Chapter 2

OpenGL and GLUT, Drawing Geometric Primitives

อ้างอิงและดัดแปลงจากสไลด์ของ อ.ประมุข ขันเงิน

ผู้สอน ชาคริต วัชโรภาส

OPENGL AND GLUT

↳ วาดรูปทรง

↳ อัลกอริทึมเป็นต้น

OpenGL

- Application Programming Interface (API) สำหรับควบคุม GPU
- ผู้ใช้ OpenGL ระบุรูปทรงและรูปร่างพื้นฐาน (จุด เส้น และรูปหลายเหลี่ยม) ผ่านทาง OpenGL
- OpenGL จะทำหน้าที่สร้างภาพไว้บน framebuffer ให้
- ใช้สร้างโปรแกรมที่มีการตอบสนองต่อผู้ใช้แบบทันทีทันควัน (interactive) และโปรแกรมที่มีภาพเคลื่อนไหว
- ทำหน้าที่เดียวกับ Direct3D และเป็นคู่แข่งทางการค้ากันอยู่

คำศัพท์

- Bitplane
  - เนื้อที่ในหน่วยความจำที่เก็บข้อมูล 1 บิทของทุกพิกเซลที่อยู่บนจอภาพ
- Framebuffer
  - Bitplane หลายๆ bitplane ที่เก็บข้อมูลทั้งหมดที่ใช้แสดงบนหน้าจอ
- Buffer
  - Bitplane กลุ่มหนึ่งที่ใช้เก็บข้อมูลบางอย่าง
- Application Programming Interface (API)
  - ฟังก์ชันและ object อื่นๆ ในภาษาระดับสูงที่ให้โปรแกรมประยุกต์ใช้สำหรับติดต่อกับระบบฮาร์ดแวร์หรือซอฟต์แวร์ต่างๆ

สิ่งที่ OpenGL ไม่ทำ

- จัดการการติดต่อกับผู้ใช้
- จัดการวินโดว์
- วาดและจัดการรูปทรงที่ซับซ้อน เช่น รถถัง ต้นไม้ ฯลฯ
  - ถึงแม้ว่าคุณจะสามารถใช้รูปทรงง่ายๆ ของ OpenGL สร้างมันได้ก็ตาม
  - ส่วนใหญ่คุณต้องเขียน library ขึ้นมาจัดการกับพวกนี้เอง
- จัดการ framebuffer
  - เป็นความรับผิดชอบของคุณที่ต้องเตรียม framebuffer ให้ OpenGL

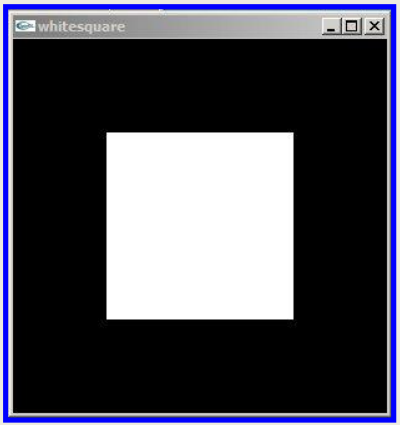
GLUT

- OpenGL Utility Toolkit
- ใช้สำหรับจัดการการติดต่อกับผู้ใช้และจัดการวินโดว์
  - ทำสิ่งที่ OpenGL ไม่ทำ
- เอาไปใช้เขียนโปรแกรมประยุกต์จริงๆ คงยาก
  - ไม่มี GUI Widget ให้ใช้เลย
  - ต้องรับข้อมูลจากผู้ใช้ตามที่ GLUT กำหนด
- แต่ทำให้การเรียนรู้ OpenGL ง่ายขึ้นมาก

ตัวอย่าง

```
import sys
from OpenGL.GL import *
from OpenGL.GLUT import *
from  openGL . GLU. import
def draw():
    glClearColor(0.0, 0.0, 0.0, 0.0)
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1.0, 1.0, 1.0)
    glBegin(GL_POLYGON)
    glVertex3f(-0.5, -0.5, 0.0)
    glVertex3f( 0.5, -0.5, 0.0)
    glVertex3f( 0.5,  0.5, 0.0)
    glVertex3f(-0.5,  0.5, 0.0)
    glEnd()
    glFlush()
```

OpenGL




```
glutInit(sys.argv)
glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE)
glutCreateWindow("whitesquare")
glutDisplayFunc(draw)
glutMainLoop()
```

GLUT

เฉพาะส่วนของ OpenGL

```
glClearColor(0.0, 0.0, 0.0, 0.0)
glClear(GL_COLOR_BUFFER_BIT)
glColor3f(1.0, 1.0, 1.0)
glBegin(GL_POLYGON)
glVertex3f(-0.5, -0.5, 0.0)
glVertex3f( 0.5, -0.5, 0.0)
glVertex3f( 0.5,  0.5, 0.0)
glVertex3f(-0.5,  0.5, 0.0)
glEnd()
glFlush()
```

