Name: Catherine Nguyen

Lab 2: Scan Conversion and Z-Buffer

1. Object Class:

In this lab, the Object class is implemented to hold information about edges and polygons of each object. This is to provide the ability to place multiple objects into the screen. The scan conversion algorithm is implemented per object.

```
class Object:
    def __init__(self, file):
        fileName = "D files\\" + file + ".d.txt"
        self.rawPolys, self.rawVertices = transformation(fileName)
        self.polygons = []
        self.vertices = []
        self.processVertices()
        self.setEdges()  # immediately updates polygons
```

Upon initialization of an object, perspective transformation and processing of polygons and objects are immediately called.
Color of each polygon is also generated at this step with random values.

2. Scan Conversion and Z-Buffer Algorithm:

The scan conversion algorithm is called after all the objects, the Image Buffer, and Depth Buffer have been initialized since they are implemented together.

a. Edge Table:

The edge table has been generated when the object is created as a member variable of each polygon called **edges**.
When generating these edges, if the edge has the same y_{max} and y_{min}, that means that it is a horizontal edge and would not be added to the edge table of the polygon.

```
if e.ymax != e.ymin:
        p.edges.append(e)  # non-horizontal edges only
```

After adding all the appropriate edges into the edge table, it's immediately sorted by the edges' y_{min} value.

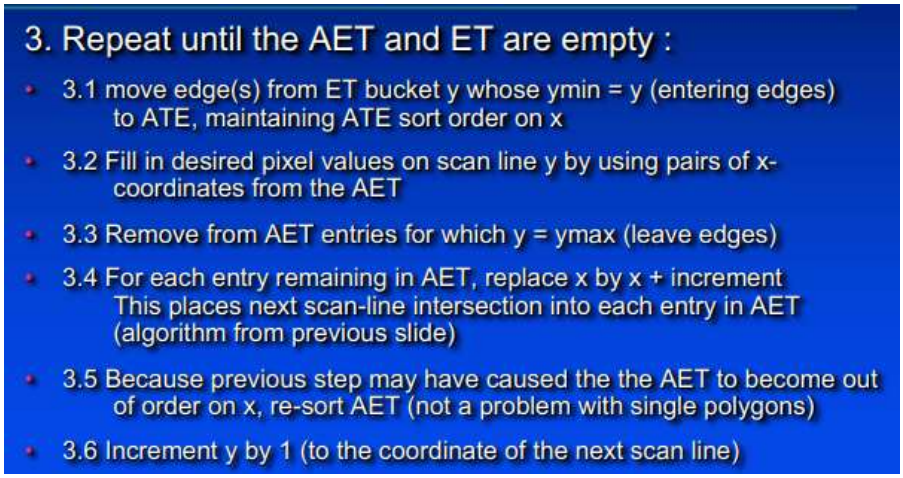
```
def sortEdges(self):
        self.edges.sort(key=lambda x: x.ymin)
```

b. Scan conversion and Z-buffer:

In the main drawing function, an image buffer and depth buffer is initialized before scan conversion of each object is done. These buffers are passed in for each scan conversion call so that Z-buffer can be implemented at the same time.

```
# initialize the image buffer and depth buffer
  image = np.zeros((1000, 1000, 3))
  depth = np.ones((1000, 1000))
```

