composed object that consists of multiple scene graph nodes.	6	visible in ufo, penguin and pillar; implemented in objectCreations.js
yes	1b	Animate separate parts of the composed object and also move the composed object itself in the scene.	4	visible in ufo and penguin; implemented in objectAnimations.js
yes	1c	Use at least two clearly different materials for the composed object.	3	visible in ufo; implemented in objectCreations.js; documented in Material_doc.md

Implemented	ID	Name	Max. Points	Issues/Comments
yes	1d	Texture parts of your composed object by setting proper texture coordinates.	5	visible in penguins and particle; implemented in objectCreations.js and TextureObjectNode.js
yes	2a	Use multiple light sources.	5	visible in all lights in the scene; implemented in lightCreations.js
yes	2b	One light source should be moving in the scene.	3	visible in lights moving around beam of ufo and together with the ufo; implemented in lightCreations.js and main.js
partial	2c	Implement at least one spot-light.	10	Did not work the way we wanted it to; implemented but not working in lightCreations.js, phong.vs.glsl and phong.fs.glsl
yes	2d	Apply Phong shading to all objects in the scene.	4	implemented in objectCreations.js, phong.fs.glsl and phong.vs.glsl
yes	3	The camera is animated 30 seconds without user intervention. Animation quality and complexity of the camera and the objects influence the judgement.	10	visible in camera movement; implemented in camAnimation.js
partial	Sx	Particle system	30	Could not get particle movement to work. Particle system can only spawn and remove particles near the origin location. implemented in ParticleSystemNode.js; visible in particles below ufo
partial	SE	Special effects are nicely integrated and well documented	20	See point above

Special Effect Description

Describe how the effects work in principle and how you implemented them. If your effect does not work but you tried to implement it, make sure that you explain this. Even if your code is broken do not delete it (e.g., keep it as a comment). If you describe the effect (how it works, and how to implement it in theory), then you will also get some points. If you remove the code and do not explain it in the README this will lead to 0 points for the effect and the integration SE.

The particle system is implemented as its own class ParticleSystemNode in the file ParticleSystemNode.js. The class contains the following functions:

constructor

inputs:

- **model:** The model which should be spawned as a particle by the particle system.
- **texture:** The texture of the spawned model.
- **maxParticles:** The maximum number of particles of a system which are allowed to live at the same time.
- **position:** An Array containing the current position of the particle system relative to its root.

The constructor sets fields for all input parameters and adds arrays for the spawn times of particles (birth), the max age of particles (age) and the directions of particles (direction). It also sets a field called startupTime which contains the system time at the creation of the system.

spawn

The spawn method spawns particles until the maximum number of particles has been reached. Sets birth, age(maximum age) and direction with every spawn of a particle.
The maximum age is 1500ms. The calculation is done with the formula: age = 1500*Math.random(); where Math.random() generates a number between 0 and 1.

isDead

inputs:

- **age:** The maximum age of the given particle.
- **birth:** The spawn time of the given particle.

Determines if a given particle has exceeded its maximum age by subtracting the spawn time from Date.now() and comparing this value to the maximum age.

setBuffer

Sets attribute buffers of direction and time for the shaders particles.vs.glsl and particles.fs.glsl.

update

Iterates over every node in the particle system and checks if the given node should be alive or dead. If it is dead, its values get removed from this.birth as well as this.age, and it is removed from the children list.
The Array is iterated backwards in order to avoid problems when removing values from the array while iterating the same array. The method 'isDead' is used in order to check if a particle is already dead.

render

Calls the render method of the superclass of class ParticleSystemNode. Loads values into attribute buffers and sets value for every uniform for the shaders particles.vs.glsl and particles.fs.glsl.

setDirection

Sets the direction of the particle in order to modify it in the shader. Uses (Math.round() ? 1 : -1) to generate positive as well as negative direction values.
Returns an array consisting of its x, y and z coordinate offsets.

Notes

We could not get the particle movement in the shader to work. Scaling the direction with time always resulted in spawning the particles completely out of sight or spawning one big particle which obstructed the view.
The particle spawn and delete mechanics work as they should, aside from the lag caused by too many polygons. We also could not find a solution for the errors thrown while creating the buffers.

6.4 Requirements

Lab project

The goal of this project is to create an animated movie that lasts exactly 30 seconds.

We use GitHub Classroom for the group building and the code submission. Use the following link to get started on GitHub Classroom: https://classroom.github.com/a/BM-SU2JX

Project groups

- Groups of two students
- Groups can be built independently of lab group assignments in KUSSS and lab session time slots (A-E).
- Single projects are not allowed.
- If you don't find a partner, use the Moodle forum to find a fellow student.
- If one of the partners drops the course after the concept submission, let us know immediately so that we can adapt the scope of the project.

