Computer Graphics

Prof. Jibum Kim

Department of Computer Science & Engineering

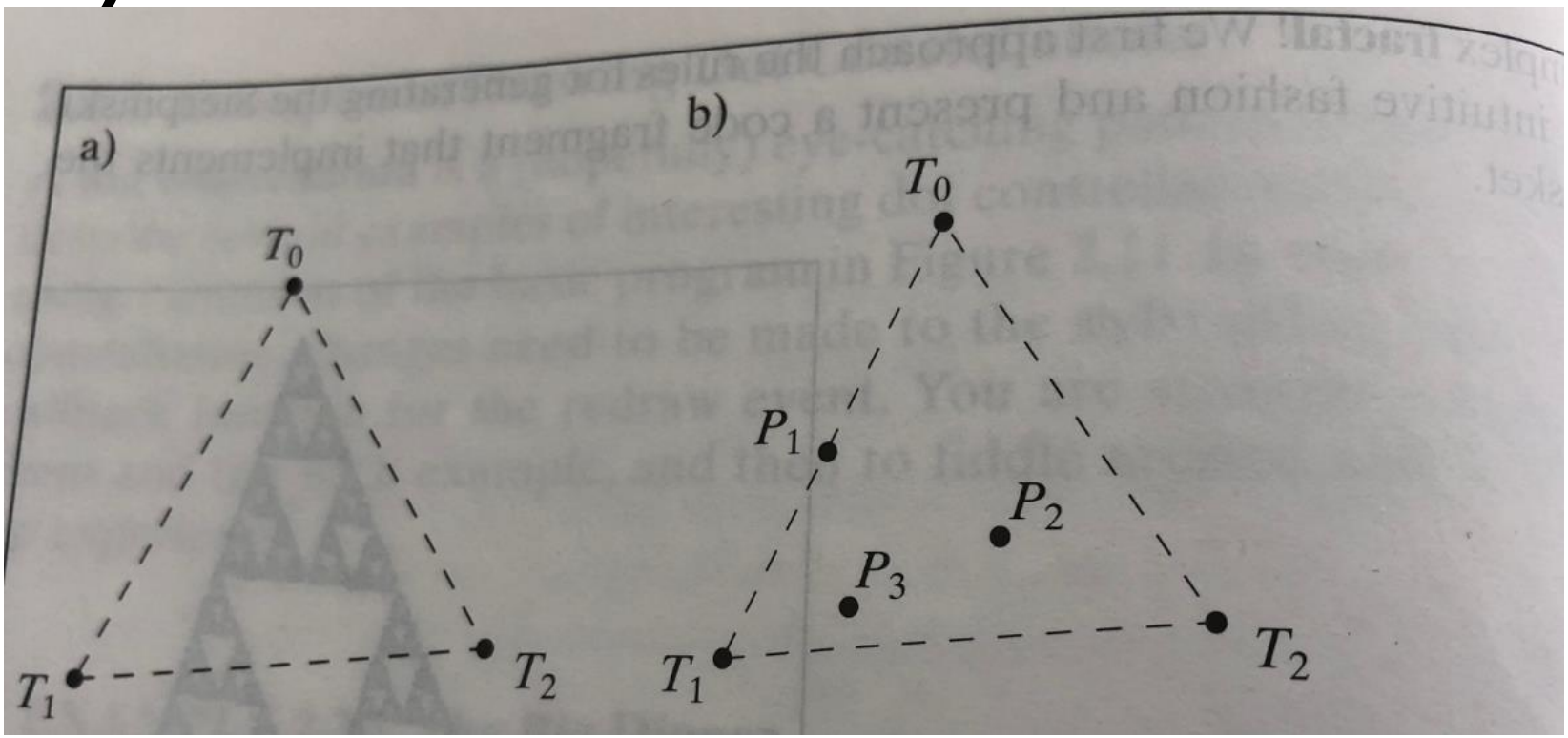
Incheon National University

- **Sierpinski Gasket (Sierpinski triangle)**

- 첫번째 **OpenGL**예제를 보면
- **GL_POINTS** 를 이용하여 점을 그렸다
- 이번에는 **OpenGL**의 **GL_POINTS**와 **C**언어에서의 **rand()** 함수를 이용하여
- **Sierpinski Gasket**을 그려보자
- **rand() is a function that returns a pseudo random between 0 and some upper limit.**
- **rand()%3 => 0 ,1 ,2** 가 난수로 발생. **Why?** 삼각형의 세 점을 임의로 선택하기 위하여

