

# cgue17 - Puzzle Plane Documentation

## Implementation

### Freely movable camera

by Markus Peitl

To get the correct Camera Transformation Matrix (ViewMatrix), to calculate the ModelViewProjection Matrix from some steps are required:

1. Polling of mouse movements, to set an angle of which the Camera should look
2. Calculation of the right and direction Vectors according to horizontal and vertical angle
3. Movement in right, left, direction, -direction according to key poll events and movement speed (because the Camera is in third person, only the player Object is moved and the camera is translated in a distance behind the object, based on orientation)
4. Calculation of the projection Matrix
5. Packing the Vector components (right, up, -direction, position) into a matrix and inverting it to gain the final ViewMatrix. (The uninverted matrix orientation is copied to the player object beforehand to make it face the camera)

### Moving Objects

by Markus Peitl

For the objects to moveable any object has a ModelMatrix which represents orientation and position of the Object.

Further simple mathematical Animations were implemented which can be applied to a Scene Object and then are called in any call of the draw() method (Continuous drawing of the scene) together with a delta time (to fix speed on any pc), where they manipulate the ModelMatrix by translating or rotating it, creating a continuous movement of the object.

### Texture Mapping

by Rupert Schaffarts

Every Object in the game has a Texture applied to it. For this purpose each vertex has in addition to its position vector, a pair of U and V coordinates which are used to map the Texture onto the mesh of the object. For each Object its Textures are loaded and passed to the Shader for rendering.

# Simple Lighting and materials

by Markus Peitl

As a shading technique Blinn - Phong shading was used for every model in the scene. There is one Source of light placed in the scene that poses as a point light source and throws a specular reflection and light at objects near it. Other than that object's brightness is dependent on the ambient light factor, which is set to a value where any object even if not lit directly can be seen.

# Advanced Lighting

by Markus Peitl

Further the Lightning Pipeline was extended to be able to display multiple types of light:

- Directional Light
- Pointlights

And more than 1 of a type (for instance 1 Directional Light and 4 Pointlights)

For that structs were added in the Fragmentshader to take different variables and factors for the lights (position, ambient factor, diffuse factor, specular factor, constant attenuation, linear attenuation, quadratic attenuation).

To create a realistic light effect for Pointlights the intensity of light was limited by using the attenuation factors (the light is weaker the further away the point in the world is)

# Assets (Created with 3ds max)

by Markus Peitl

To fill the world some 3D objects were created with 3ds max:

environment1.obj

Is a big and long curved mesh that fits the purpose to limit the bounds of the playing field. Was designed to look like a spaceship and combined with a metallic looking texture.

trichter.obj

A cylinder/pyramid like object with a hole in it for the player to fly through with a texture of blue steel

zeroGem.obj

A little Gem Mesh that can be picked up by the player in the game, has a gem like texture

plane1.obj

A warped plane for player collisions, has a wood texture

longtube.obj

A tube for the player to fly through, has a grass texture

arrow.obj

A mesh of Arrows which point to the middle, has a red carpet texture

## **Additional Assets**

by Markus Peitl

Music File which plays in Loop while the game is running

Chapstick.wav

Was used from the Youtube Audio Library

## **Controls**

by Markus Peitl

(see Freely movable camera)

Controls are in third person where the camera is rotating around the player object in a fixed distance behind the object.

While the camera is rotating the player object assumes the same rotation as the camera.

The camera can be rotated by using the mouse.

By using the keys w, a, s, d the camera can move around the player object.

a, d = strafing left and right

w = moving forward

s = moving backward/ decelerating

The player object itself has a forward movement animation attached to it, so even without the player using the controls the player will move forward

r = Reset the player, timer, collectable objects in level

## **Advanced Controls/Camera**

by Markus Peitl

For the second submission the controls were extended to be more smooth and made the camera move in a more non linear fashion.

This was done by saving the movement direction and vector in camera and manipulating its values over time by non linear functions.

For this an  $1 - \exp^{-x}$  function over time was mainly used so that the player speed does increase fast at first and then slower until it reaches the maximum speed after some time. Also things like keeping the velocity of the player object and braking movement were implemented

# Basic Game Play

by Markus Peitl

Goal of the game as it is now is to collect the blue gems which are in the holes of the rotating planes in front of you when starting the game.

This can happen by moving forward with the controls described above and trying to hit the gems with you player object.

If you hit the gems in the correct order and do not miss any of them you get bonus points which are displayed at the top of the screen.

When missing a gem there will be penalty points added in the hud.