→ color ที่อยู่ใน buffer ถูกเคลียร์

→ state machine

ทีละคำสั่ง

- glClearColor(0.0, 0.0, 0.0, 0.0)
  - กำหนดสีที่จะใช้ล้างหน้าจอ โดยให้เป็นสีดำ
- glClear(GL_COLOR_BUFFER_BIT)
  - ล้าง bitplane ที่เก็บสีด้วยสีทีกำหนดใน glClearColor
- glColor3f(1.0, 1.0, 1.0)
  - เปลี่ยนสีเป็นสีขาว
  - จุดที่วาดต่อจากนี้ไปจะเป็นสีขาว

ทีละคำสั่ง (ต่อ)

- glBegin(GL_POLYGON)
  - บอกว่าต่อไปเราจะวาดรูปหลายเหลี่ยม
- glVertex3f(x, y, z)
  - กำหนดจุด
- glEnd()
  - บอกว่าสิ่งที่เริ่มไปตั้งแต่ glBegin ที่แล้วได้เสร็จสิ้นแล้ว
  - ในที่นี้คือบอกว่ากำหนดรูปหลายเหลี่ยมเสร็จแล้ว
- glFlush()
  - ทำให้คำสั่ง OpenGL ที่เคยสั่งมาถูกนำไปปฏิบัติงาน แทนที่จะถูกเก็บไว้ในหน่วยความจำเพื่อรอคำสั่งอื่น

คำสั่ง OpenGL

- ขึ้นต้นด้วย gl
- ตามด้วยชื่อคำสั่ง เช่น Vertex หรือ Color
- บางคำสั่งอาจมี*จำนวนและชนิด*ของ argument
  - 3f บอกว่าต้องการ argument เป็น float 3 ตัว
    - glVertex3f(1.0, 3.0, 4.0);
  - 2i บอกว่าต้องการ argument เป็น int 2 ตัว
    - glVertex2i(-1, 5);
  - 3fv บอกว่าต้องการ argument เป็น pointer ไปยัง float 3 ตัว
    - ตัวอย่างในภาษา C

      ```
      float colorArray[] = {1.0f, 0.0f, 0.0f};
      glColor3fv(colorArray);
      ```
    - ตัวอย่างในภาษา Python (สามารถใช้ได้กับ tuple หรือ list)

      ```
      q = [1.5, 2.5, 3.0]
      glVertex3fv(q)
      ```

ชนิดของ argument ในชื่อคำสั่ง

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned long	GLuint, GLenum, GLbitfield

OpenGL เป็น State Machine

- OpenGL จะจำค่าต่างๆ ที่ผู้ใช้กำหนดได้เอาไว้
- เมื่อผู้ใช้กำหนดค่า ค่านั้นจะถูกใช้ต่อไปเรื่อยๆ จนกว่าจะเปลี่ยน
- ค่าที่จำไว้ เช่น
  - สีที่ใช้ล้างหน้าจอ
  - สีของจุด
  - ทิศทางและตำแหน่งของกล้องถ่ายรูป
- ยกตัวอย่างเช่น เวลาเราเรียก glColor3f(1,1,1) แล้วสีของจุดที่กำหนดด้วย glVertex จะเป็นสีขาวไปจนกว่าจะเรียก glColor ใหม่อีกครั้ง

โค้ดตัวอย่างเฉพาะส่วนของ GLUT

```
glutInit(sys.argv)
glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE)
glutCreateWindow("whitesquare")
glutDisplayFunc(draw)
glutMainLoop()
```

glutInit

- glutInit(sys.argv)
  - ทำการตั้งค่าเริ่มต้นหลายๆ ค่าของ GLUT
  - สิ่งที่ต้องส่งให้คือ list ไปยังจำนวน argument ของโปรแกรม และ argument อื่นๆ
  - ต้องเรียกเป็นคำสั่งแรกก่อนคำสั่งอื่นของ GLUT ทั้งหมด
  - ความจริงไม่มีอะไรมาก ปกติเราแค่ให้เรียก glutInit(sys.argv) เป็นคำสั่งแรกก็พอ

glutInitDisplay

- glutInitDisplay(mode)
  - เลือกว่าสีของ pixel ในโปรแกรมของเราจะเป็นแบบใด
    - มีให้เลือกสองแบบคือ RGB กับ Indexed Color
    - เราจะไม่ใช้ Indexed Color เลย
  - เลือกว่าจะใช้ single buffer หรือ double buffer
    - ใช้ double buffer จะทำให้ animation ดูลื่นไหลกว่า
  - เลือกว่าจะให้มี buffer อื่นๆ นอกจาก buffer สีอะไรบ้าง
    - ปกติจะใช้แค่ depth buffer สำหรับเก็บความลึกของจุดแต่ละจุด
  - ค่า mode เกิดจากการเอาค่าคงที่ของตัวเลือกต่างๆ มา or กัน
    - ปกติเราจะใช้ GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH
    - กรณีของ code ตัวอย่างใช้ GLUT_RGBA | GLUT_SINGLE

