Assignment 4 report

Giulia Benvenuto

June 11, 2022

Environment:

- Windows 11
- Visual studio 2022 17.1

1 Implementation

I decided to create a new main file and a new attributes file for each exercise, both inside the "extra" folder.

2 Ex.1: Load and Render a 3D model

- main_Ex1.cpp
- attributes_Ex1.h

To run it: cd src; python rename.py "Ex1"; cd ..; mkdir build; cd build; cmake ..; make; ./assignment4;

I extended the provided code in order to load a mesh using the **"load_off"** function taken from the Assignment 3.

The **"main"** function loads data from an .off file using the *load_off* function which stores the faces and the vertices of the mesh inside two matrices: F and V.
In order to use the *"rasterize_triangles"* function I saved the vertices in a vector which contains objects of type VertexAttributes called *"bunnyVertices"*.

If I use the identity vertex shader (Figure 1), the fragment shader with a light blue fixed color and without making any transformation the result will be a small bunny.(Figure 2).

```
program.VertexShader = [](const VertexAttributes &va, const UniformAttributes &uniform) {
    return va;
};
```

Figure 1: Identity vertex shader.

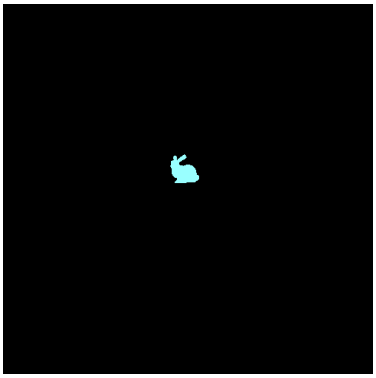


Figure 2: Result of Ex.1 using the identity vertex shader and without transformations.

In order to make the bunny bigger and place it in the center of the scene, after loading and saving the vertices of the mesh inside the vector, I performed the following steps:

- I changed the vertex shader in order to take into account the transformations applied to the scene.

```
program.VertexShader = [](const VertexAttributes &va, const UniformAttributes &uniform) {
    VertexAttributes out;
    out.position = uniform.view * va.position;
    return out;
};
```

Figure 3: Modified vertex shader.

- I computed the center of the bunny using the vertices stored in the vector. The obtained point will be used to compute the translation matrix.
- I computed the translation matrix in order to center the bunny, using the values of the obtained point as translation values (t_x, t_y, t_z).

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

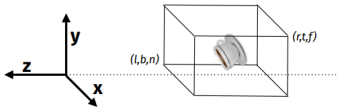
Figure 4: 3D translation matrix.

- I computed the scaling matrix in order to make the bunny five times bigger, this means that $s_x = s_y = s_z = 5$.

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 5: 3D scaling matrix.

- I computed the model matrix by multiplying the transformation matrices.
- I computed the orthogonal matrix defining the left bottom corner and the right top corner of the bounding box.

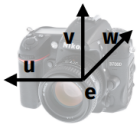


$$\mathbf{M}_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 6: Orthogonal matrix.

- I computed the camera reference system and the camera transformation matrix defining the eye position, the gaze direction and the view up vector.

1. Construct the camera reference system given:
   1. The eye position **e**
   2. The gaze direction **g**
   3. The view-up vector **t**



$$\mathbf{w} = -\frac{\mathbf{g}}{||\mathbf{g}||}$$
$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{||\mathbf{t} \times \mathbf{w}||}$$
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

2. Construct the unique transformations that converts world coordinates into camera coordinates

$$\mathbf{M}_{cam} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

Figure 7: Camera matrix.

- I set the uniform attribute for the view computed.

In this case the obtained result is:

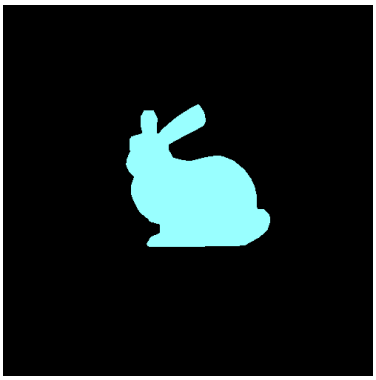


Figure 8: Result of Ex.1 with transformations

3 Ex.2: Shading

- main_Ex2.cpp
- attributes_Ex2.h

To run it: cd src; python rename.py Ex2; cd ..; mkdir build; cd build; cmake ..; make; ./assignment4;

To render three different versions of the model you can change the value of the *"uniform.renderOption"* variable inside the main, choosing between **"Wireframe"**, **"Flat_Shading"** and **"Per_Vertex_Shading"**.