Deadlines

- Concept submission due on 31.03.2023
- Intermediate Submission due on 29.04.2023
- Final Submission due on 20.06.2023

Minimal requirements

- The movie must be 3D.
- The movie must last exactly 30 seconds.
- All basic movie effects and special effects must be clearly visible.
- All basic movie effects and special effects must be implemented as scene graph nodes.
- All special effects must be explained.
- Framework: The implementation must be based on the unmodified lab framework (= everything in the libs folder). However, you are free to add new scene graph nodes. The usage of other external 3D/game engines, such as Three.js, is not allowed. The Github template repository which you get when creating a group via Github classroom contains the framework (so don't just use one of the lab session projects).
- Documentation: Code needs to be documented and modular (create own function for each feature if possible).
- Executability: The movie needs to run on regular PCs with Visual Studio Code, as in the lab sessions. The movie needs to be self-contained including all required resources.

Basic movie effects

1. Composed object
   a) Add at least one manually composed object that consists of multiple scene graph nodes. Your object should significantly differ from the robot created during the lab sessions.
   b) Animate separate parts of the object and also move the composed object itself in the scene.
   c) Use at least two clearly different materials with specular properties for the various parts of your composed object. Note that a material is different from a texture and that just setting different vertex colors is not sufficient.
2. Illumination
   a) Use multiple light sources.
   b) One light source should be moving in the scene.
   c) Implement at least one spot-light by extending the existing light node LightSGNode and a Phong shader. Apply Phong shading to all objects in the scene.
3. Automated camera flight (Animation) For the 30 seconds camera flight, the camera needs to be animated without user intervention and the animation needs to start automatically. You need to create the animation objects (see class Animation). Examples can be found in the comments above the class definition. The first parameter is the TransformSGNode that is to be animated. The second parameter details what the animation should look like. If true is supplied as the third parameter, the animation will keep looping. You can start the animations using the start() function and update the animation in the render step. After the automatic camera flight, it should be possible to freely control the camera using the mouse and keyboard to move through the scene. The camera navigation is implemented already. You can use the WASD-keys to manually control the camera along the viewing direction. (W-key: forward movement, S-key: backward movement, A-key: leftward movement, D-key: rightward movement). You can use the mouse to control the heading (mouse-x; rotation around y-axis) and pitch (mouse-y; rotation around x-axis) of the camera relative to the ground (no roll!).

Special effects

- Multiple effects must add up to 30 points.
- Trivial solutions can result in decreased points.
- The effects need to conceptually fit into the submitted movie screenplay.
- Describe the special effects and how you implemented them in the README.md file.
- 20 points for the complexity, integration, and documentation of the special effects.

Effect selection

Simple (15 points)

S1. Multi texturing
Mix multiple textures with an alpha map in the shader. Make sure to combine at least two different textures. For the texture lookup different texture coordinates are typically required. For example, combine a texture showing grass with a texture showing pavement using an alpha texture that defines a walking path.
S2. Level of detail

Implement a level-of-detail render node and use three different detail levels of your model. Decide which version to render by using the distance of the object to the camera. Low resolution (i.e., low polygon count) should be used, if the object is further away from the camera. Make sure the swap is visible at some point during your animated camera flight. Note that this is NOT level of detail for textures.

S3. Billboarding
Implement a billboarding node that makes a flat object face the camera orthogonally at all time. The billboards should mimic 3D objects, such as trees or clouds, in the scene. Make sure the billboard is viewed from multiple different directions during the animated camera flight to prove that its orientation is updated correctly.

Advanced (30 points)

S4. Terrain from heightmap
Compute the normal vectors for your terrain to allow correct Phong shading. Reading the precomputed normals from another texture is allowed as well.

S5. Post-processing shader
Examples are a toon shader, bloom shader, glare effect, sobel operator, or color space transformation. Tune your post-processing shader carefully such that it looks good, while all other aspects, e.g. textures and different material properties, are still clearly visible. Keep in mind that a post-processing effect is applied to the image generated by rendering the scene to a texture. This means you need multiple render passes: One for rendering the scene to a texture and a second one for rendering the texture to the screen and thereby applying the post-processing effect.

S6. Animated water surface
The animated water surface should at least reflect a skybox and simulate wave movements in a nice way. The normals and thus the reflections should be influenced by the wave movements.

S7. Minimap
Show minimap in one of the corners of the screen that contains a minified version of the world (2D texture or 3D rendered) together with the path of the camera over the last 10 seconds. Moving objects don't have to be shown. Use a separate viewport for the minimap.

S8. Particle system
Examples of particle effects are rain, smoke, or fire. The particle system has to use a basic physics simulation. The animations have to be framerate-independent as well. The particle movements should be implemented in the shader.

S9. Motion blur
The motion blur should apply to both the moving objects and the camera motion. Own effect suggestions are welcome but need to be described in the movie concept submission. Approval will be given by email.