TP3 Graphic 3D

Rasterizer & Depth buffer

In this part, we aim to implement a rasterizer as seen in the lecture.
First, you need to download the camera.py, projection.py, and graphicPipeline.py.

Then you need to complete the graphicPipeline.py :

```
import numpy as np

class Fragment:
  def __init__(self, x : int, y : int, depth : float):
    self.x = x
    self.y = y
    self.depth = depth

def edgeSide(p, v0, v1) :
  pass
  #todo

class GraphicPipeline:
  def __init__ (self, width, height):
    self.width = width
    self.height = height
    self.depthBuffer = np.ones((height, width))

  def VertexShader(self, vertices, data) :
    outputVertices = np.zeros_like(vertices)
    for i in range(vertices.shape[0]) :
      x = vertices[i][0]
      y = vertices[i][1]
      z = vertices[i][2]
      w = 1.0

      vec = np.array([[x],[y],[z],[w]])

      vec = np.matmul(data['projMatrix'],np.matmul(data['viewMatrix'],vec))
      outputVertices[i][0] = vec[0]/vec[3]
      outputVertices[i][1] = vec[1]/vec[3]
      outputVertices[i][2] = vec[2]/vec[3]

    return outputVertices

  def Rasterizer(self, v0, v1, v2) :
    fragments = []

    for j in range(0, self.height) :
      for i in range(0, self.width) :
        #x = ...
        #y = ...

        #if inside
          #emit a fragment
        pass

    return fragments

  def draw(self, vertices, triangles, data):
    newVertices = self.VertexShader(vertices, data)

    fragments = []
    for t in triangles :
      #call the rasterizer the triangle t
      pass

    for f in fragments:
      #todo Process each fragment using the depth buffer
      pass
```

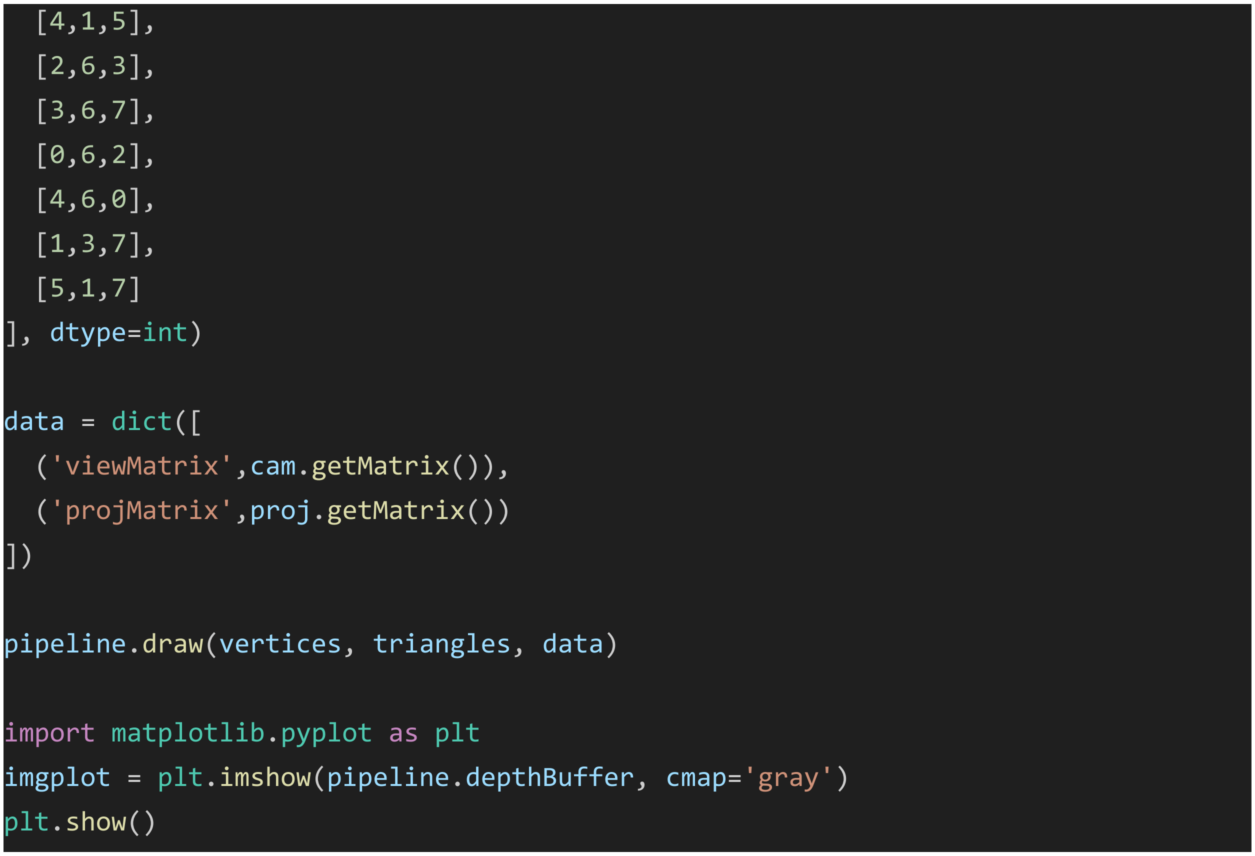
Inside outside test

To do so you should start by completing the edgeSide Function then implement the inside-outside test in the rasterizer and emit a fragment when it is needed. (For now, use 0 for the depth of the fragments)

Then as a first test, fill the depth buffer with fragment z regardless of the current depth buffer.