- **Sierpinski Gasket 알고리즘**
- **1. T0, T1, T2**의 세 고정된 점을 이용하여 **Parent (**부모**)** 삼각형을 만든다**.**
- **2. T0, T1, T2**중 한 점을 무작위로 선택해 최초 시작점 **p0**로 놓는다
- 이 후에 다음 **3**번에서 **5**번 사이 과정을 반복한다
- **3. T0, T1, T2**중 한 점을 무작위로 선택해 **T**라고 놓는다
- **4.** 다음점 **pk** 를 **T**와 **p(k-1)**의 중간점으로 정한다
- **5. pk**를 찍는다

- 예)

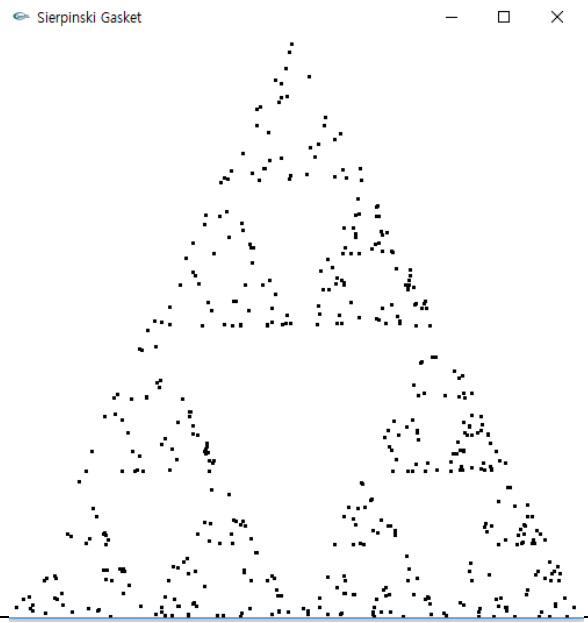


- **https://www.youtube.com/watch?v=GCkJ2EufQuY**

- **OpenGL**로 작성한 **Sierpinski gasket** 코드 예
- 가시 공간 : **gluOrtho2D(0.0, 500.0, 0.0, 500.0);**

- **1.** 최초 **3**개의 **vertex** 위치**. (0, 0) (250, 500) (500, 0)**
- **2.** 삼각형 안의 내부 점 하나 정함 **p=(75, 50)**
- 아래 **3**과 **4** 반복 **(500**번 반복**)**
- **3. 3**개의 **vertex** 중 하나를 무작위로 선택
- **4. p**와의 중점 구하고 이를 새로운 **p**로 정함

- **https://www.dropbox.com/s/q2kvrerqpt5sqk9/gasket.txt?dl=0**



- **Geometric primitives**

- **Geometric primitives (also, called drawing primitives) of OpenGL are the parts that programmers use in Lego-like manner to create objects**
- **Simplest geometric objects that the system can draw**
- **https://en.wikipedia.org/wiki/Geometric_primitive**

- OpenGL geometric primitives
- 1. Points (GL_POINTS)
- 2. Lines (GL_LINES, GL_LINE_STRIP, GL_LINE _LOOP)
- 3. Polygons (GL_POLYGON)
- 4. Rectangles (GL_QUADS, GL_QUAD_STRIP)
- 5. Triangles (GL_TRIANGLES, GL_TRIANGLE_STRIP)
- …

- **Line drawings in OpenGL**

- **OpenGL에서의 line 그리기**
- 예)
- **glBegin(GL_LINES);**
- **glVertex2i(40, 100);**
- **glVertex2i(202, 96);**
- **glEnd();**