คำสั่งสำหรับจัดการวินโดว์

- glutCreateWindow(title)
  - สร้างวินโดว์ที่มี title เป็น string ที่ให้
- glutInitWindowPosition(x, y)
  - กำหนดตำแหน่งขอบบนของวินโดว์
- glutInitWindowSize(width, height)
  - กำหนดขนาดของวินโดว์

glutDisplayFunc

- glutDisplayFunc(display_callback)
  - กำหนดฟังก์ชันที่ GLUT จะเรียกทุกครั้งเมื่อมันต้องวาดหน้าจอใหม่
  - ฟังก์ชันที่จะส่งให้ glutDisplayFunc ต้องมีการนิยาม
    def <ชื่อฟังก์ชัน>()
  - ยกตัวอย่างเช่นฟังก์ชัน draw() ในโค้ดตัวอย่าง
  - ฟังก์ชันนี้ส่วนมากจะเต็มไปด้วยคำสั่ง OpenGL

glutMainLoop

- glutMainLoop()
  - ฟังก์ชันสุดท้ายที่เราเรียกในโปรแกรม
  - สั่งให้ GLUT ไปทำงานของมัน
  - งานของ GLUT
    - รับ input จากผู้ใช้
    - เรียกฟังก์ชันที่ให้ใน glutDisplayFunc
    - เริ่มต้นใหม่อีกครั้ง
  - ระวัง: ต้องสร้าง windows และกำหนด displayFunc ให้เรียบร้อยก่อนเรียก glutMainLoop

การดูเวอร์ชัน OpenGL บนเครื่องที่ใช้

```
lists = [['Vendor', GL_VENDOR],
         ['Renderer',GL_RENDERER],
         ['OpenGL Version', GL_VERSION],
         ['GLSL Version', GL_SHADING_LANGUAGE_VERSION]]

for x in lists:
    print('{0}: {1}'.format(
        x[0], glGetString(x[1]).decode("utf-8")))
```

วัตถุเรขาคณิตใน OPENGL

วัตถุเรขาคณิตใน OpenGL

- OpenGL สามารถวาดวัตถุเรขาคณิตง่ายๆ ได้ 3 อย่าง
  - จุด
  - ส่วนของเส้นตรง
  - รูปหลายเหลี่ยม
- ไม่สามารถวาดเส้นโค้งหรือพื้นผิวโค้งได้
- แต่เราสามารถวาดเส้นโค้งด้วยการวาดเส้นตรงสั้นๆ หลายเส้น

Vertex

- การกำหนดวัตถุทางเรขาคณิตใน OpenGL ทำได้โดยการกำหนด vertex หรือ “จุดมุม” ของวัตถุนั้น
  - จุดใน OpenGL มี 1 vertex
  - ส่วนของเส้นตรงใน OpenGL มี 2 vertices (เพราะส่วนของเส้นตรงเกิดจากการลากเส้นเชื่อมจุดสองจุด)
  - รูปหลายเหลี่ยมมีจำนวน vertex เท่ากับจำนวนเหลี่ยม
    - สามเหลี่ยมมี 3 vertices
    - สี่เหลี่ยมมี 4 vertices
    - n เหลี่ยมมี n vertices

glVertex

- glVertex[234][sifd][v](TYPE coords)
  - ใช้กำหนดตำแหน่ง vertex
  - สามารถมี argument 2, 3, หรือ 4 ตัวก็ได้
    - สี่ตัวตรงกับพิกัดแนว x, y, z, w
      - เราจะพูดถึงพิกัดแนว w ในสัปดาห์หน้าเมื่อเรียนเรื่อง homogeneous coordinate
    - ถ้ามี 3 ตัว จะเข้าใจว่า w = 1
    - ถ้ามี 2 ตัว จะเข้าใจว่า z = 0 และ w = 1
    - ปกติจะใช้ argument แค่ 3 ตัว
  - ตัวอย่าง:
    - glVertex2i(10, 5)
    - glVertex3d(8, 7, 3.14153265)

glVertex (ต่อ)

- เติม v ถ้าต้องการให้ argument เป็น pointer ไปยัง array ของพิกัด
  - ตัวอย่างในภาษา C
  - GLint p0[] = {1,2,3};

    glVertex3iv(p0);
  - GLfloat *p1 = {2.0f, 3.0f, 4.0f, 5.0f};

    glVertex4fv(p1)
  - ตัวอย่างในภาษา Python (สามารถใช้ได้กับ tuple หรือ list)
  - q = [1.5, 2.5, 3.0]

    glVertex3fv(q)

