

Scrum Update IV (YANNIX)

☰ Tags	Scrum Update
🕒 Create	@March 21, 2022 10:28 AM
🔍 Nickname	
🕒 Last edited time	@March 30, 2022 1:09 PM
🔍 Site	
➦ Student	
🔗 URL	
☰ Scrum Update	[Last week] - ศึกษาการนำ pyFLTK รวมกับ OpenGL - ศึกษา Model view projection matrix ของ OpenGL - ศึกษาวิธีการทำ Segmentation ด้วย implicit function [This week] - Design UI with Main Feature - ดูเรื่องการ Segmentation + implementation -> Region Growing Segmentation [Blocks] - การบ้านที่สั่งในอาทิตย์นี้ - อ่าน Paper และทำ Presentation วิชา deep learning
➦ Week	
➦ Scores	
📌 Status	

Resource:

MVP Matrices

Projection matrix: <https://www.youtube.com/watch?v=xZs6K7VLM7A>

<https://www.youtube.com/watch?v=xZs6K7VLM7A>

MVP Matrices: https://www.youtube.com/watch?v=x_Ph2cuEWrE

https://www.youtube.com/watch?v=x_Ph2cuEWRE

pyFLTK-OpenGL

pyFLTK: https://pyfltk.sourceforge.io/docs/CH8_Opengl.html#opengl


My Note:

ในอาทิตย์นี้จะเน้นไปในการศึกษาและการจด Note แต่ก็ยังมีการเขียน Code อยู่ซึ่งจะเป็นเรื่องของ การรวม pyFLTK-OpenGL และ MVP Matrices ซึ่ง code จะอยู่ใน Github folder Integration และ OpenGL > Projection matrices


Code source: <https://github.com/use555555/Onboard/tree/main/Basic>

Onboard/Basic at main · use555555/Onboard


Contribute to use555555/Onboard development by creating an account on GitHub.