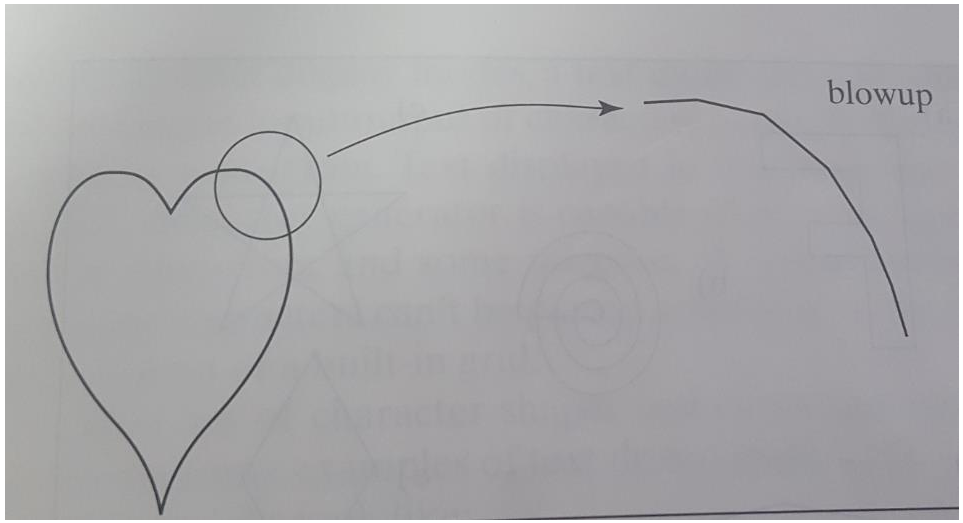
- 만일 **glBegin(GL_LINES)**와 **glEnd()** 사이에 **2**개 이상의 **vertex**가 있다면**?**
- 예)
- **glBegin(GL_LINES);**
- **glVertex2i(10, 20);**
- **glVertex2i(40, 20);**
- **glVertex2i(20, 10);**
- **glVertex2i(20, 40);**
- **glEnd();**

- 예)
- **https://www.dropbox.com/s/j84219x51jp3j13/line_drawing.txt?dl=0**

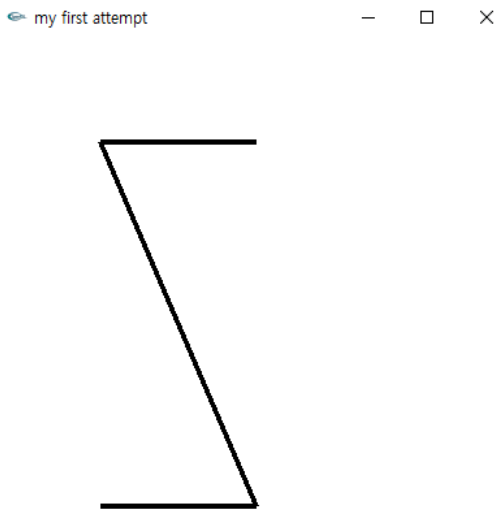
- **Line**의 두께 **(thickness)** 조절 방법
- **glBegin(GL_LINES)** 위에 아래 코드 추가
- **glLineWidth(4.0);**

- **Polyline, Line strip , Line loop**

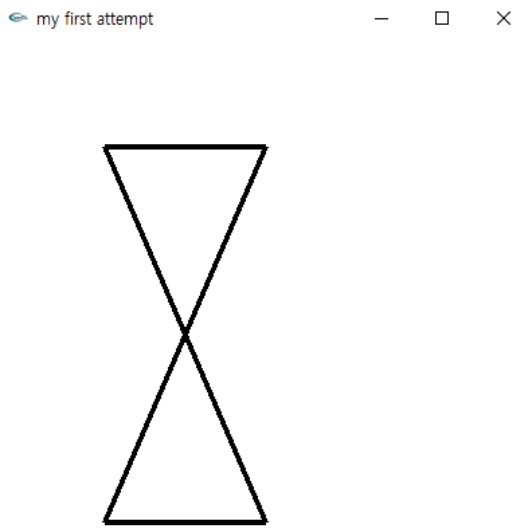
- **polyline** 이란?
- **A polyline is a connected sequence of straight lines**
- **polyline**은 직선이지만 어떤 경우에는 부드러운 **curve** (곡선) 형태를 보일 때도 있다



- **OpenGL**에서는 **polyline**을 **line strip**이라고 불린다
- 먼저 **line strip**을 사용한 다음 예제를 실행해 보자
- **https://www.dropbox.com/s/6arfm9e2vznwvu0/line_strip.txt?dl=0**



- 여기서
- **GL_LINE_STRIP 을 GL_LINE_LOOP**
- 로 바꾸어 보자
- 어떤 변화가 있는가?