The game ends when all blue gems in the scene are collected and then a winning message pops up in the middle of the screen.

Which tells you the time it took to finish the level and the bonus and penalty points which you acquired

## Bullet Physics integration

by Rupert Schaffarts

Physics Engine / Collision

To handle collision physics the bullet library used. For collision object `gtImpact TriMesh` shapes were used to enable collision detection between dynamic concave objects (the plane exposes for example a hole the player has to fly thru without touching the corner of the hole). Objects colliding are pushed back using bullet's builtin collision mechanisms.

## Features

- Object Loading, Image Loading
- Texture sampling and shading with Texture
- Object drawing
- Scene Drawing (collection of Objects)
- Level Loading of .txt file
- Simple attachable Animations (move, rotate)
- Vbo indexing
- Third Person Camera
- Player movement
- Blinn Phong shading
- Pick Up Objects
- Pickup Order check (objects were picked up in the right order)

- Bonus and Penalty points -> displayed as 2d Texture Strings
- Winning when right objects were picked up
- Loading Objects from .txt with position, rotation, scale, animation, collectable flag
- Textured Meshes
- Using Geometry center feature

## Special Features

### Geometric Position Origin

by Markus Peitl

When loading the object and pushing the vertices onto the vector, the minimal and maximal positions are stored.

Then at the end the geometrical center is calculated as a result and stored into a boundsMatrix.

This boundsMatrix is additionally used when rotating and moving the object, so the geometrical center stays in the right position.

This has some advantages:

- The object center and geometrical center can differ from each other
- Checking the distance to the object rather than the pivot
- Composition of a scene of multiple objects in a 3d modelling program and then single objects can be picked up and the whole scene can be animated (See how the planes and gems were done)

Disadvantages:

- boundsMatrix also needs to be transformed in move or rotation methods so transformations will be done twice
- difficult to implement the relationship between object pivot and geometric center when transforming in multiple ways

In newest version this is only used to create the bounding box of the object which is needed when doing frustum culling

### Attachable Animations

by Markus Peitl

Every object contains a shared\_ptr<Animation> to its attached animation.

The method applyanimation is called in the SceneObject in every drawcall.

The class Animation was used as an interface (abstract class) from which concrete animations derive. So in the attached animation, different types of concrete animations can be stored

and in the drawcall the anonymous method applyAnimation is called in the correct animation.

2 Animation types are currently implemented: MoveAnimation and RotateAnimation

# Combined Objects and Animations

by Markus Peitl

To attach one object to another it can be set as a child object of a parent object.

This has the advantages that Transformations of the parent object are passed to the child object (under consideration of the relative position to the parent object).

By doing this the child object appears to have a fixed position on the parent object, even if the parent object is moved or rotated.

This is especially important when trying to create combined Animations where an Animation is applied on the child object and then an animation is applied on the parent object, which causes the parent animation to affect the position of the child, which can have a separate animation effect (combined animation).

This effect can be seen when looking at the rotating planes that have a rotating gem inside the hole where the player can fly through.

# Scene Objects in a Map

by Markus Peitl

For storing the Scene Objects in the Scene a map was used as data structure, because single Objects can be accessed fast and can be called by their ID.

# Level Scripts

by Markus Peitl

To load meshes into the scene with certain flags (can be collected, is a child object), certain positions and orientations and applied animations a text file structure was created to interpret a Level script and therefore be able to load different levels out of text files.

This way a playable level can be loaded of a file.

The LevelLoader interprets these files and creates a Scene (multiple Meshes, Lights, etc.) out of them.

# Collectable Objects

by Markus Peitl

The aim of the game is to collect all collectable objects in the scene by the right order in which they were placed. Therefore some objects need to be collectable.

This is done by checking the distance between player object and collectable objects and removing them (+ scoring points) when the distance falls under a certain threshold.

# Level Score

by Markus Peitl

For picking up the gems in the right order the player scores bonus points, if gems of the right order are ignored penalty points will be issued.

Further the time between level start and level finish will be measured.

All of these things will be displayed on the screen if the player has won a level.