การกำหนดวัตถุเรขาคณิต

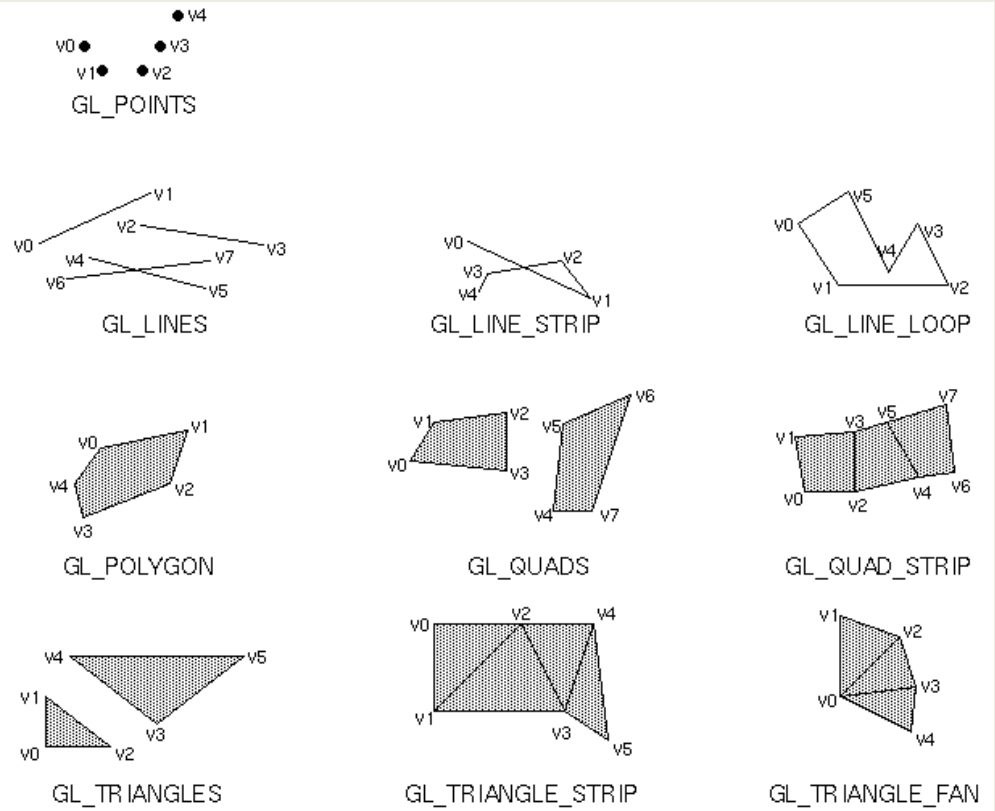
- เริ่มด้วย glBegin(ชนิดของวัตถุ)
- หลังจากนั้นใช้ glVertex กำหนด vertex ของวัตถุนั้น
- แล้วจบด้วย glEnd()
- ที่เคยเห็นมาจากการบรรยายที่ผ่านมา:

```
glBegin(GL_POLYGON)
glVertex3f(-0.5, -0.5, 0.0)
glVertex3f( 0.5, -0.5, 0.0)
glVertex3f( 0.5,  0.5, 0.0)
glVertex3f(-0.5,  0.5, 0.0)
glEnd()
```

ชนิดของวัตถุ

ค่าที่เอาไปใส่ใน glBegin(...)	ชนิดของวัตถุ
GL_POINTS	จุด
GL_LINES	ส่วนของเส้นตรง
GL_LINE_STRIP	ส่วนของเส้นตรงต่อกันหลายเส้น ปลายเปิด
GL_LINE_LOOP	ส่วนของเส้นตรงต่อกันหลายเส้น ปลายปิด
GL_TRIANGLES	สามเหลี่ยม
GL_TRIANGLE_STRIP	สามเหลี่ยมต่อกันเป็นสาย
GL_TRIANGLE_FAN	สามเหลี่ยมต่อกันเป็นรูปพัด
GL_QUADS	สี่เหลี่ยม
GL_QUAD_STRIP	สี่เหลี่ยมต่อกันเป็นสาย
GL_POLYGON	รูปหลายเหลี่ยม

ชนิดของวัตถุ (ต่อ)



glColor

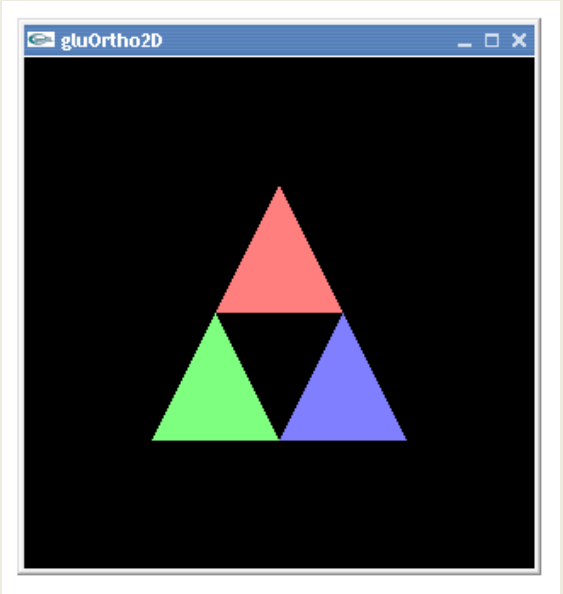
- glColor[34][fd][v](colors)
  - ใช้กำหนดสีให้กับ vertex
  - กำหนดให้แล้ว vertex จะมีสีนั้นไปจนกว่าจะเรียน glColor เพื่อเปลี่ยนมัน
  - สามารถมี argument 3, หรือ 4 ตัวก็ได้
    - Argument คือ r (สีแดง), g (สีเขียว), b (สีน้ำเงิน), a (ความโปร่งแสง)
    - แต่ละตัวมีค่าตั้งแต่ 0.0 (ไม่มีความเข้มเลย) ถึง 1.0 (เข้มเต็มที่)
    - ถ้ามี argument สามตัว a จะมีค่าเท่ากับ 1.0 (ทึบแสง)
  - ตัวอย่าง
    - glColor3f(1.0, 1.0, 0.0) = สีเหลือง
    - glColor4d(0.5, 0.5, 0.5, 0.5) = สีเทา โปร่งใส่ 50%

glColor (ต่อ)