- 외부의 저장된 파일과 **OpenGL**의 **line strip**을 이용하여 보다 실제적인 **polyline** 그리기

- **poly line을 이용한 dinosaurs**

외부 데이터 파일 **dino.dat** 를 열어보면 다음과 같이 구성되어 있다

- https://www.dropbox.com/s/q69t0yj7vy27jux/dino.dat?dl=0
- 총 **21**개의 **polyline**
- 각 **polyline**의 **(x, y) coordinate**
- **glvertex2i** 로 **vertex** 위치 정함
- 정해진 **vertex**들을 이용해
- **line strip**으로 **polyline** 그림

```
21
29
32 435
10 439
4 438
2 433
4 428
6 425
10 420
15 416
21 413
30 408
42 406
47 403
56 398
63 391
71 383
79 369
84 356
87 337
89 316
88 302
86 294
83 278
79 256
78 235
79 220
85 204
94 190
```

- 실행 방법: 현재 실행 폴더에 **dino.dat** 파일을 옮기고 아래 코드를 실행해 보자
- **https://www.dropbox.com/s/26wz6qx5q6b9x6m/dinosaur.txt?dl=0**

- **Drawing Rectangles**

- **Rectangle** 그리기
- **Rectangle**을 그리기 위해서는 두 점이 필요하다
- 이 두 점 **(x1,y1), (x2,y2)** 은 **rectangle**의 양 코너점을 의미한다
- **glRecti(GLint x1, GLint y1, GLint x2, GLint y2);**

```
#include <GL/glut.h>
void myInit(void)
 {
    glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
    glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
    glLoadIdentity();// to be explained
    gluOrtho2D(0.0, 200.0, 0.0, 200.0);// to be explained
 }
      void myDisplay(void)
 {
            glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
    glColor3f(0.6, 0.6, 0.6);
            glRecti(20, 20, 100, 70);
            glColor3f(0.2, 0.2, 0.2);
            glRecti(70, 50, 150, 130);
            glFlush();                                    // send all output to display
 }
 void main(int argc, char **argv)
 {
            glutInitWindowSize(400,400);     // set the window size
            glutInitWindowPosition(100, 150); // set the window position on the screen
            glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
            glutDisplayFunc(myDisplay);     // register the redraw function
            myInit();
            glutMainLoop();                          // go into a perpetual loop
 }
```

- 예:
- 가시 공간: **gluOrtho2D(0.0, 200.0, 0.0, 200.0)**
- **glColor3f(0.6, 0.6, 0.6);**
- **glRecti(20, 20, 100, 70);**
- **glColor3f(0.2, 0.2, 0.2);**
- **glRecti(70, 50, 150, 130);**

- **Filling polygons**

- **Polygon:** 다각형
- 하지만 **OpenGL**에서는 **convex polygon**만 지원
- **Convex (볼록) 란? Non-convex?**
- **glBegin(GL_POLYGON);**
- **glVertex2i(x0, y0);**
- **glVertex2i(x1, y1);**
- **…**
- **glVertex2i(xn, yn):**
- **glEnd();**

