CS4182-Report

Xian Jia Le, Ben 56214537

Basic Requirements

1. Adding objects

```
def addStar(x,z):
    allstars.append(cone.cone(x,z))
    obstacleCoord.append((x,z))
```

In the project, I added star objects to the virtual scene. They will be randomly distributed on the virtual road

1.1 Issues with importing .obj objects

1. Not able to find the correct path of the texture
    a. to address this issue, I added the following code:

```
elif vals[0] == "map_Kd":
   ## record the texture file name
   fileName = vals[1]
   path= os.path.abspath(self.fileName)
   parentPath = os.path.abspath(os.path.join(path, os.pardir))
 self.materials[-1][5]=(self.loadTexture(parentPath+'\\'+fileName))
   self.hasTex = True
```

As such, the program will load the texture which is placed near the .obj file.

2. Not able to load the texture
    a. After fixing the path problem, an error still occurred. This is because the function of “try-catch” is not written well. To solve this issue, simply delete the word “SystemError” which is shown in the picture below.

```
texImage = ImageOpen(texFile)
try:
    ix, iy, image = texImage.size[0], \
                    texImage.size[1], \
                    texImage.tobytes("raw", "RGBA", 0, -1)
except SystemError:
    ix, iy, image = texImage.size[0], \
                    texImage.size[1], \
                    texImage.tobytes("raw", "RGBX", 0, -1)
## GL.glGenTextures() and GL.glBindTexture() name and create a texture
```

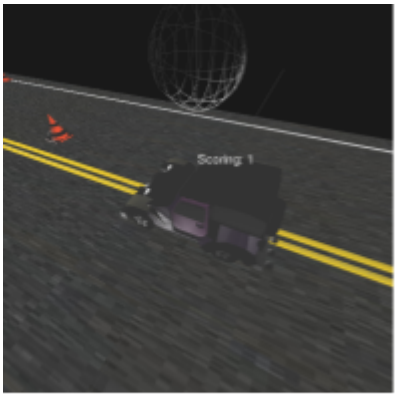
2. Menu and Light

In order to adjust the light condition of the virtual scene, I have created a sub-menu for only handling the light. The below picture is the implementation of the light and its usb menu.

```
global bPointLight, bAmbientLight
if value==1:
    if(bPointLight==False):
        print("Point light")
        glPushMatrix()
        diffuseLight = [0.1, 0.1, 0.1, 1.0]
        lightPosition = [0, 20.0, 0, 1.0]
        glEnable(GL_LIGHT1)
        glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuseLight)
        glLightfv(GL_LIGHT1, GL_POSITION, lightPosition)
        glShadeModel(GL_SMOOTH)
        glPopMatrix()
        bPointLight=True
    else:
        glDisable(GL_LIGHT1)
        bPointLight=False
if value ==2:
    if(bAmbientLight==False):
        ambientLightIntensity = [0.2, 0.2, 0.2, 1.0]
        glEnable(GL_LIGHT2)
        glLightfv(GL_LIGHT2, GL_AMBIENT, ambientLightIntensity)
        bAmbientLight=True
    else:
        glDisable(GL_LIGHT2)
        bAmbientLight=False
return 0
```

```
lightSubMenuID=glutCreateMenu(setLightSubMenu)
glutAddMenuEntry("Point light",1)
glutAddMenuEntry("Ambient light",2)
```

Users can turn on the ambient light and point light through the sub-menu.



Default light

Ambient light

Point light

3. Manipulation

3.1 Control the jeep car

```
def specialKeys(keypress, mX, mY):
    # things to do
    # this is the function to move the car
    if(keypress==GLUT_KEY_UP):
        jeepObj.move(rot=False,val=speed)

    if(keypress==GLUT_KEY_DOWN):
        jeepObj.move(rot=False,val=-speed)

    if(keypress==GLUT_KEY_LEFT):
        jeepObj.move(rot=True,val=speed)

    if(keypress==GLUT_KEY_RIGHT):
        jeepObj.move(rot=True,val=-speed)
```

In the application, users can control the car by pressing the following keys:

- Arrow_Up: move the car forwards
- Arrow_Down: move the car backwards
- Arrow_Left: rotate the car in the left direction
- Arrow_Right: rotate the car in the right direction

3.2 Following camera

When users start the game, the camera will follow the jeep automatically. Users can also press the middle mouse button to rotate the camera around the car. In addition to that, when the user scrolls the middle mouse button, the camera will zoom in/out toward the car

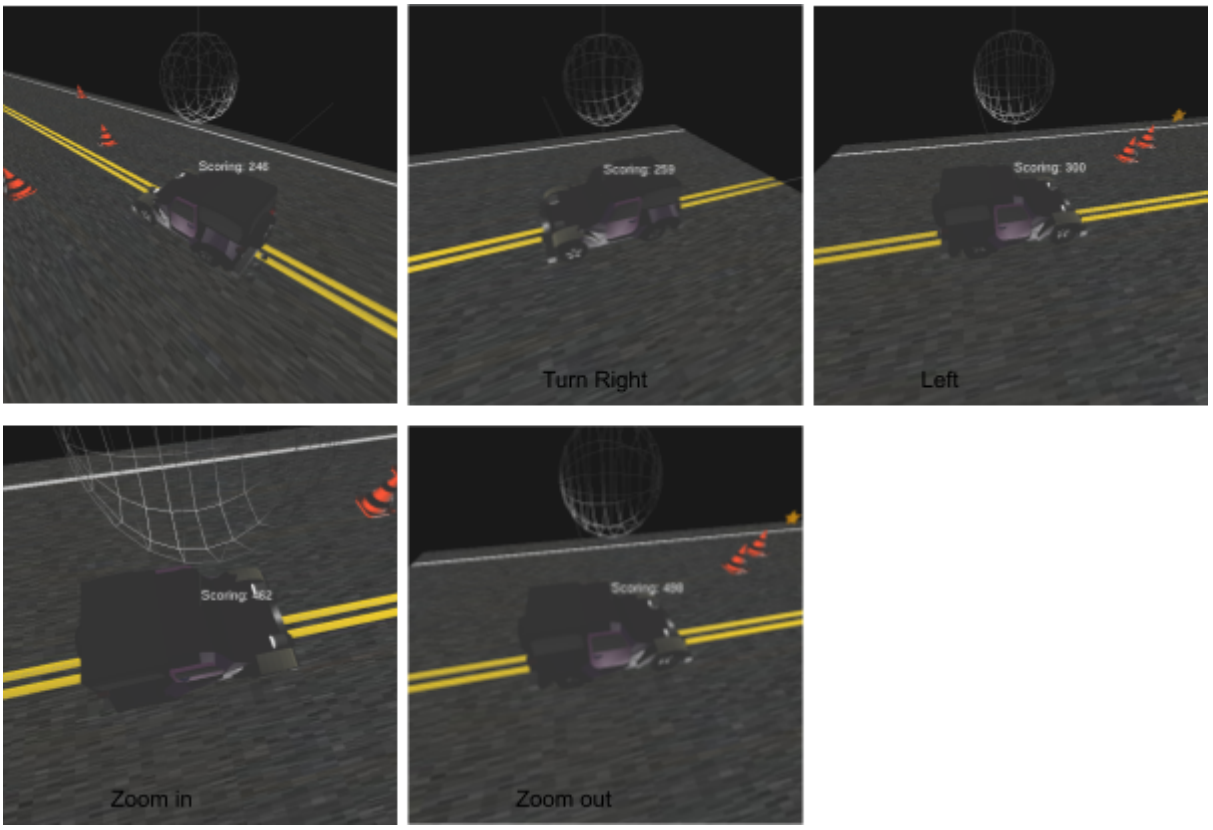
```
def setObjView():
    # things to do
    # realize a view following the jeep
    # refer to setview
    global bfollowView
    bfollowView = True
```