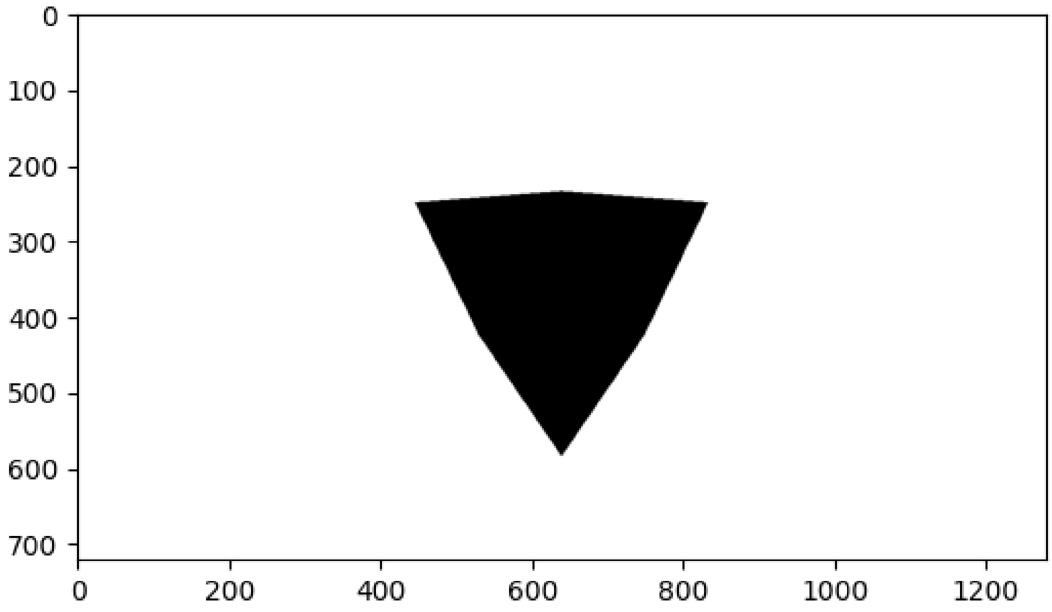
To test your code you can use the following code :

```
import numpy as np

from graphicPipeline import GraphicPipeline
width = 1280
height = 720
pipeline = GraphicPipeline(width,height)

from camera import Camera
position = np.array([1.1,1.1,1.1])
lookAt = 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)

from projection import Projection
nearPlane = 0.1
farPlane = 10.0
fov = 1.91986
aspectRatio = width/height
proj = Projection(nearPlane ,farPlane,fov, aspectRatio)

vertices = 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],
  [3,6,7],
  [0,6,2],
  [4,6,0],
  [1,3,7],
  [5,1,7]
], dtype=int)

data = dict([
  ('viewMatrix',cam.getMatrix()),
  ('projMatrix',proj.getMatrix())
])

pipeline.draw(vertices, triangles, data)

import matplotlib.pyplot as plt
imgplot = plt.imshow(pipeline.depthBuffer, cmap='gray')
plt.show()
```

It should give you the following image :
(As we didn't use any optimization the operation might take some time to complete)



Depth buffering

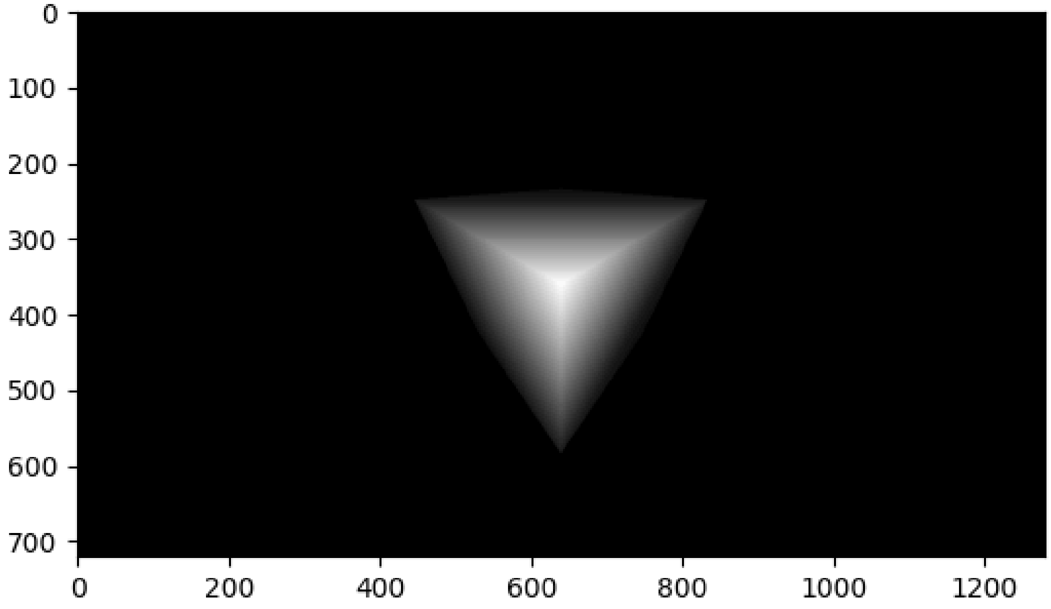
1) Compute the depth of each fragment in the Rasterizer function
2) In the draw function, modify the loop over fragments to use the depth testing
3) In the main file replace the line:

```
imgplot = plt.imshow(pipeline.depthBuffer, cmap='gray')
```

By:

```
imgplot = plt.imshow(1/pipeline.depthBuffer, cmap='gray')
```

You should obtain the following image :



Bonus Optimisation :

Implement The Axis Aligned bounding box optimization to reduce computation times.
If correctly implemented it should significantly decrease computation times
Measure the rendering time without the optimization and with it.