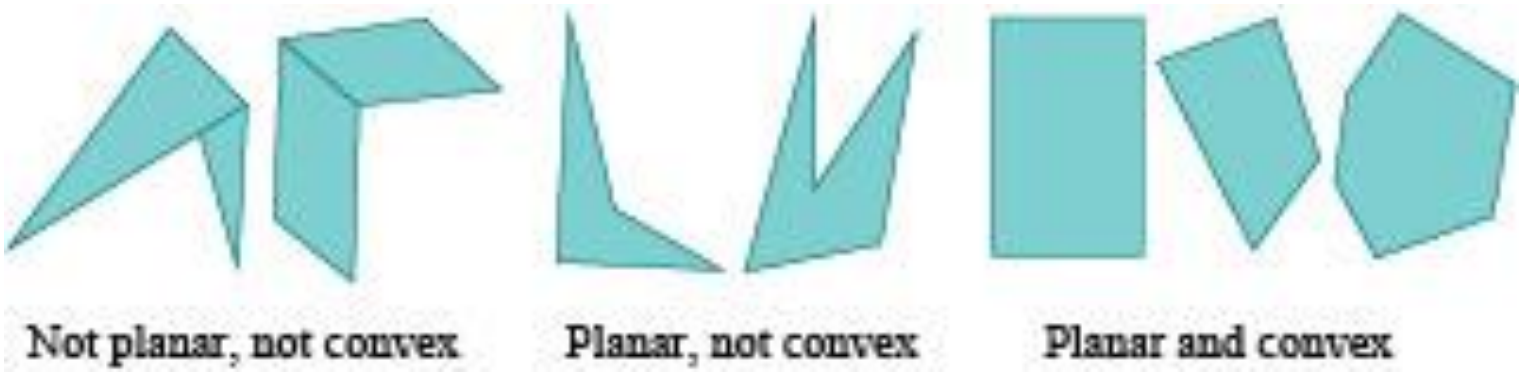
```
#include <GL/glut.h>
void myInit(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
glLoadIdentity();// to be explained
gluOrtho2D(0.0, 200.0, 0.0, 200.0);// to be explained
}
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);      // clear the screen
glColor3f(0.6, 0.6, 0.6);
glBegin(GL_POLYGON);
glVertex2i(10, 10);
glVertex2i(10, 60);
glVertex2i(50, 60);
glVertex2i(70, 30);
glVertex2i(80, 10);
glEnd();
glFlush();                // send all output to display
}
void main(int argc, char **argv)
{
glutInitWindowSize(400, 400);     // set the window size
glutInitWindowPosition(100, 150); // set the window position on the screen
glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
glutDisplayFunc(myDisplay);      // register the redraw function
myInit();
glutMainLoop();      // go into a perpetual loop
}
```

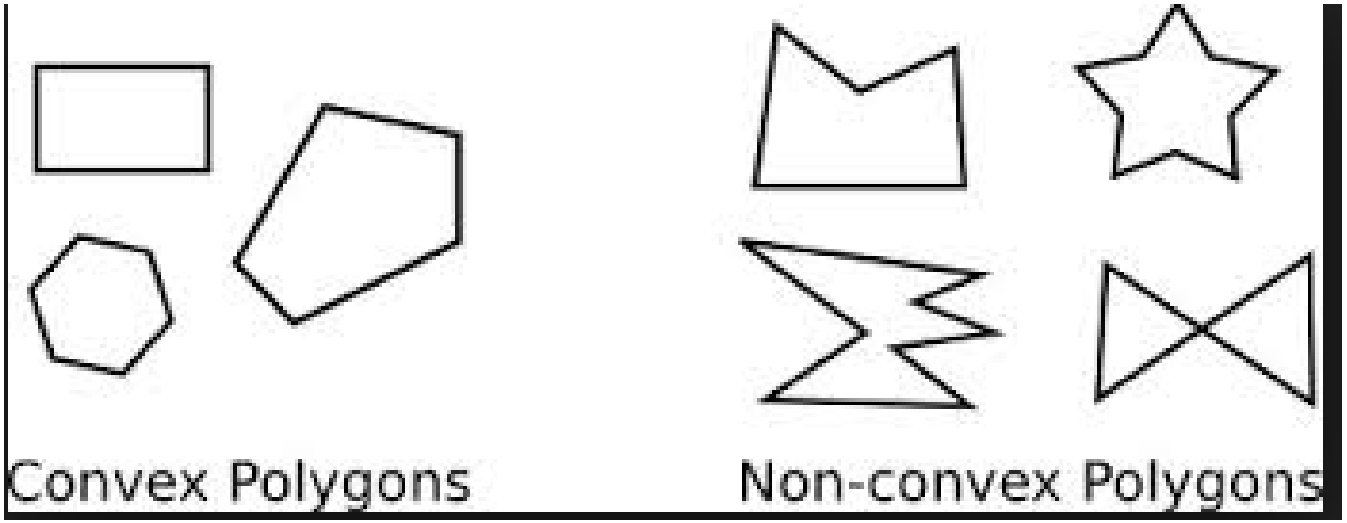
- 가시 공간: **gluOrtho2D(0.0, 200.0, 0.0, 200.0);**
- **5**개의 **vertex**를 이용한 **convex polygon** 예
- **glBegin(GL_POLYGON);**
- **glVertex2i(10, 10);**
- **glVertex2i(10, 60);**