## Collision Pushback Impact

by Rupert Schaffarts & Markus Peitl

When the player objects hit an object a impact vector will be passed to the player objects, which then adds the full amount of this impact over time (smooth move impact)

## View Frustrum

by Markus Peitl

The view frustrum was implemented by using a pyramid shape which covers the whole field of view angle, as inspired by this website:

[http://cgvr.informatik.uni-bremen.de/teaching/cg\\_literatur/lighthouse3d\\_view\\_frustum\\_culling/index.html](http://cgvr.informatik.uni-bremen.de/teaching/cg_literatur/lighthouse3d_view_frustum_culling/index.html)

This is done by creating the frustrum shape as defined by 6 plane points and their normal and then checking if a point of the object is inside all of this planes ( $(p - p_0) \cdot \text{normal} \geq 0$ ). This is done for objects by using their bounding box as a shape to check if it is inside the view frustrum.

## Following light

by Markus Peitl

To properly light areas around the player the position of a light follows in front of the player. This is done by uploading the position + direction \* X, where X is a distance to the player, to a pointlight in the fragmentshader that does the lighting, causing the light to move with the player

## Texture Loading

by Rupert Schaffarts

## Blending

by Markus Peitl

To display all object as half transparent, blending can be enabled.

This is done by enabling blending in opengl and passing a function to calculate alpha values.

Further the the Textures need to be reconfigured to hold RGBA instead of RGB values.

As a result parts of the fragmentshaders were rewritten to also output a alpha value as w component of a vec4.

# Post Processing Pipeline

by **Markus Peitl**

To implement post - processing effects further down the line the scene needs to be drawn into a different framebuffer which holds a texture.

Then this texture holding the fully rendered scene is passed to a postprocessing shader and then drawn onto a screen spanning quad.

# Effects

## Environment Mapping 1

by **Rupert Schaffarts**

Baked Lightmaps and Texturemaps

In Game Levels 6 and 7 static Lightmaps and static Texture were used which were baked using 6 pass processing in Blender.

Each were applied to the static large environment Sceneobject which is used in every Game Level.

The lightmap uses a sun light to lit the environment scene object. The light included is static and cannot be moved at runtime.

The texture map uses metal10.jpg which was downloade from CG Labs Texture repository.

It was baked in Blender as well.

## Lightmapping + Seperate Textures 1

by **Rupert Schaffarts**

Skybox / Environment mapping

The skybox / environment mapping visual effect was used to give the player the impression of an unbounded play environment. For the skybox a cube of a mountainous lake area was selected in the format of a 6 sided map out of which the skybox cube was built for rendering. The skybox was then mapped using reflection method onto the main Scene object.

## Cel Shading + Contours 1.5

by **Markus Peitl**

To implement discrete shades as done in cel shading, we need to take a look at the fragmentshader that does the lighting and map the smooth lighting values to a set of discrete samples right after we have calculated and combined all intensities of the lights in the scene and right before combining these values with the texture color.

To implement Contours we need to leverage the post - processing pipeline which structure was discussed in the feature point **Post Processing Pipeline** above.

For this we use the 2nd fragmentshader before rendering the texture to the screen and manipulate it first. Contours are detected in screen space by running a laplacian filter kernel over the texture image, which derives the image only leaving strong changes of intensities in the image behind, effectively highlighting the edges which are then mapped to a black value and combined with the original scene to create a contour highlighting (or blackening) effect.

## Bloom 1

by Markus Peitl

Bloom is an effect where the light of brightly lighted surfaces overshines the area surrounding this surface.

To do this we first need to detect and store bright areas, this is done by writing the color in an additional Texture if the light intensity reaches a certain threshold.

We do this in the fragmentshader where the lighting is done.

Now we have a texture which only holds all the bright areas of the scene.

We use that Texture in the postprocessing shader, where we first blur the image with a kernel as done with contour detection and then combine the blurred pixel back with the original image at a certain split (how much of the original pixel - how much of the blurred brightness pixel color), this creates an effect where bright areas seem to overshine surrounding space and appear to be slightly blurred.

## Additional libraries used

**GLFW** library:

<http://www.glfw.org/>

The **GLFW** library has been used creating windows, contexts and surfaces and for receiving input and events.

**GLEW** (OpenGL Extension Wrangler) library:

<http://glew.sourceforge.net/basic.html>

The **GLEW** library has been used in glewExperimental mode to create a valid OpenGL rendering context.

**ImageMagick Magick++** library

<http://www.imagemagick.org/Magick++/>

The ImageMagick Magick++ library has been used for loading the images for the game objects.

**Assimp** library

<http://assimp.sourceforge.net/>

To load .obj and .mtl file into the program the assimp library was used. The objects were then rendered by us.

**Bullet Physics**

<http://bulletphysics.org/wordpress/>

Used for collision and collision point detection

**SFML** library

<https://www.sfml-dev.org/>

Multimedia library used for playing a .wav music file