Motion Parallax

Luke Kelty

Dr. Hieu Bui

11/23/2020

**Exploring Motion Parallax with OpenGL**

Introduction

In this project, a brief demonstration of the phenomenon in graphics known as motion parallax is shown using OpenGL. According to the Oxford English Dictionary, Motion Parallax is “the effect whereby the position or direction of an object appears to differ when viewed from different positions, e.g. through the viewfinder and the lens of a camera.” This creates an illusion of depth in the image. In our example, the mouse serves to change the viewpoint of the camera, and the varying distances of the objects in the fore and background are shown through the amount that they shift within the frame. A larger shift alludes to an object that is closer to the camera, and a smaller shift indicates that an object is further away. Through completing this project, I seek to demonstrate the applications of OpenGL, the phenomenon of motion parallax, and an understanding of camera and viewpoint manipulation techniques learned in the class.

Software

The only software used was Python 3.8, OpenGL, and OpenGL-accelerate. All code editing was done in Windows notepad text editor.

Methods

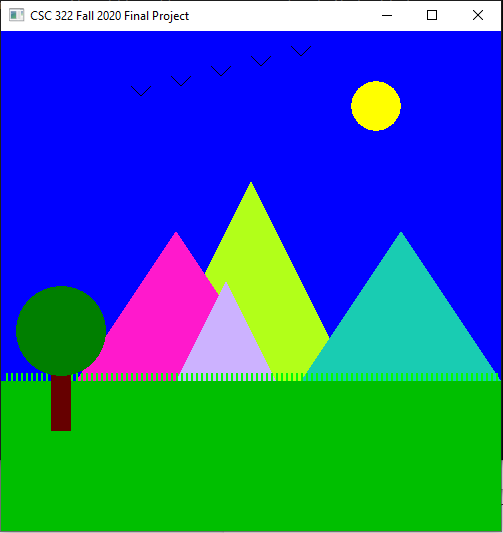
First steps were to install Python 3.8. For someone who wished to replicate this project, a link to download is <https://www.python.org/downloads/release/python-380/> Next, the OpenGL and OpenGL-accelerate packages are needed. These can be found here: <https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyopengl> After installing the appropriate packages for you based on your operating system (32 v 64 bit) and version of Python (in my case, 3.8), you need to install the packages used either the command line or the PowerShell. This can be done using the search line to find either of these interfaces, and then typing ‘pip install <file name>’. After these steps are complete, opening any .py file that imports from the OpenGL API and running it with Python 3.8 should be successful. Originally, I had wanted to code in PyCharm, a community based IDE that some of my friends had recommended to me. However, even after uninstalling and reinstalling my OpenGL files, I was not able to avoid a “attempt to call invalid function” error. As a result, my coding and code editing was done completely in notepad, which is admittedly a lot clunkier and less efficient than I would have liked. Finding simple syntax errors took much longer than I had expected.

Coding Process

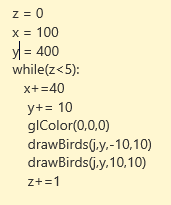
As with all coding problems, I started by drawing out a rough algorithm of the steps I would need to take to construct the final project. This began with importing bits of code from the first homework we had completed using OpenGL. This was mainly syntactical content, such as setting up a window and constructing basic shapes like squares and triangles. Then, I moved on to defining all the functions I would need to have within my draw functions in order to create my desired shapes. With these completed, I focused on creating two draw functions. One would be the initial drawing, before any parallax was applied. The other would be the parallax effect. The reason that both are necessary is that without an initial drawing, the program completes itself upon startup, as it does not detect a mouse moving in its window. Within both of these draw functions, I coded the layers of the drawing. The reason for layers is that in OpenGL, whatever is drawn first will then be drawn over by subsequent drawings. Thus, to create that illusion of depth, it is necessary that objects that are different distances away be drawn in the proper order so that they overlap properly. First was the sun, then the rear mountain. Then the two middle mountains, and then a smaller frontal mountain I added as extra. Then, there is the foreground, composed of a green square for grass, blades of grass, and a tree. Finally, I added a grouping of 5 birds, who would be unaffected by parallax. Once all the layers are in place, I could go about implementing the parallax effect. This I did by assigning each layer a factor based on the current position of the mouse (x and y coordinates - 250, so simulate a cartesian plane in the OpenGL window). This factor was then multiplied against all the starting coordinates of the shapes in each respective layer. This would cause the shapes to shift in an amount respective to their “distance” from the viewpoint. Finally, I would implement a callback function using glutPassiveMotionFunc(), which would update the image every time that the mouse moved in the window, creating the parallax effect.

