# Programming Studio 2 – Project – Own Topic

**Erika Marttinen 885665**
**1st year student of Computer Science (Tietotekniikka)** **16.2.2021**

## GENERAL PLAN

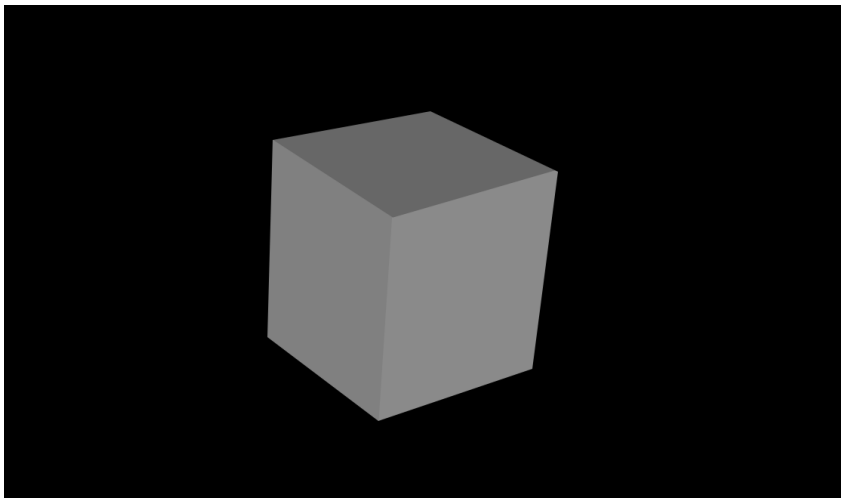
The program will be a simple 3D viewer. The user can upload a 3D object from a file and view said object from different angles. It is also possible to apply a simple animation to the object (e.g. rotating). The object consists of points and triangles connecting the points.

The program will use matrices to calculate the location of each point in the three-dimensional space. Translation, scaling, rotation and projection matrices will make sure that the object is moved according to the users wishes. These will be written by hand, so no pre-made 3D libraries are used. A light source will pe placed somewhere in the virtual space. Using laws of reflection, the corresponding color of each triangle will be calculated. This ensures that the final product actually looks three-dimensional.

The coordinates of the points and triangles will be projected onto a two-dimensional surface. The Java Swing (or JavaFX) library will render these points, thus making the whole scene visible to the user.

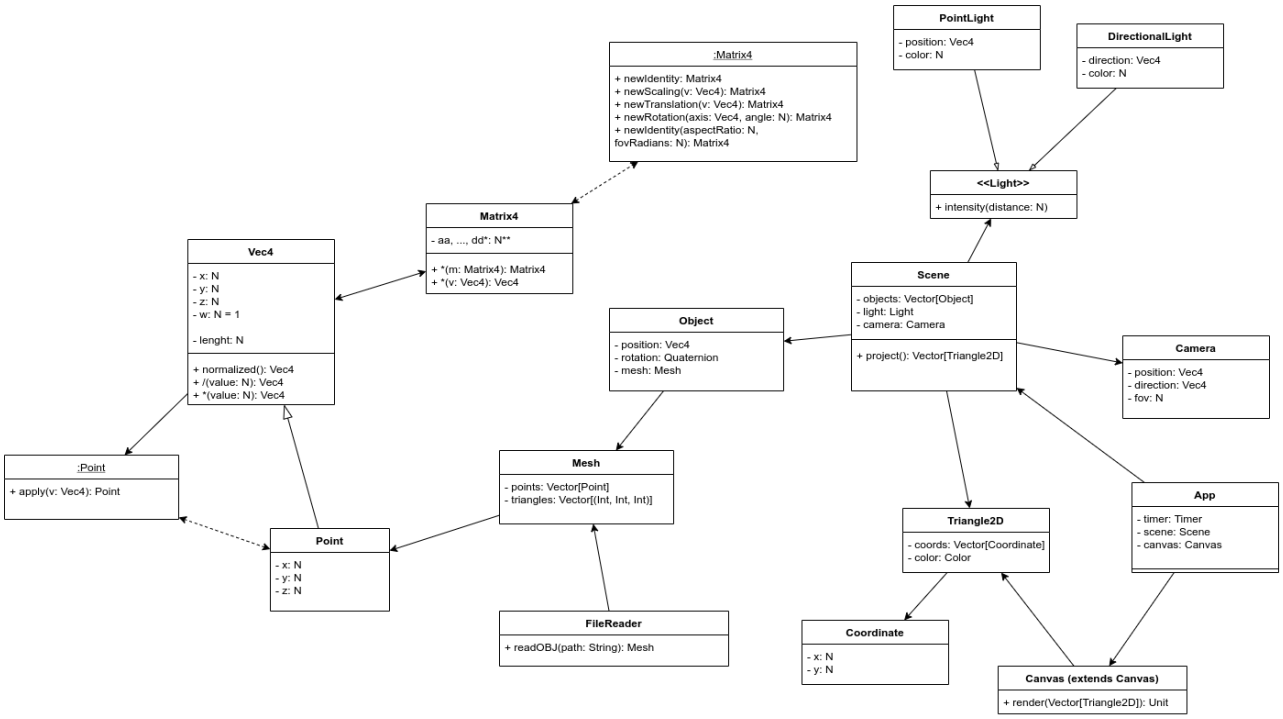
Objects can be uploaded from files. For example the user can render a simple shape (consisting of triangles) in Blender (or a similar program) and export it to a format suitable for the project's 3D viewer. The file type will most likely be .obj, possibly some other commonly used 3D file format. The user will either have a steady camera and have the object moving/spinning, or the object will remain still while the user moves around in the three-dimensional space. (The latter option might be removed, if it proves too difficult to implement.)