Middle mouse button is pressed:
eye's location is updated

```
def FollowView():
    if bfollowView==True:
        global eyeX, eyeY, eyeZ,windowWidth,windowHeight
        aspectRatio = float (windowWidth / windowHeight)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(90, float(aspectRatio), 0.1, 100)
        gluLookAt(jeepObj.posX+eyeX, jeepObj.posY +10, jeepObj.posZ+eyeZ, jeepObj.posX, jeepObj.posY, jeepObj.posZ, 0, 1, 0)
        glMatrixMode(GL_MODELVIEW)
        glutPostRedisplay()
```

```
def mouseWheelHandle(button, direction, x, y):
    global radius, eyeX, eyeZ
    radius=radius+direction
    radius = 4 if radius<4 else radius
    radius = 12 if radius>12 else radius
    eyeX = radius * math.sin(angle)
    eyeZ = radius * math.cos(angle)
```

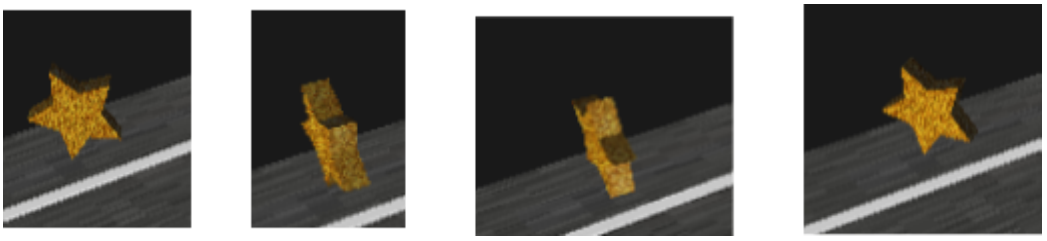
Middle mouse button is scrolled:
radius is updated