  glVertex2i(50, 60);
- **glVertex2i(70, 30);**
- **glVertex2i(80, 10);**

  glEnd();

- **OpenGL에서 지원하는 polygon은 반드시 convex polygon이고 planar (lies on one plane)이어야 한다**
- **정의: convex polygon: A polygon is convex if it contains every line segment delimited by any two points on its boundary**



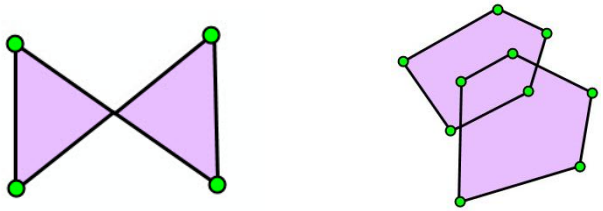


- 어떤 결과를 예상 하는가?
- **Convex polygon**인가? 아닌가?
- 가시 공간: **gluOrtho2D(-1.0, 1.0, -1.0, 1.0);**
- **glBegin(GL_POLYGON);**
- **glVertex2f(0.8, 0.2);**
- **glVertex2f(0.4, 0.4);**
- **glVertex2f(0.2, 0.8);**
- **glVertex2f(0.2, 0.2);**
- **glEnd();**

```
#include <GL/glut.h>
void myInit(void)
 {
   glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
   glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
   glLoadIdentity();// to be explained
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0);// to be explained
}
     void myDisplay(void)
{
            glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
   glColor3f(0.6, 0.6, 0.6);
   glBegin(GL_POLYGON);
   glVertex2f(0.8, 0.2);
   glVertex2f(0.4, 0.4);
   glVertex2f(0.2, 0.8);
   glVertex2f(0.2, 0.2);
   glEnd();
            glFlush();                                  // send all output to display
}
void main(int argc, char **argv)
{
            glutInitWindowSize(400,400);     // set the window size
            glutInitWindowPosition(100, 150); // set the window position on the screen
            glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
            glutDisplayFunc(myDisplay);     // register the redraw function
            myInit();
            glutMainLoop();                          // go into a perpetual loop
```

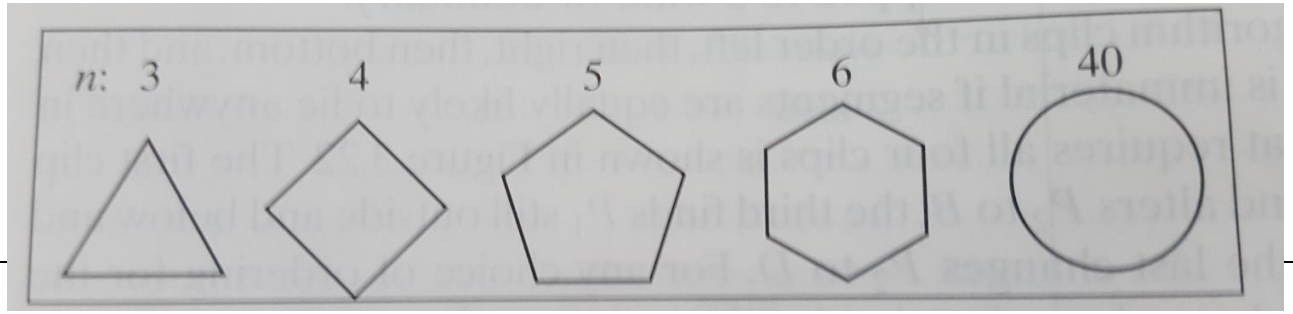
- **Regular polygon**

- **Simple polygon (단순 다각형): a polygon is simple if no two of its edges cross each other**