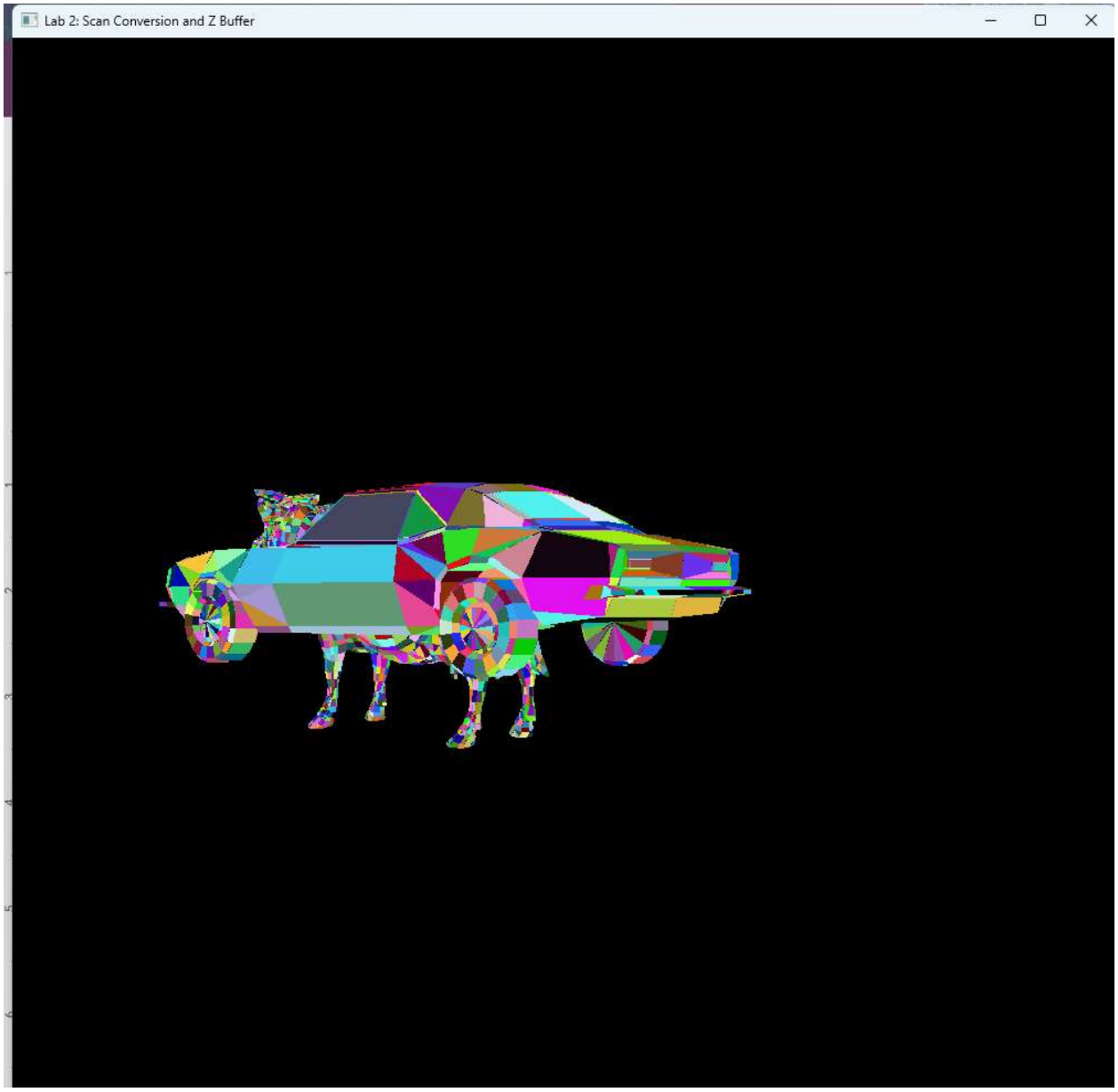
In the scan conversion for each object, each polygon in the object is considered. An Active Edge Table is created following the algorithm described in the lecture slides:



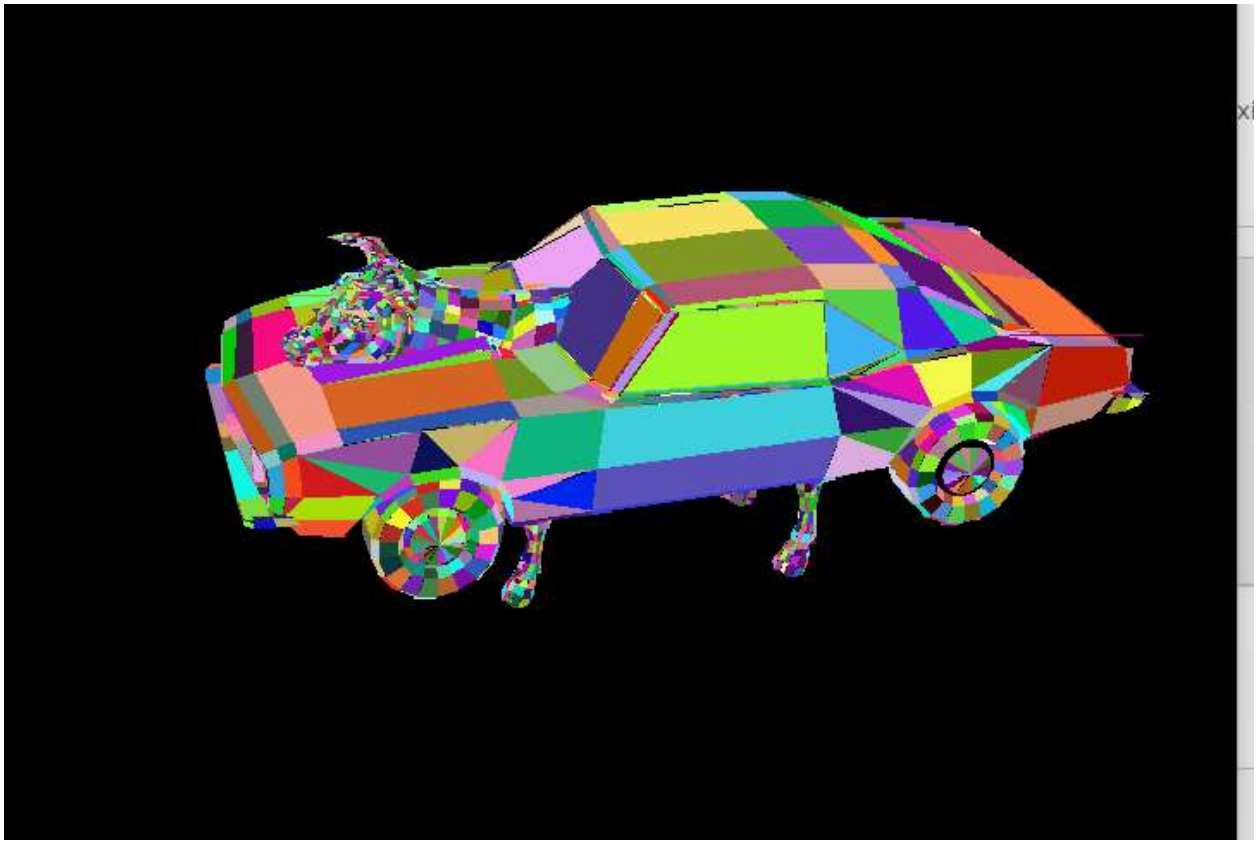
One big change is that instead of filling in desired pixel values (step 3.2), I compare that pixel to the depth buffer, and replace the value in the image buffer as appropriate. This is all the changes needed to interlace Z-buffer in the Scan Conversion algorithm.

3. Lab Samples:

Some lab samples are generated using the same camera and pRef coordinates, but some are generated using separate camera and pRef coordinates to ensure they are interlaced and placed in the middle.




```
# Cow Camaro
pRef = np.array([10, 0, 20])
cam = np.array([-20, 0, -20])  # back
```



```
#Cow camaro from top
cow_pRef = np.array([0, 0, 20])
camaro_pRef = np.array([0, 0, 20])
cam = np.array([10, 15, -25])
```


```
# Bench and car from front
cam = np.array([0, 0, -5])  # front
bench_pRef = np.array([0, 0, 0])
car_pRef = np.array([0, 0, 0])
```