 <https://github.com/use555555/Onboard/tree/main/Basic>

use555555/
Onboard



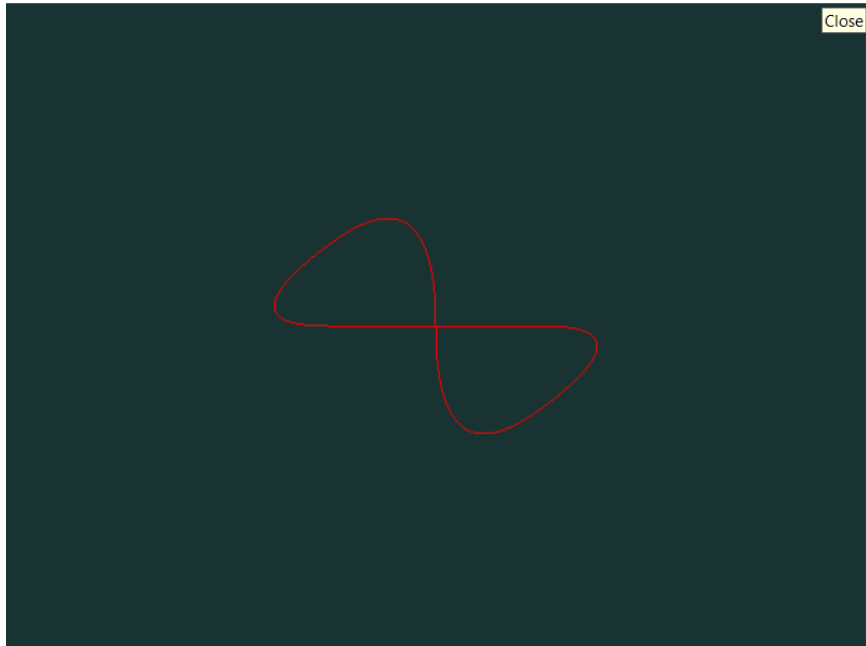
1 Contributor
0 Issues
0 Stars
0 Forks



B-spline curve

ในส่วนของ B-spline นี้ผมขอสรุปในส่วนของการ Generate knot vector ที่ในอาทิตย์ที่แล้วตัว Code เองนั้นยังผิดอยู่ แต่ในอาทิตย์นี้ได้ทำความเข้าใจใหม่และแก้ไข Code แล้วซึ่งมีอยู่ว่าการที่ตัว knot vector นั้นจะกำหนดหน้าตาของ curve ที่ออกมาซึ่งจะทำให้เกิดชนิดของ curve จากการกำหนด knot vector ซึ่งก็จะมี รูปแบบของ Curve ที่พบบ่อยๆ อยู่ 2 แบบ คือ

1. Uniform b-spline curve ซึ่งเกิดจากการกำหนด knot vector ที่มีอัตราส่วนเท่าๆกันเช่น (0, 1, 2) ซึ่ง curve จะมีหน้าตาแบบที่มีส่วน โค้งที่เท่าๆ กันซึ่งโค้งจะไม่ค่อยมีการเบ้ในทิศทางใดๆ ก็จะมีรูปดังตัวอย่าง



จากรูปจะเป็น b-spline สมการกำลังสองและกำหนด control point คือ $(-0.5, 0.0)$, $(0.0, 0.5)$, $(0.0, -0.5)$

และ $(0.5, 0.0)$ ซึ่งจะเห็นได้ว่า curve นั้นจะมีการวนกลับมาจุดเริ่มต้น

2. Open uniform b-spline curve จะเกิดจากการกำหนด knot vector แบบที่จะมีตัวซ้ำ หัก
 ท้ายตามกำลังที่จะกำหนดในสมการเช่นถ้ากำหนด กำลังสองและมี control point 4 จุดจะมี
 การกำหนด knot vector ว่า $(0, 0, 0, 1, 2, 2, 2)$ และถ้าเป็นกำลังสามจะมีการกำหนด
 knot vector ว่า $(0, 0, 0, 0, 1, 1, 1, 1)$ ซึ่งทั้งสองก็จะมีหน้าตาดังรูป
 สมการกำลังสอง



สมการกำลังสาม



จากรูปจะเป็น b-spline สมการกำลังสองและกำหนด control point คือ $(-0.5, 0.0)$, $(0.0, 0.5)$, $(0.0, -0.5)$

และ $(0.5, 0.0)$ ซึ่งจะเห็นได้ว่า curve นั้นจะไม่มี การวนกลับหาจุดเริ่มต้น และจะมีหน้าตาคล้ายกับ Bezier ที่

มีกำลังเท่ากัน

PyFLTK-OpenGL

ส่วนนี้จากการที่ได้ไปทำการอ่านตัว Document มานั้นผมสามารถสรุปได้ว่าการที่จะเขียนรวม OpenGL และ FLTK นั้นคือการที่เราเขียน widget ของ FLTK ที่ใช้ function ของ OpenGL ในการ Render สิ่งของลงไปบน Widget นั้น โดยการที่จะเขียนจะต้องเริ่มจากการสร้าง class ที่มีการ inherit window ของ FLTK

```
class MyWindow(Fl_Gl_Window):  
    def __init__( self, xpos, ypos, width, height, label ):  
        Fl_Gl_Window.__init__( self, xpos, ypos, width, height, label )
```