Results

* **Errors and Flaws**
  + Errors
    - I could not find a way to get the birds to move across the screen without using pygame. This is because I’m fairly sure I need a timer to contain my while loop, in order to ensure that the drawing keeps refreshing. Without a timer, if I were to draw the birds inside the drawWindow and drawParallax functions, the birds would wildly shift and redraw whenever the mouse changed from being motionless to in motion and back again. I did code the birds as a still, but couldn’t keep them in the final program due to what I assume was a syntactical error I could not find because I was using Notepad.



(Still image with the 5 birds, before I had to remove them)



(Code to implement the birds, that I removed from Final Submission)

* + Flaws
    - The grass sometimes doesn’t fully fill in on the left side when the mouse moves all the way to the right of the window. I think this is because I can’t set the grass to start being drawn at a negative x location.
    - If the window does not detect mouse movement, it will revert the image back to the non-parallaxed version. This can cause some image tearing if the mouse is not kept in motion.
* **Positive Elements**
  + The extra layer of mountain provides a more detailed sense of depth and enhances the parallax effect.

Analysis

While All elements of the project were to some degree demonstrated, a lack of a firm grasp on using OpenGL, the lack of an IDE to work in, and the lack of access to Pygame served to increase the challenge of implementing all aspects of the project simultaneously. While with enough time and effort, these challenges could have been overcome, the outstanding circumstances of this online semester even further compounded this difficulty. Flaws notwithstanding, the parallax effect that was successfully demonstrated was optically convincing and well executed.

Conclusion

This project gave a fair and valid assessment of my coding proficiency, my use of OpenGL, and my understanding of the implementation of viewpoint manipulation. In the areas of OpenGL usage, my unfamiliarity was detrimental, as of the roughly 20 hours I spent on the project, more than half were dedicated to learning more about the OpenGL API, finding examples of functions from that API being implemented in code, and then bug fixing my own usage of OpenGL functions. However, as this project is even named Motion Parallax, I did strive to fully grasp this topic and implement it fully in my code. This is a camera and viewpoint manipulation technique that is essential for me moving forward with both my education and career, and having completed and saved this project, I feel confident that I would be able to take what I have learned from this experience and utilize it further down the road. All outside sources I used or referenced are cited below. Attached is a copy of the code if you wish to see the visual effect for yourself, and the .py file will be uploaded alongside this document to the github.

**GitHub Link -** <https://github.com/LukeKelty/Final-Project/>

.

**Final Code**

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from OpenGL.GLU import \*

from math import \*

import random

window = 0

width, height = 500, 500 # window size

c1 = random.randint(0,10)

c2 = random.randint(0,10)

c3 = random.randint(0,10)

c4 = random.randint(0,10)

color1 = c1/10

color2 = c2/10

color3 = c3/10

color4 = c4/10

def drawSquare(x, y, width, height):

glBegin(GL\_QUADS) # start drawing a square

glVertex2f(x, y) # bottom left point

glVertex2f(x + width, y) # bottom right point

glVertex2f(x + width, y + height) # top right point

glVertex2f(x, y + height) # top left point

glEnd() # done drawing

def drawTriangleL(x, y, width, height):

glBegin(GL\_TRIANGLES)

glVertex2f(x, y)

glVertex2f(x + width, y)

glVertex2f(x + width, y + height)

glEnd()

def drawTriangleR(x, y, width, height):

glBegin(GL\_TRIANGLES)

glVertex2f(x, y)

glVertex2f(x, y + height )

glVertex2f(x + width, y)

glEnd()

def drawBirds(x,y, width, height):

glBegin(GL\_LINES)

glVertex2f(x+width,y+height)

glVertex2f(x,y)

glEnd()

def drawSphereish(posx, posy, sides, radius):

glBegin(GL\_POLYGON)

for i in range(100):

cosine = radius \* cos(i \* 2 \* pi / sides) + posx

sine = radius \* sin(i \* 2 \* pi / sides) + posy

glVertex2f(cosine, sine)

glEnd()

def drawScene():

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT) # clear the screen

glLoadIdentity() # reset position

refresh2d(width, height)

# background

glColor3f(0, 0, 1.0) # set color to blue

drawSquare(0, 0, 500, 500)

#layer 1

glColor3f(1.0, 1.0, 0.0) # sun color yellow

drawSphereish(375, 425, 16, 25)

#layer 2

glColor3f(color1, color2, color3) # tallest mountain in back

drawTriangleL(50, 0, 250, 350)

drawTriangleR(300, 0, 250, 350)