- เติม v ถ้าต้องการให้ argument เป็น pointer ไปยัง array ของสี
  - ตัวอย่างในภาษา C
  - GLdouble color0[] = {0,1,1};
    gColor3dv(p0);
  - GLfloat *color1 = {0.1f, 0.9f, 0.5f, 0.75f};
    glVertex4fv(p1)
  - ตัวอย่างในภาษา Python
  - color2 = (0.5, 0.5, 0.5, 0.25)
    glVertex4fv(color2)
- ความโปร่งแสงจะไม่มีผลจนกว่าเราจะบอก OpenGL ให้จัดการความโปร่งแสงให้ (เรื่องนี้เราจะเรียนรู้ในเนื้อหาถัดๆ ไป)

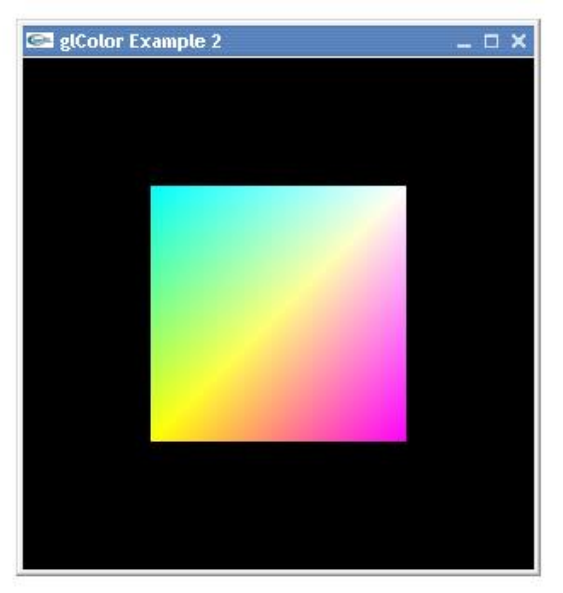
ตัวอย่าง

```
glBegin(GL_TRIANGLES)

# Red
glColor3f(1.0, 0.5, 0.5)
glVertex3f( 0.0,  0.5, 0.0)
glVertex3f(-0.25, 0.0, 0.0)
glVertex3f( 0.25, 0.0, 0.0)

# Green
glColor3f(0.5, 1.0, 0.5)
glVertex3f( -0.25,  0.0, 0.0)
glVertex3f( -0.5,  -0.5, 0.0)
glVertex3f(  0.0,  -0.5, 0.0)

# Blue
glColor3f(0.5, 0.5, 1.0)
glVertex3f( 0.25,  0.0, 0.0)
glVertex3f( 0.0,  -0.5, 0.0)
glVertex3f( 0.5,  -0.5, 0.0)

glEnd()
```



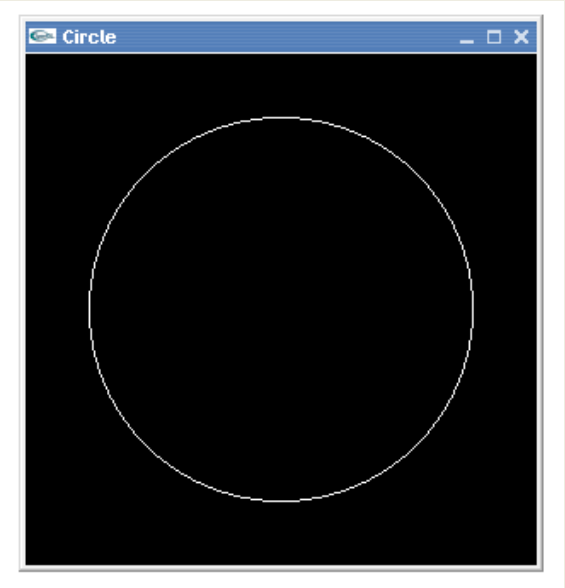
ตัวอย่าง

```
glBegin(GL_QUADS)

glColor3f(  1.0,  1.0, 0.0)
glVertex3f(-0.5, -0.5, 0.0)

glColor3f(  1.0,  0.0, 1.0)
glVertex3f( 0.5, -0.5, 0.0)

glColor3f(  1.0,  1.0, 1.0)
glVertex3f( 0.5,  0.5, 0.0)

glColor3f(  0.0,  1.0, 1.0)
glVertex3f(-0.5,  0.5, 0.0)

glEnd()
```



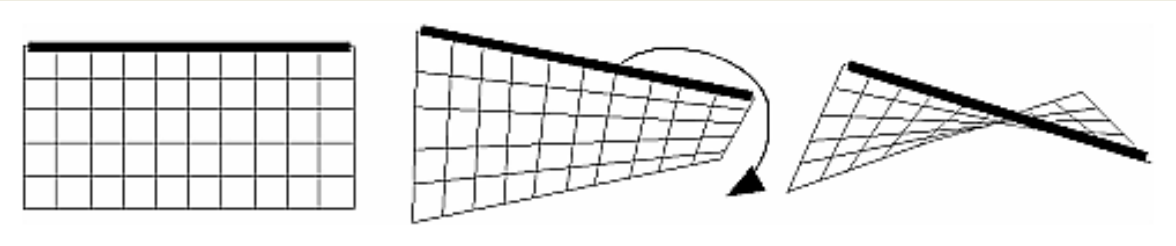
ตัวอย่าง

```
from math import pi, sin, cos

glBegin(GL_LINE_LOOP)

for i in range(256):
   theta = 2*i*pi/256
   y = 0.75*sin(theta)
   x = 0.75*cos(theta)
   glVertex2f(x,y)

glEnd()
```