- **Wireframe:** at first the data are collected from the .off file and saved in the two matrices F and V, then the vertices in each line are saved inside the *"vertices"* variable and at the end the *"rasterize_lines"* function is called obtaining the following result.

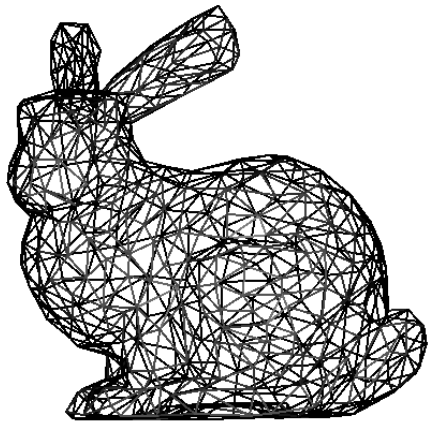


Figure 9: Wireframe bunny.

- **Flat shading:** in this case the inputs are all the triangles and all the lines that make the bunny. For each triangle is computed the normal and it is stored inside the *"normal"* vertex attribute of each vertex. At the end both the two function *"rasterize_triangles"* and *"rasterize_lines"* are called in order to see the wireframe on top of the flat shading obtaining the following result.

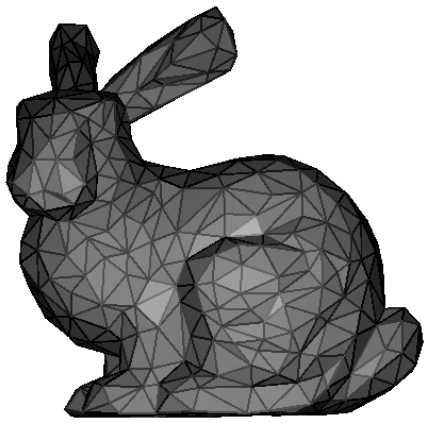


Figure 10: Flat shaded bunny with wireframe on top of it.

- **Per-vertex shading:** in this case the inputs are all the triangles of the bunny. At first I computed the normal for each face and then I computed their average on the neighboring vertices, in this way the normal of the vertex of a mesh is the average of the normals of the faces touching it. The result is the following one.

Figure 11: Per vertex shaded bunny.

4 Ex.3: Object Transformation

I implemented this exercise in the same main file of the previous exercize.

- main_Ex2.cpp
- attributes_Ex2.h

To run it: cd src; python rename.py Ex2; cd ..; mkdir build; cd build; cmake ..; make; ./assignment4;

In order to make a gif animation you can set to true the **"uniform.objectTransformation"** variable.

The rotation around the y-axis is implemented creating a matrix like the following one.

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 12: Matrix for the rotation around the y-axis.

While the bunny rotates around the y-axis it gets closer to the light in fact it becomes increasingly illuminated.

The results are inside the **"Gif"** folder inside the archive and they are called:

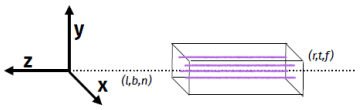
- "Wireframe Bunny Gif"
- "Flat Shading Bunny Gif"
- "Per Vertex Bunny Gif"

5 Ex.4: Camera

- main_Ex4.cpp
- attributes_Ex4.h

To run it: cd src; python rename.py Ex4; cd ..; mkdir build; cd build; cmake ..; make; ./assignment4;

In this exercize I added a perspective matrix defined in the following way:



$$\mathbf{P} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 13: Perspective matrix.

in order to implement a perspective camera.

I also added a transformation to compensate for the aspect ratio of the framebuffer, in fact if the aspect ratio is less then 0, the view matrix is modified in order to obtain a scaling on the x axis, otherwise the scaling is on the y axis.

By changing the framebuffer size we can see the results:

- **1000 x 500:**

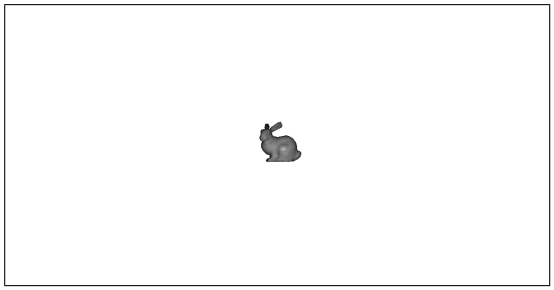


Figure 14: Framebuffer 1000 x 500.

- **500 x 1000:**

Figure 15: Framebuffer 500 x 1000.

Note that now we have a perspective camera, so we see the object rotate and we also see it getting bigger while it's getting closer. In order to see it open the gif files named "1000 x 1000", "500 x 1000" and "1000 x 500" inside the "Gif" folder.