#layer 3

glColor3f(color2, color3, color4) # left mountain

drawTriangleL(25, 0, 200, 300)

drawTriangleR(225, 0, 200, 300)

glColor3f(color3, color4, color1) # right mountain

drawTriangleL(200, 0, 200, 300)

drawTriangleR(400, 0, 200, 300)

glColor3f(color4, color1, color2) # extra front mountain

drawTriangleL(100, 0, 100, 250)

drawTriangleR(200, 0, 100, 250)

glColor3f(0.0, 0.75, 0.0) # set color to green

drawSquare(0,0 , 500, 150)

#layer 4 #grass loop

x = 0

i = 0

while (x < 160):

x += 1

i += 5

glColor3f(0.0, 1.0, 0.0)

drawSquare(i, 150, 2, 8)

glColor3f(0.4, 0.0, 0.0) # set color to brown

drawSquare(50, 100, 20, 75) # draw the trunk

glColor3f(0.0, 0.5, 0.0) # set color to green

drawSphereish(60, 200, 35, 45) # draw the leaves

glutSwapBuffers() # important for double buffering

def drawParallax(Mx, My):

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT) # clear the screen

glLoadIdentity() # reset position

refresh2d(width, height)

foreX = (Mx-250)\* 0.5

foreY = (My-250)\* 0.5

fMX = (Mx-250)\* 0.25

fMY = (My-250)\* 0.25

mMX = (Mx-250)\* 0.15

mMY = (My-250)\* 0.15

rMX = (Mx-250)\* 0.05

rMY = (My-250)\* 0.05

sX = (Mx-250)\* 0.02

sY = (My-250)\* 0.02

# background

glColor3f(0, 0, 1.0) # set color to blue

drawSquare(0, 0, 500, 500)

#layer 1

glColor3f(1.0, 1.0, 0.0) # sun color yellow

drawSphereish(375+sX, 425+sY, 16, 25)

#layer 2

glColor3f(color1, color2, color3) # tallest mountain in back

drawTriangleL(50+rMX, rMY, 250, 350)

drawTriangleR(300+rMX, rMY, 250, 350)

#layer 3

glColor3f(color2, color3, color4) # left mountain

drawTriangleL(mMX+25, mMY, 200, 300)

drawTriangleR(mMX+225, mMY, 200, 300)

glColor3f(color3, color4, color1) # right mountain

drawTriangleL(mMX+200, mMY, 200, 300)

drawTriangleR(mMX+400, mMY, 200, 300)

glColor3f(color4, color1, color2) # extra front mountain

drawTriangleL(fMX+100, fMY, 100, 250)

drawTriangleR(fMX+200, fMY, 100, 250)

glColor3f(0.0, 0.75, 0.0) # set color to green

drawSquare(0, 0, 500, 150+foreY)

#layer 4 #grass loop

x = 0

i = -10

while (x < 140):

x += 1

i += 5

glColor3f(0.0, 0.75, 0.0)

drawSquare(i+foreX, 150+foreY, 2, 8)

glColor3f(0.4, 0.0, 0.0) # set color to brown

drawSquare(50+foreX, 100+foreY, 20, 75) # draw the trunk

glColor3f(0.0, 0.5, 0.0) # set color to green

drawSphereish(60+foreX, 200+foreY, 35, 45) # draw the leaves

glutSwapBuffers() # important for double buffering

def refresh2d(width, height):

glViewport(0, 0, width, height)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

glOrtho(0.0, width, 0.0, height, 0.0, 1.0)

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity()

# initialization

glutInit() # initialize glut

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE)

glutInitWindowSize(width, height) # set window size

glutInitWindowPosition(0, 0) # set window position

wind = glutCreateWindow("CSC 322 Fall 2020 Final Project") # create window with title

glutDisplayFunc(drawScene) # set showScreen function callback

glutIdleFunc(drawScene) # draw all the time

glutPassiveMotionFunc(drawParallax);

glutMainLoop() # start everything

Sources

“Parallax”*, OED Online*, Oxford University Press, September 2020, www.oed.com/viewdictionaryentry/Entry/11125. Accessed 23 November 2020.

*OpenGL.org*, edited by Mark Kilgard, 23 Feb. 1996, [www.opengl.org/resources/libraries/glut/spec3/node51.html](http://www.opengl.org/resources/libraries/glut/spec3/node51.html).

Kent State University Lecture Slides on the Topic of Game Engines (specifically OpenGL and glut usage) -

<http://www.cs.kent.edu/~ruttan/GameEngines/lectures/glut.pdf>

StackOverflow Query related to the question of obtaining a mouse location within an OpenGL window - <https://stackoverflow.com/questions/9912746/using-glutpassivemotionfunc-in-glut>