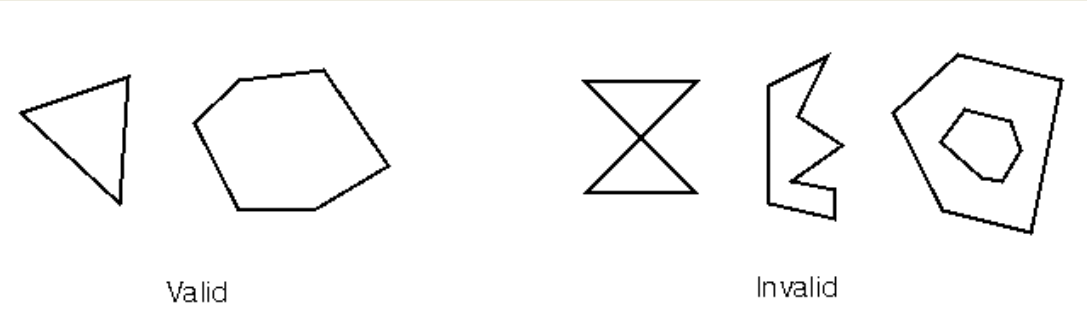
เกี่ยวกับรูปหลายเหลี่ยม

- OpenGL รับประกันว่าจะวาดรูปหลายเหลี่ยมที่จุดทั้งหมดอยู่ในระนาบเดียวกันได้ถูกต้อง
  - ถ้าไม่เป็นเช่นนั้นจะไม่รับประกันว่าจะถูกต้องหรือไม่
- **ข้อสังเกต:** จุดทุกจุดที่อยู่บนรูปสามเหลี่ยมอยู่บนระนาบเดียวกัน
  - แต่นี่ไม่เป็นจริงสำหรับสี่เหลี่ยมหรือรูปหลายเหลี่ยมอื่น



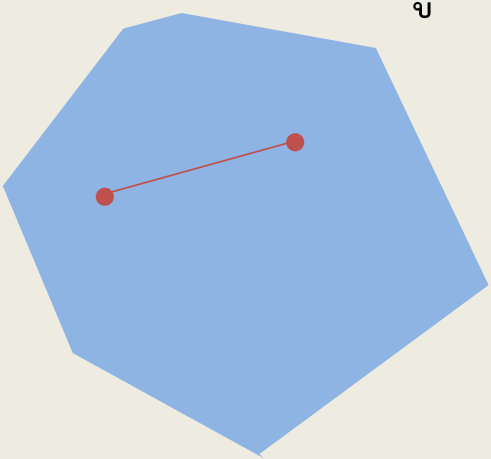
เกี่ยวกับรูปหลายเหลี่ยม (ต่อ)

- รูปหลายเหลี่ยมที่วาดได้ด้วย glBegin(GL_POLYGON) จะมีคุณสมบัติดังนี้
  - เส้นขอบของมันจะต้องไม่ตัดกัน
  - รูปหลายเหลี่ยมนั้นจะต้องเป็นรูปหลายเหลี่ยมนูน (convex polygon)
  - รูปหลายเหลี่ยมนั้นจะต้องไม่มี “รู”

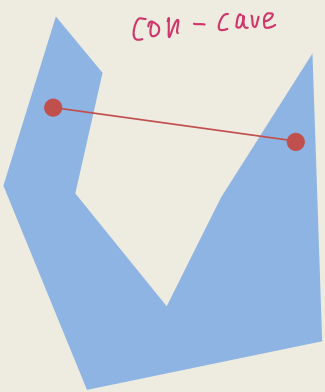


รูปหลายเหลี่ยมนูน (convex polygon)

- สำหรับจุดสองจุดใดๆ ที่อยู่ในรูปหลายเหลี่ยม เมื่อลากส่วนของเส้นตรงเชื่อมจุดสองจุดนั้น ส่วนของเส้นตรงนั้นต้องอยู่ในรูปหลายเหลี่ยมนั้นด้วย
- **ข้อสังเกต:** สามเหลี่ยมเป็นรูปหลายเหลี่ยมนูนเสมอ