Concept for the graphics:



**TECHNICAL PLAN**

UML:




UML notes:

- * aa, ab, ac, ad, ba, …, dd are the elements of the 4x4 matrix
- ** Type N will originally be double. Defining it separately makes editing (e.g. to long) easier in the future.

- The Matrix4 and Vec4 classes will contain all basic operations (most importantly multiplication). Points are essentially Vec4s, where the fourth element w = 1 (for directions w = 0). The Matrix4 object will have the methods for creating the necessary matrices for rendering.
- The light will either be a point light or a directional light..
- A File Reader -class is able to turn an .obj file into a mesh.
- The Camera class might be removed later.
- The App class (and the timer) will do the animating.

USE CASE DESCRIPTION

The user starts the program. First he uploads the desired object, which the program will then turn into points and triangle surfaces. Then the user starts the animation and the object will start rotating by default. The program will then apply the corresponding matrices and start rendering each frame. The user can (possibly) also move around freely. The animation will stop when the user closes the window.

ALGORITHMS

The program will read the .obj -file and turn each read line into a point (vec4) or triangle. A buffered reader will do the reading. The triangle is a tuple containing the three points that are the triangles edges. All of this will be saved in two (collection) vectors in the mesh.

The math behind the scenes is based on matrices. When applying the rotation, scaling, translation and projection matrices in the correct order, the object will be visible on the screen in the desired way. The matrix (and vector) multiplication operator will be written by hand.

Euclidean rotation matrices don't not work when combining multiple rotations. Therefore, Quaternions are needed to make complex rotations possible.

A normal vector will also be calculated for each triangle. This will be done using cross products.

A point light (possibly directional light) is located somewhere in the 3D space. The program calculates the color of each triangle using Lambertian reflectance $I_D = \boldsymbol{L} \cdot \boldsymbol{N} C I_L$ , where **N** is the surface's normal vector, **L** is the direction of the light, C is the color and $I_L$ the intensity of the light. The program will first only render single colored objects. Other colors may be implemented later.

It is possible that this law of reflection will end up not working the way I want it to. In that case another option is needed.

Now the points will be projected onto a 2D surface. With each points 2D coordinates and the colors of the triangles, the ScalaFX library will draw the scene.

DATA STRUCTURES

The program will use Scala's pre-made collections, mainly vectors. No dynamic arrays are needed, as the object will not change between frames. The vectors are only there to store information.

All numbers will be of type N. Using this format makes future editing easier.

SCHEDULE

| Week | TO-DO | Time estimate (h) |
|---|---|---|
| 9-10 | • Create project, set up GitLab<br>• Open window with ScalaFX<br>• Draw simple image with ScalaFX<br>• Implement Matrix4 and Vec4 classes<br>• Project point with the matrices<br>• Animate square consisting of 4 points<br>  ◦ To test that the matrices work | 8 |
| 11-12 | • Add point light<br>• Add surfaces to the square<br>• Add testing cube<br>  ◦ To more clearly see how the point light calculations work | 6 |
| 13-14 | • Implement reader from .obj file<br>• Test reader<br>• Implement initial UI | 10 |
| 15-16 | • General refining<br>• Improve UI | 4 |
| 17 | • Final DL | |

TESTING PLAN

The program will be implemented in small parts. First the basics: opening a window and animating something on it. After this the core mechanics will be tested one by one. The testing of the independent features will take place before the UI implementation. A rough estimate of the order:

1. Opening/animating a scene
2. Testing the matrices
3. Testing a simple object
4. Testing surfaces with the point light
5. Testing implemented objects
6. Implementing a simple UI
7. Expanding the UI

Unit tests will be used for testing the mesh parser (FileReader). Integration testing will be used when testing the scenes (if the projected triangles are correct).

REFERENCES/LINKS


- Scala Standard Library: https://www.scala-lang.org/api/current/
- Sample .obj files https://groups.csail.mit.edu/graphics/classes/6.837/F03/models/
- ScalaFX: http://www.scalafx.org/api/8.0/#scalafx.package, https://github.com/scalafx/ScalaFX-Tutorials
- Wikipedia