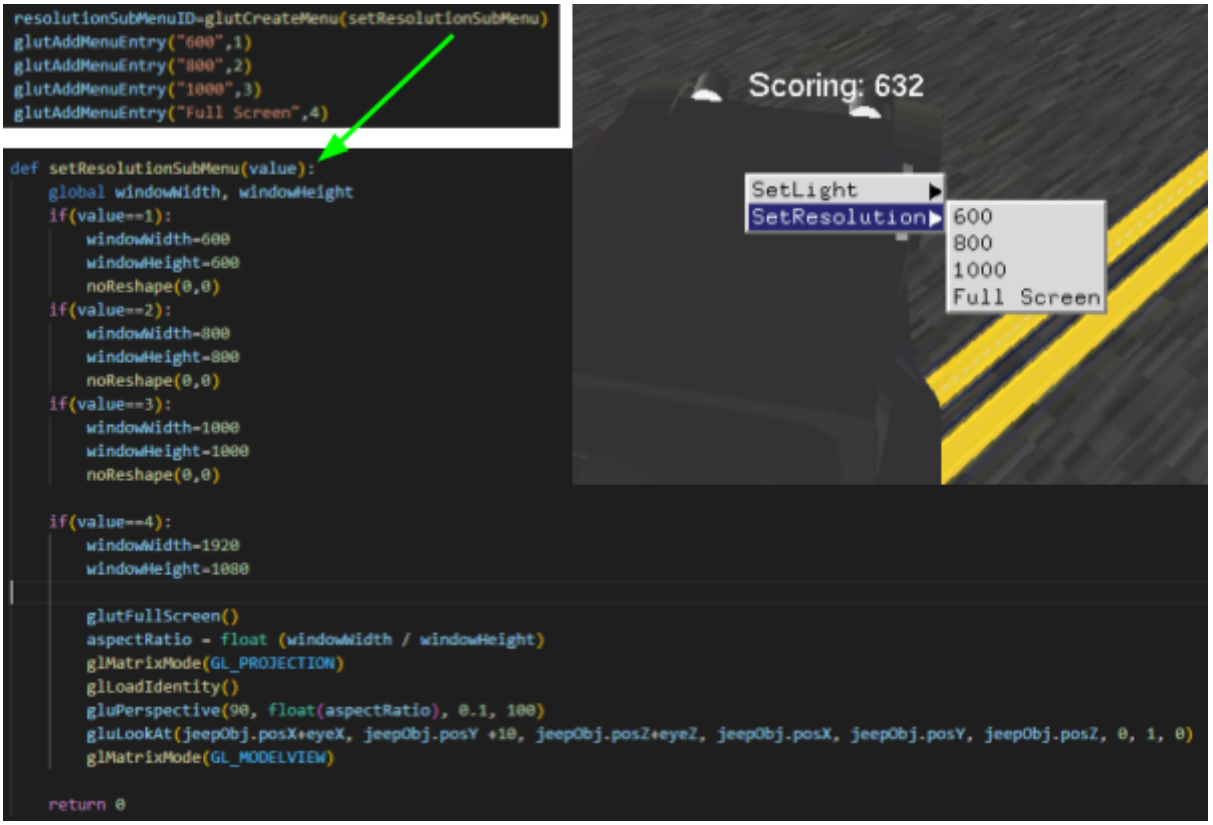
4. Adding autonomous objects

In the application, the star will be rotated by itself after they have been generated. Because of that, they will have a reaction with the light reflection which they are rotating.

```
def update(self):
    self.rotation=(self.rotation+0.5)%360
```



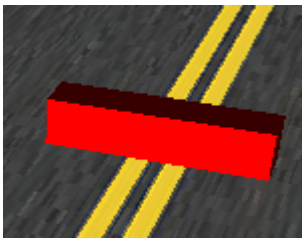
5. Window resolution



User can choose different sizes of the screen through the sub-menu. There are the following settings:

- 600: change the screen resolution to 600x600
- 800: change the screen resolution to 800x800
- 1000: change the screen resolution to 1000x1000
- full screen: change to full screen.

6. Accelerating ribbon



In the application, I use red rectangular objects to represent the accelerating ribbon. When users' cars overlap with these objects, the car will move faster within a second. In order to create the ribbon, there are following things need to be done:

1. render the object:

```
def makeDisplayLists(self):
    # set execute function for drawing inside the list
    self.displayList = glGenLists(1)
    glNewList(self.displayList, GL_COMPILE)
    glEnable(GL_COLOR_MATERIAL)
    glColor3f(1,0,0)
    glutSolidCube(self.length)
    glDisable(GL_COLOR_MATERIAL)
    glEndList()
```

```
def draw(self):
    glPushMatrix()

    glTranslatef(self.posX,self.posY,self.posZ)
    glRotatef(self.rotation,0.0,1.0,0.0)
    glScalef(self.sizeX,self.sizeY,self.sizeZ)

    glCallList(self.displayList)
    glPopMatrix()
```

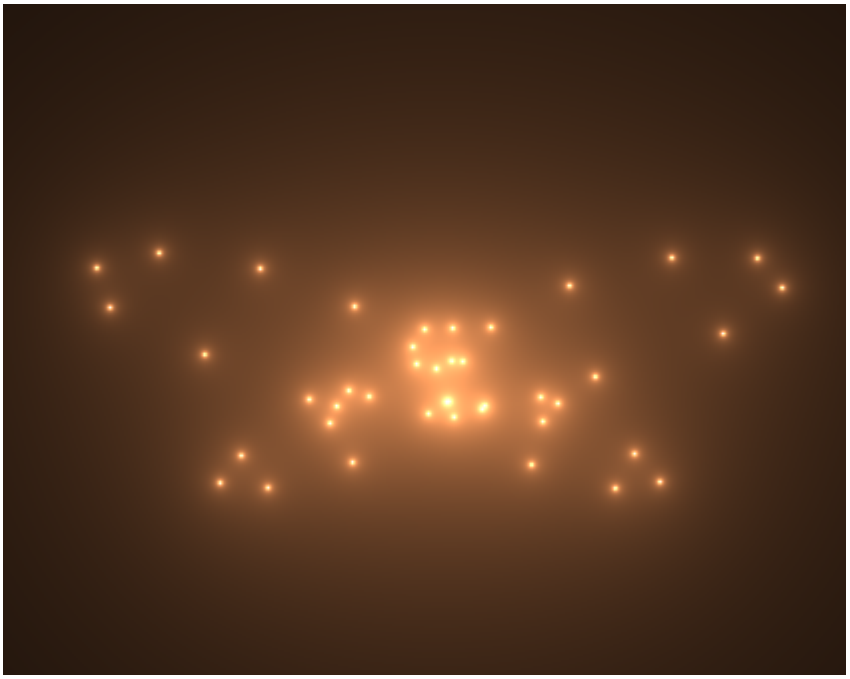
2. implement the collision check function

```
def checkOverlapped(self, pt):
    x=pt[0]
    z=pt[1]

    front=False
    back = False
    left=False
    right = False

    frontThreshold =  self.posZ - (self.length*self.sizeZ)/2
    backThreshold =  self.posZ +(self.length*self.sizeZ)/2
    leftThreshold = self.posX - (self.length*self.sizeX)/2
    RightThreshold = self.posX + (self.length*self.sizeX)/2
    if z>frontThreshold:
        front=True
    if z<backThreshold:
        back=True
    if x>leftThreshold:
        left = True
    if x<RightThreshold:
        right = True
    if(front and back and left and right):
        return True
    return False
```