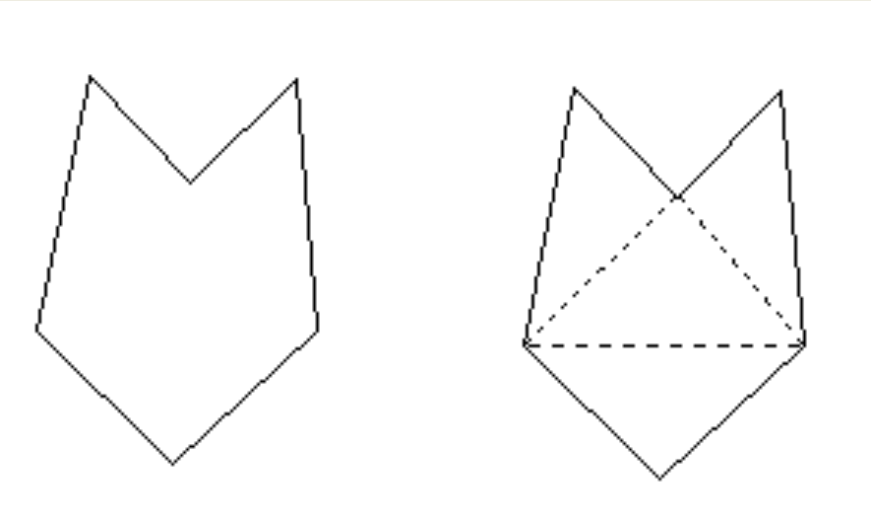
รูปหลายเหลี่ยมนูน



ไม่ใช่รูปหลายเหลี่ยมนูน

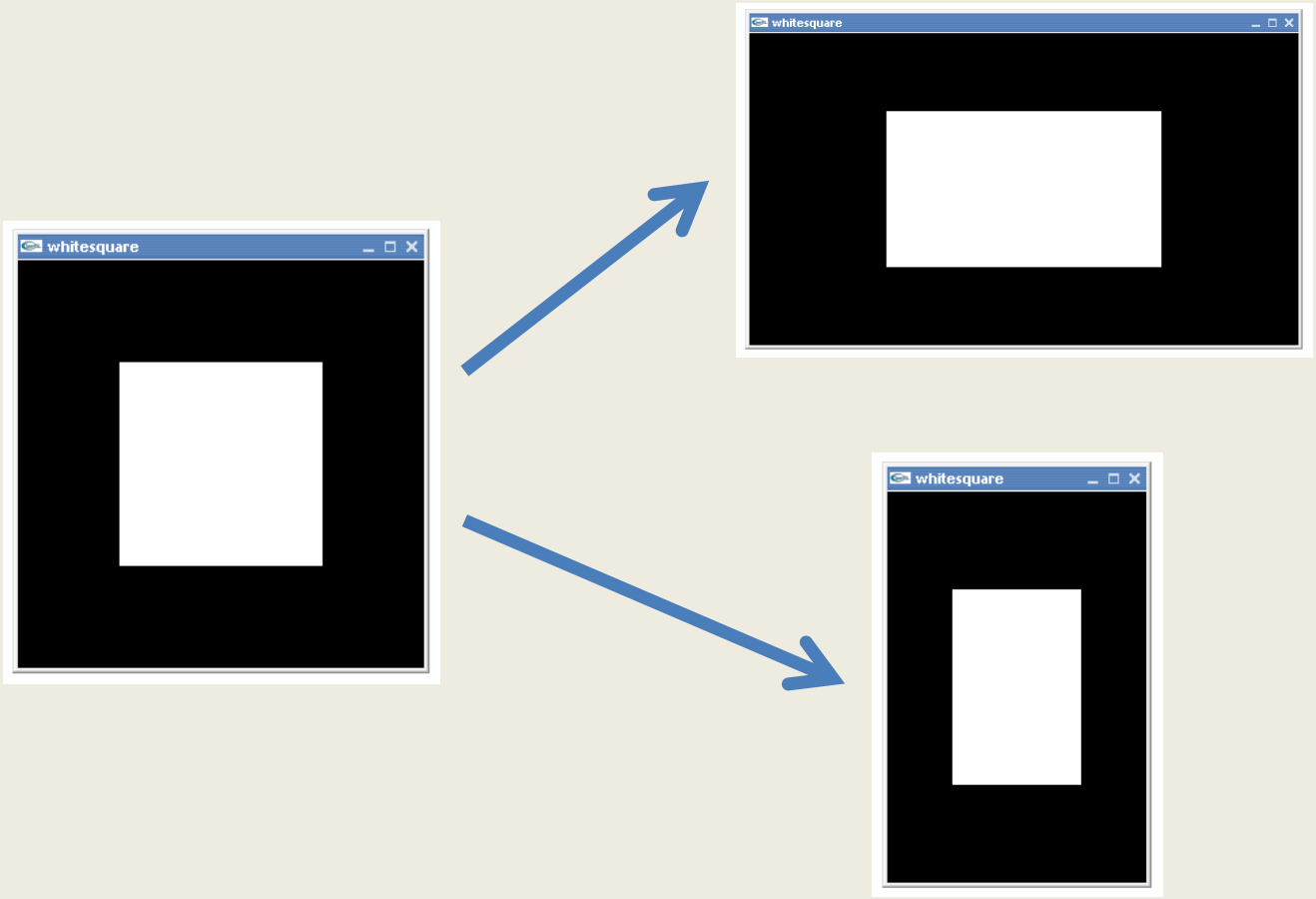
รูปหลายเหลี่ยมใดๆ

- แล้วเราจะวาดรูปหลายเหลี่ยมที่ไม่ใช่รูปหลายเหลี่ยมนูน หรือรูปหลายเหลี่ยมที่มีรูอย่างไร?
- แยกรูปหลายเหลี่ยมเหล่านั้นออกเป็นรูปหลายเหลี่ยมนูนหลายๆ รูป