จากนั้นก็จะทำการสร้าง Function ของ class นี้ซึ่งหลักๆ ในการรวมก็จะมี draw ที่จะใช้ในการวาดบน Widget ที่จะมีการใช้ OpenGL อยู่ใน function นี้

```

def draw(self):
    if len( self.coordinate ) >= 4:
        self.shader = self.createShader( "shaders/vertex.txt", "shaders/fragment.txt" )
        glUseProgram(self.shader)

        self.bezierStart = Bezier( self.coordinate, self.resolution )

        # Creating projection matrix
        projectionTransform = pyrr.matrix44.create_orthogonal_projection_matrix( 0.0, self.width, 0.0, self.height,
                                                                                 0.1, 100, np.float32 )
        # Send the projection matrix to be use in shader
        glUniformMatrix4fv( glGetUniformLocation( self.shader, "projection" ), 1, GL_FALSE, projectionTransform )

        # Creating model matrix
        modelTransform = pyrr.matrix44.create_from_translation( pyrr.Vector3( [ 0.0, 0.0, -1.0 ] ) )
        # Send the model matrix to be use in shader
        glUniformMatrix4fv( glGetUniformLocation( self.shader, "model" ), 1, GL_FALSE, modelTransform )

        # refresh screen
        glClear( GL_COLOR_BUFFER_BIT )
        # COLOR_BUFFER = Big array storing color value on the screen
        # Every pixel color in OpenGL is stored in 32 bit u_int

        # Draw shape
        glUseProgram( self.shader )
        glBindVertexArray( self.bezierStart.vao )

        # Draw data from array
        glDrawArrays( GL_LINE_STRIP, 0, self.bezierStart.vertexCount )
        # glDrawArrays(shape, initial point, number of point to draw)
    else:
        glClear( GL_COLOR_BUFFER_BIT )

    pass

```

จะเห็นได้ว่าการใส่ Function ต่างๆที่ผมได้ใช้ในตอนเขียน OpenGL จาก code ข้างต้นผมจะใช้เป็น Bezier curve ที่ผมเคยทำไว้

นอกจาก function draw แล้วนั้นก็จะมีอีก function หนึ่งคือ handle ที่จะใช้ในการ handle event ที่จะเป็นการตอบสนองต่างๆเมื่อเรากำกับ widget นั้นเช่นเวลาที่เรากดบน widget หรือกดแล้วลาก ฯลฯจากที่ผมได้ไปลองเขียนมาดังภาพ

```

def handle(self, event):
    if event == FL_PUSH:
        if self.collectingData == 1:
            self.coordinate += ( Fl.event_x(), self.height - Fl.event_y() )
            print(self.coordinate)
            self.redraw()
            return 1
        else:
            return FL_Gl_Window.handle(self, event)

```

จากภาพผมได้ใช้เป็นการกดบนหน้าจอเพื่อเก็บพิกัดที่กดแล้วให้นำไปใช้ในการคำนวณ curve

หลังจากที่เราได้สร้าง class ที่ใช้ในการสร้าง widget แล้วนั้นเราก็จะต้องทำการสร้างตัว App ตามที่เคยเขียนไปตอนที่ศึกษา FLTK ในช่วงก่อนหน้านี้ดังภาพ

```
##### SetUp #####
window = FL_Window(860,556)      ###Creating Windows
window.label("Height Estimator")  ###Name the Windows

windowGL = MyWindow( 220, 76, 640, 480, "GLWindow" )
#####
box = FL_Box( 0, 0, 860, 76, "Curve Drawer Program" )  ###Creating box (x,y,width,height)
box.box(FL_UP_BOX)  ### Draw box
box.labelsize(36)
box.labelfont(FL_BOLD+FL_ITALIC)
box.labeltype(FL_SHADOW_LABEL)
#####
button = FL_Button( 0, 76, 220, 40, "Draw" )  ###(x, y, width, height, "label")
button.type( FL_NORMAL_BUTTON )  ###Normal Button
button.color( FL_GREEN )  ### Set Color
button.color2( FL_DARK_GREEN )  ### Set Color
button.when( FL_WHEN_RELEASE )  ###Set the action
button.callback(callback)
#####
clearButton = FL_Button( 0, 116, 220, 40, "Clear" )  ###(x, y, width, height, "label")
clearButton.type( FL_NORMAL_BUTTON )  ###Normal Button
clearButton.color( FL_WHITE )  ### Set Color
clearButton.color2( FL_GRAY )  ### Set Color
clearButton.when( FL_WHEN_RELEASE )  ###Set the action
clearButton.callback(callback)
#####
FL.event_button()
#####
window.end()  ### ปิดหน้าต่างเก่า
window.show( sys.argv )  ### แสดงผลหน้าต่างใหม่
#####
FL.run()  ###run the program
```