3. handle the car when the car object overlaps with the ribbon

```
for ribbon in allribbons:
    if ribbon.checkOverlapped((jeepObj.posX, jeepObj.posZ)):
        print("accelerate")
        speed = 1
        t= Timer(1,resetSpeed)
        t.start()
```

Advanced Requirements



In the advanced requirements, I create the application with the use of OpenGL shading language to display the animation. To render the animation, I mainly modify the fragment shader of OpenGL which works pixel by pixel.

In order to use the fragment shader, it is required to create a program which will be sent to GPU and executed by GPU. The following picture is about creating the shader program.

```
def CompileShader(type, source):
    id = glCreateShader(type)
    glShaderSource(id, source)
    glCompileShader(id)
    result= glGetShaderiv(id,GL_COMPILE_STATUS)
    if result==GL_FALSE:
        error_msg = glGetShaderInfoLog(id)
        glDeleteShader(id)
        error_msg="\n"+error_msg.decode('utf-8')
        raise Exception(error_msg)
    return id
```

```
def CreateProgram(vertexShader,fragmentShader):
    program = glCreateProgram()
    vs = CompileShader(GL_VERTEX_SHADER,vertexShader)
    fs = CompileShader(GL_FRAGMENT_SHADER,fragmentShader)

    glAttachShader(program, vs)
    glAttachShader(program, fs)
    glLinkProgram(program)
    glDeleteShader(vs)
    glDeleteShader(fs)
    return program
```

```
shaderProgram = CreateProgram(vertexShader,fragmentShader)
global shaderProgramID
shaderProgramID  = shaderProgram
glUseProgram(shaderProgram)
```

After finishing setting up the shader, implement the fragment shader which will be passed to GPU through the above program. The following is the implementation of the fragment shader.

```
fragmentShader = """
#version 410 core
out vec4 fragColor;
uniform vec2 u_resolution;
uniform float u_time;
#define EULERVAL 2.718
#define pi 3.14
void main(){
    vec2 uv = (gl_FragCoord.xy-0.5*u_resolution.xy)/u_resolution.y;
    vec3 col = vec3(0.0);
    float r =0.1;
    float pointNums = 50 + cos(u_time/3)*10;
    for (int i =0; i< pointNums; i++){
        float step = i*(2*pi/pointNums)+(sin(u_time)+1);
        float tempVal= (pow(EULERVAL,cos(step))) -2 * cos(4*step)- pow(sin(step/12),5);
        float dx = r*sin(step)*tempVal;
        float dy = r*(max(cos(u_time/3),0.4))*cos(step)*tempVal;
        col = col + (0.001+(cos(u_time)+1)*0.0005)/length(uv-vec2(dx,dy));
    }
    /*col = col + 0.001/length(uv-vec2(0,0.2));
    col = col + 0.001/length(uv-vec2(0,-0.2));*/
    col *= abs(sin(u_time)+0.2)*vec3(0.8,0.5,0.3);
    fragColor = vec4(col,1.0);
}
"""
```

Manipulation

Users can control the animation speed through the pop-up menu by pressing the right mouse button.