การย่อขยายขนาดวินโดว์

เมื่อย่อขยายวินโดว์



glutReshapeFunc

- glutReshapeFunc(reshape_callback) → เกิด event Func เรียกมาใช้งาน
  - ฟังก์ชันที่จะส่งให้ glutReshapeFunc ต้องมีการนิยาม
    def <ชื่อฟังก์ชัน>(width, height)
  - เป็นฟังก์ชันที่รับ int สองตัว
  - ฟังก์ชันนี้จะถูกเรียกทุกครั้งที่วินโดว์เปลี่ยนขนาด
  - Argument ที่เป็น int สองตัว
    - ตัวแรกคือความกว้างของหน้าต่างหลังถูกเปลี่ยนความกว้าง หน่วยเป็นพิกเซล
    - ตัวที่สองคือความสูงของหน้าต่างหลังถูกเปลี่ยนความกว้าง หน้วยเป็นพิกเซล
  - เราสามารถใช้ฟังก์ชันที่ให้ glutReshapeFunc ไปเป็นตัวปรับอัตราส่วนของรูปที่แสดงออกมาได้

ตัวอย่างการใช้ glutReshapeFunc

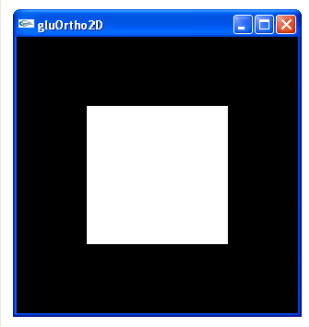
```
def reshape(w, h):
    .........
    .........


def draw():
    .........
    .........
```

```
glutInit(sys.argv)
glutInitDisplayMode(GLUT_RGBA |
    GLUT_SINGLE)
glutCreateWindow("window")
glutReshapeFunc(reshape)
glutDisplayFunc(draw)
glutMainLoop()
```

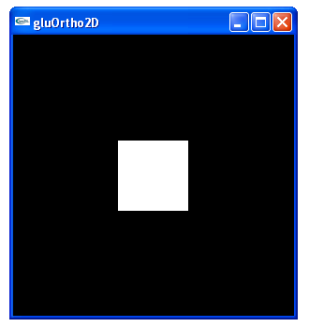
gluOrtho2D

- gluOrtho2D(left, right, bottom, top)
  - ชนิดของพารามิเตอร์มีชนิดเป็น GLdouble
  - ใช้เปลี่ยน projection matrix ให้เป็นการฉายแบบ orthogonal projection
    - เราจะไปพูดถึงคำศัพท์เหล่านี้ในอีกประมาณสองอาทิตย์หน้า
  - ตอนนี้เข้าใจว่าเป็นการเซตพิกัดของจุดมุมของบริเวณที่เราจะวาดรูป
    - มุมล่างซ้ายเป็น (left, bottom)
    - มุมล่างขวาเป็น (right, bottom)
    - มุมบนซ้ายเป็น (left, top)
    - มุมบนขวาเป็น (right, top)

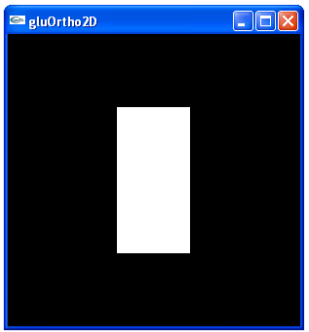
gluOrtho2D (ต่อ)



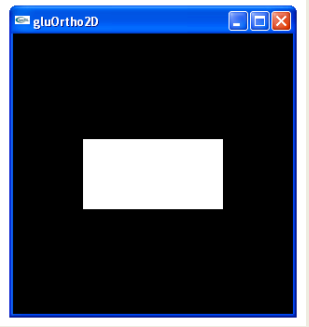
gluOrtho2D(-1,1,-1,1)



gluOrtho2D(-2,2,-2,2)



gluOrtho2D(-2,2,-1,1)



gluOrtho2D(-1,1,-2,2)

glViewport

- glViewport(*x*, *y*, *width*, *height*)
  - กำหนดพื้นที่ในวินโดว์ที่จะใช้แสดงผลภาพที่ OpenGL สร้าง
  - x, y, width, height มีหน่วยเป็นพิกเซล
  - พิกัด (x,y) กำหนดตำแหน่งมุมบนซ้ายของพื้นที่
  - width กำหนดความกว้างของพื้นที่
  - height กำหนดความสูงของพื้นที่
- ในตัวอย่างเราใช้ glViewport(0, 0, w, h) หมายความว่าเราใช้พื้นที่ทั้งหมดของวินโดว์

Callback สำหรับเวลาวินโดว์เปลี่ยนขนาด (ต่อ)

- เราเรียก

  glMatrixMode(GL_PROJECTION)

  glLoadIdentity()

  ก่อนจะเรียก

  gluOrtho2D(...)

  เพื่อกำหนดระบบพิกัด
- ทั้งสองฟังก์ชันนี้เกี่ยวข้องกับการกำหนด projection transform ซึ่งเราจะพูดถึงในสองสัปดาห์หน้า
- ตอนนี้ให้จำไปก่อนว่าต้องเรียนสองฟังก์ชันนี้ก่อนใช้ gluOrtho2D เสมอ
- ก่อนเรียกใช้ gluOrtho2D เราจำเป็นต้อง import OpenGL.GLU ด้วย