ซึ่งล่าสุดที่ได้เขียนไปก็สามารถทำได้ตาม Video ข้างล่างนี้

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/3f610cee-df48-4567-b98e-fd7e79c4fb86/FLTK-OpenGL.mp4>

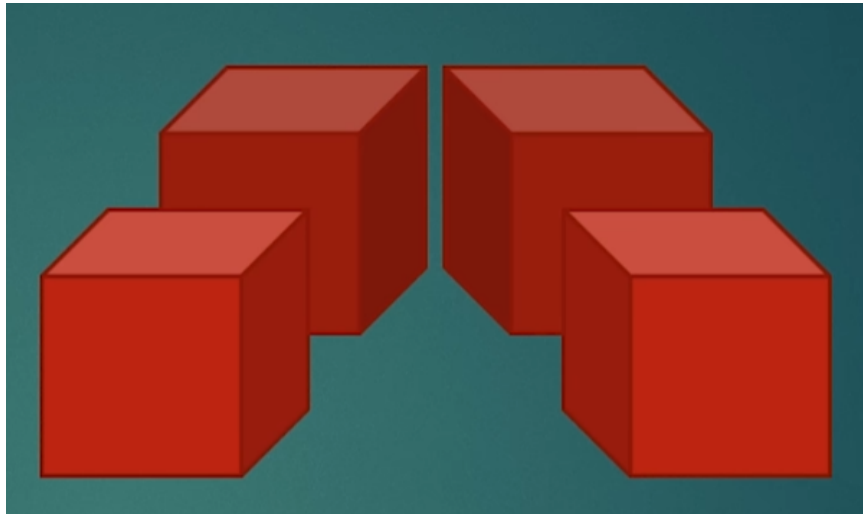
MVP Matrices

ในส่วนของ MVP Matrices นี้ผมจะต้องขอเริ่มจากการอธิบาย Projection matrix ก่อน

ซึ่ง Projection matrix คือการที่เรานำ Transformation matrix มาทำการ map coordinate ใน 2D หรือ 3D ทำการ map มาเป็น coordinate บนหน้าจอคอมพิวเตอร์ ซึ่งใน 3D ตัว projection matrix ก็จะทำให้เกิดเหตุการณ์ที่ว่าเมื่อสิ่งของใน world นั้นอยู่ไกลจะทำให้ object ที่อยู่ห่างจะมีขนาดเล็กลง และ object ที่อยู่ใกล้จะมีขนาดใหญ่ขึ้น ซึ่ง projection นี้ก็มีอยู่ 2 ชนิดคือ

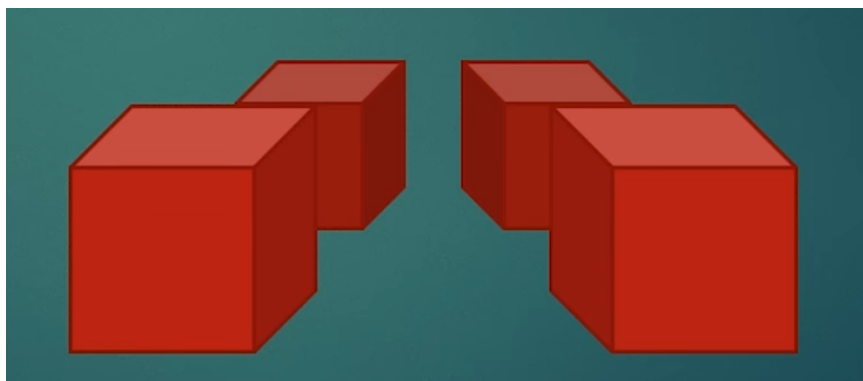
1. Orthographic (มักจะใช้ใน 2D)

