The vertex shader

The goal of this practical work is to implement a vertex shader. The vertex shader is the first stage of our rendering pipeline:

It takes vertices expressed in the scene space and camera information as inputs and output vertices re-expressed in the projected space.

First, let's define our scene, for that, we'll reuse the code to create a cube from the previous practical work:

```python
textures = np.array([  
    [0.0,0.0,0.0], #0  
    [1.0,0.0,0.0], #1  
    [0.0,1.0,0.0], #2  
    [1.0,1.0,0.0], #3  
    [0.0,0.0,1.0], #4  
    [1.0,0.0,1.0], #5  
    [0.0,1.0,1.0], #6  
    [1.0,1.0,1.0], #7
])

triangles = np.array([  
    [1,0,2],  
    [3,1,2],  
    [4,5,6],  
    [5,7,6],  
    [0,1,4],  
    [4,1,5],  
    [2,6,3],  
])
```
Then we need to define a camera for that you’ll need the camera.py and the projection.py files. The first one contain a class that will help us define a Camera in space:

```python
import numpy as np
class Camera:
    def __init__(self, position, lookAt, up, right):
        self.position = position
        self.lookAt = lookAt
        self.up = up
        self.right = right

    def getMatrix(self):
        #todo
        pass
```

The second one will help us define the part of the camera space seen by the camera (frustum):

```python
import numpy as np
class Projection:
    def __init__(self, near, far, fov, aspectRatio):
        self.nearPlane = near
        self.farPlane = far
        self.fov = fov
        self.aspectRatio = aspectRatio

    def getMatrix(self):
        #todo
        pass
```
Let's create a camera with the following command:

```python
position = np.array([5,5,5])
lookAt = normalize(np.array([-0.577,-0.577,-0.577]))
up = np.array([0.33333333, 0.33333333, -0.66666667])
right = np.array([-0.57735027, 0.57735027, 0.])

cam = Camera(position, lookAt, up, right)
```

And a projection:

```python
nearPlane = 1.0
farPlane = 20.0
fov = 1.22173
aspectRatio = 16/9

proj = Projection(nearPlane ,farPlane,fov, aspectRatio)
```

Now using the following lines we will export a representation of the Frustum:

```python
from generateFrustum import generateFrustum
generateFrustum(cam, proj)
```

This should create a frustum.ply file that we can open in Blender, you should see the following:
Now we can export our cube as we did in the previous practical work using the following commands:

```python
from exportToPly import write_ply_file
write_ply_file(vertices, triangles, 'cube.ply')
```

Import it in Blender and activate the x-ray mode in blender you should see something like this:
As you can see our cube is inside the frustum so it is visible to the camera. Now try changing the parameters of the projection to see how they affect the frustum shape.

**Question 1: Vertex shader and camera space**

The job of the vertex shader is to re-express each vertex in the projected space frustum space. We first need to project all vertices in the camera space to do that. For this operation, we need the view matrix as seen in the lectures.

1) Modify the getmatrix function in the Camera class such that it returns the view matrix.

2) Call this function on the camera we defined, and store the result in a dictionary as follows:

```python
data = dict([('viewMatrix', cam.getMatrix())])
```

3) Open the VertexShader.py file and modify the vertex shader function such that it applies the camera matrix on every vertex:
import numpy as np

def VertexShader(vertices, data):
    outputVertices = np.zeros_like(vertices)
    for i in range(vertices.shape[0]):
        #todo
        pass
    return outputVertices

4) call your vertex shader using the following lines:

from vertexShader import VertexShader
vertices_cam = VertexShader(vertices, data)

5) export your scene in the camera space with the following lines:

from generateFrustum import generateFrustumCameraSpace
generateFrustumCameraSpace(proj)
write_ply_file(vertices, triangles, 'cubeCamera.ply')

It should generate two ply files, one named frustumCameraSpace.ply and one named cubeCamera.ply, if you open them in Blender you should see the following:
As you can see now our frustum is aligned on the three main axes of the scene, and our cube is still at the same position inside the frustum.

Now we need to apply the projection to represent the visible part of the screen in a cube rather than in a frustum as it will simplify future operation.

**Question 2: Vertex shader and Project space**

1) **Modify the getMatrix function in the Projection Class to return the projection matrix.**

2) **Modify the data dictionary to add the projection matrix:**

   ```python
   data = dict([
   ("viewMatrix", cam.getMatrix()),
   ("projMatrix", proj.getMatrix())
   ])  
   ```

3) **Modify the Vertex Shader to project all vertex in the projected space.**
   (do not forget to divide you x, y, and z coordinates by w to correctly apply the projection).
Export the projected mesh and the projected frustum with the following lines:

```python
write_ply_file(vertces_cam,triangles,'cubeProjected.ply')
from generateFrustum import generateProjectedFrustum
generateProjectedFrustum()
```

In Blender you should see the following:

As you can see the cube is now very small and its shape changed. It is normal, as the projected space is a lot smaller than the shape of the space went from a pyramid to a Rectangular cuboid.