Multimedia Retrieval

Pieter Michels (5081211)

24 September 2023

1 Introduction

Before we get into the meat of this assignment some information to start is required. The goal of this project was to create a content-based 3D shape retrieval system that, given a 3D shape, finds and shows to the user the most similar shapes in a given 3D shape database.

For this task, the Java programming language was chosen. It is a language I am very comfortable with and that I haven't gotten to use in a while now. Although Python may be more kitted out with libraries that are particularly useful for this project I am confident I am able to find replacement libraries for Java or otherwise make solutions myself.

2 Reading and Viewing the Data.

Reading and viewing the shapes is easily the simplest step in the project. For reading I made a simple set of readers so I can handle every type of 3D object file that was specified per description. For the rendering part I already had to make a rather big decision on the library I would use. This is discussed in depth in subsection 2.2. Finally, I decided to also use this step to make a simple control schema, allowing users to rotate and pan the objects they are viewing.

2.1 Reading Mesh Data.

As mentioned in the assignemtn description, the tool should be able to handle `OBJ`, `OFF` and `PLY` object files. I opted to make three separate readers, one for each file type. Once reading all the vertex and face data is done the code immediately prepares buffers that are later used for rendering. These buffers are essentially flattened versions of the vertex and face arrays, a format that is needed for the rendering.

Furthermore, preparing these buffers right away will make it easier for me later to implement the normalization. This way I can have all of the preprocessing needed in one spot and I will not have to worry about it too much when the time comes.

2.2 Viewing Meshes.

Viewing and inspecting the meshes comes in two parts: Rendering and moving the mesh. The renderer is obviously a key component of this assignment and thus is the part of this step I have spent the most time on. Control wise I decided to stick to a relatively standard set of instructions for the user, which can be found in subsubsection 2.2.2.

2.2.1 Rendering

When looking for rendering libraries I found two contenders that I had to decide between: The Lightweight Java Game Library [2], LWJGL for short, and the Java OpenGL project, or JOGL, which is part of the JogAmp project [1]. Both libraries give different ways of using the powerful OpenGL API to render and move around scenes.

When choosing between the two I opted to go with JOGL. JOGL seems to be more prominent in multimedia tools rather than gaming and offers more support for important basics such as mathematics functions and control of rendering, at the cost of a slightly more complicated rendering pipeline.

At the moment, the renderer consists of a few simple parts, including the main loop of a JOGL based engine. To initialize buffers are allocated for the vertices and faces that are to be drawn, listeners for for instance the controls are attached and shaders are loaded.

The tool then enters an animation loop. In this animation loop the user can move and rotate the mesh they are viewing. Furthermore, I added three different drawing modes: A fully shaded view, a wireframe view, and a view with only the vertices. The shaded view comes with flat and Gouraud shading as can be seen in Figure 1.

2.2.2 Controls

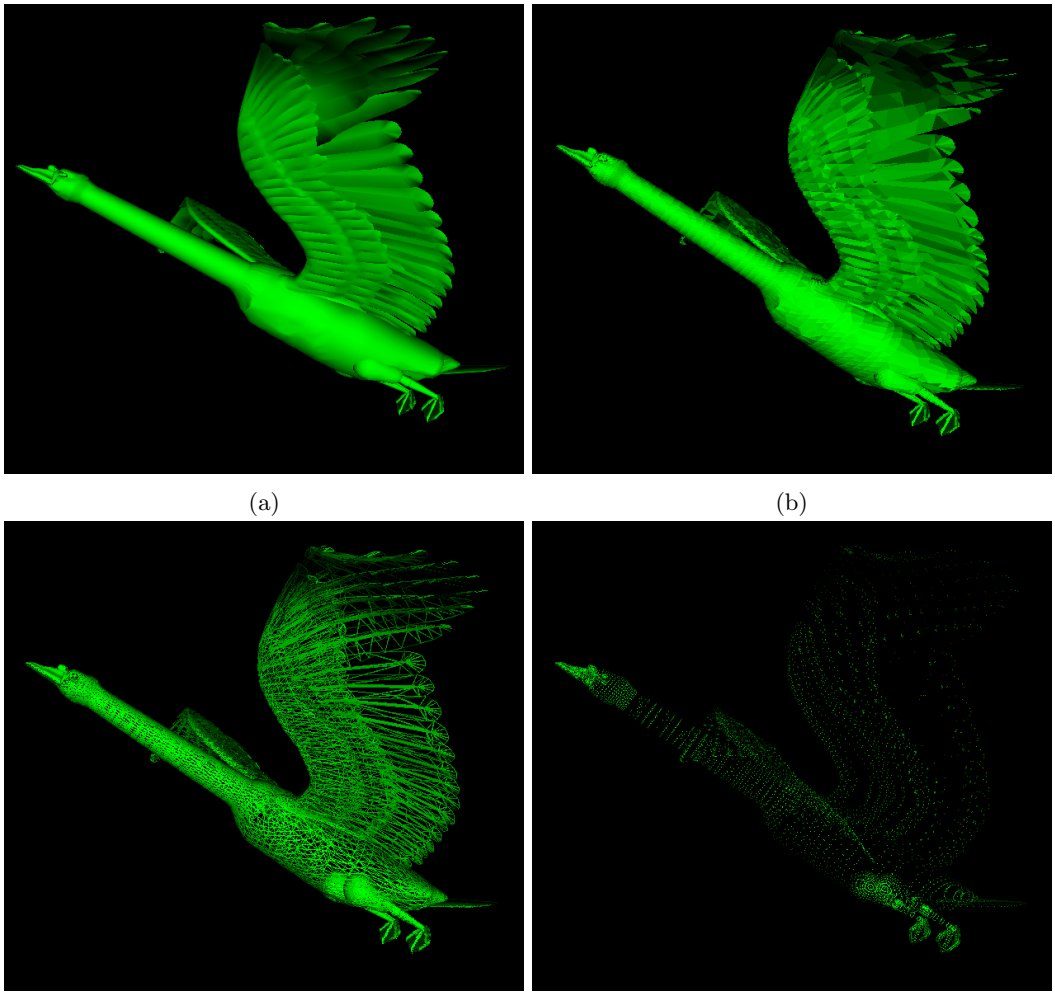
For the controls I decided to go with a pretty basic schema that I implemented using JOGL's KeyEventListener. The controls are as follows:

- W, A, S, D, Space and Ctrl for moving the object away from the camera, to the left, towards the camera, to the right, up and down respectively.
- The arrow keys, Q and E to rotate the object. The vertical arrow keys rotate around the x axis, the horizontal ones around the y axis and Q and E are used to rotate around the z axis.
- The Z key is used to swap between the three different rendering modes.

References

[1] JogAmp: High Performance Cross Platform Java Libraries for 3D Graphics, Multimedia and Processing. `http://jogamp.org/`, 2003–2023.

[2] LWJGL: Lightweight Java Game Library. `https://www.lwjgl.org/`, 2007–2023.



(a) (b)

(c) (d)

Figure 1: The four different rendering modes in the tool. (a) Gouraud shaded, (b) flat shaded, (c) wireframe and (d) the vertices.