ส่วนนี้ในเรื่องของความลึกหรือความห่างของ object จะไม่มีการส่งผลกับขนาดของ object



2. Perspective (มักจะใช้ใน 3D)

ส่วนนี้ในเรื่องของความลึกหรือความห่างของ object จะมีการส่งผลกับขนาดของ object



ซึ่งถ้านำเรื่อง Projection matrix มาพูดถึง MVP matrices ก็สามารถพูดได้ว่า Projection matrix เป็นส่วนหนึ่งใน MVP matrices เนื่องจาก MVP matrices หรือ Model view projection matrices คือการที่เราทำ Transformation สามส่วนซึ่งก็คือ

1. Model matrix: รูปแบบที่เราทำการ simulate object นั้น (Position, Transformation ของ object)

2. View matrix: มุมมองของกล้อง (Position, Transformation ของ กล้อง)

3. Projection matrix: ตามที่ได้กล่าวมาข้างต้น

ซึ่งแต่ละ matrix ก็มีขนาด 4*4 ซึ่งจะมีการมาคูณในลำดับแบบ M*V*P แต่ใน OpenGL มีการใช้ข้อมูลแบบ Column major ordering ทำให้การคูณจะเรียงแบบ P*V*M เมื่อเราทำการคูณเสร็จแล้วเราจะได้ vertex position บนหน้าจอ

ในส่วนนี้ผมได้นำไปใช้ในตอนที่เขียนเพื่อรวม FLTK กับ OpenGL ซึ่งทำให้ง่ายในการเก็บข้อมูลที่ตอนแรก input เป็นช่วง -1 ถึง 1 เปลี่ยนให้สามารถใช้ input เป็น ค่าพิกัด pixel ที่กดบนหน้าจอ

Segmentation

ในส่วนนี้ผมเน้นไปทำความเข้าใจจากสิ่งที่ Advisor ได้แนะนำมา ซึ่งจะเป็นการทำ Segmentation แบบ implicit คือการทำ segmentation จากสมการที่เราใช้ในการวาดบนรูป แต่การที่เราจะทำได้ นั้นเราต้องมีสมการที่สามารถใช้ในพิกัด x, y ได้ซึ่งถ้ายกตัวอย่างก็จะเป็นการใช้สมการวงกลมในการดูซึ่งมีสมการคือ

$(x - c_x)^2 + (y - c_y)^2 - r^2 = 0$ ถ้าเรากำหนดว่าเรามีจุดศูนย์กลางอยู่ที่ (0, 0) และมีรัศมี 5 หน่วย ถ้าเราจะเช็คพิกัด (1, 0) และ (0, 10) อยู่ในพื้นที่ยังวงกลมหรือไม่ ถ้าเราดูที่สมการแล้วเราใช้พิกัด (5, 0) ซึ่งเป็นจุดที่อยู่บนเส้นรอบวงพอดีเมื่อแทนในสมการจะได้ $(5 - 0)^2 + (0 - 0)^2 - 5^2 = 0$ แต่ถ้าเราลองนำพิกัด (1, 0) ที่ควรจะอยู่ในพื้นที่ยังวงกลมมาแทนในสมการ $(1 - 0)^2 + (0 - 0)^2 - 5^2 = -24$ จะเห็นได้ว่าค่าที่ออกมาจะเป็นค่าลบ และถ้าเราลองนำพิกัด (0, 10) ที่ควรจะอยู่นอกพื้นที่ยังวงกลมมาแทนในสมการ $(0 - 0)^2 + (10 - 0)^2 - 5^2 = 75$ จะเห็นได้ว่าค่าที่ออกมาจะเป็นค่าบวก ถ้าสรุปสมการวงกลมจากที่เห็นออกมาเลยก็คือเราสามารถดูได้ว่าพิกัดใดอยู่นอกหรือในวงกลมจากการหาค่าสมการโดยการสร้าง condition จากที่เราดูจากการคำนวณบนสมการได้