non-simple polygon 예

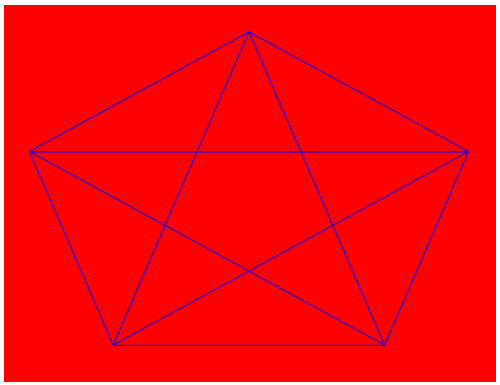
- **Regular polygon: a polygon is regular if it is simple and equiangular (모든 각이 같음) and equilateral (모든 변의 길이가 같음)**
- **n-gon:** 변의 길이가 **n**개인 **regular polygon**

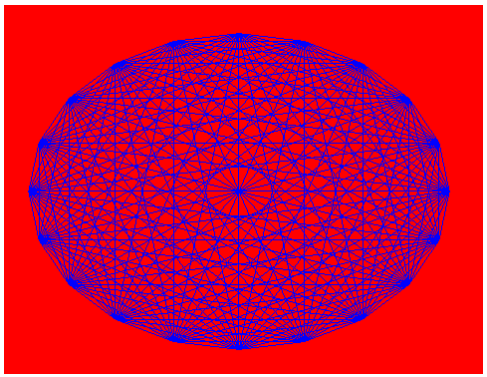


Observation: n-gon에서 **n**이 커지면 원에 가까워 진다

원을 바로 그리지 않고 **n**을 증가시키면서 **n-gon**의 형태로 근사화 가능하다

- **rosette : n-gon with each vertex joined to every other vertex**
- **왼쪽 N=5, 오른쪽 N=20**

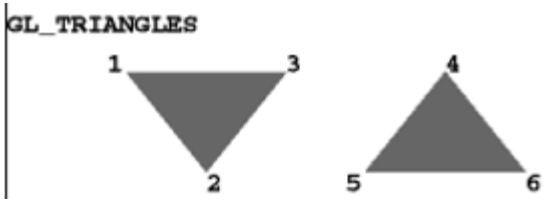




- **https://www.dropbox.com/s/vaxjcren1g562ja/rosette.txt?dl=0**

- **Triangles**

- **GL_TRIANGLES**
- **Draws a sequence of triangles using three vertices at a time**



If there are n vertices

$v_0v_1v_2$, $v_3v_4v_5$, …. , $v_{n-3}v_{n-2}v_{n-1}$

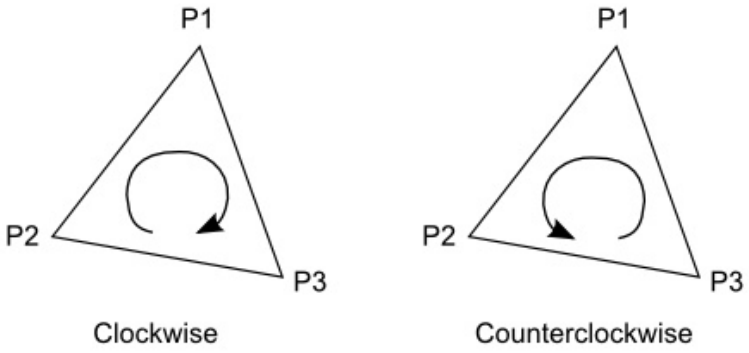
Note: if n is not a multiple of 3, the last one is ignored

By default, triangles are filled

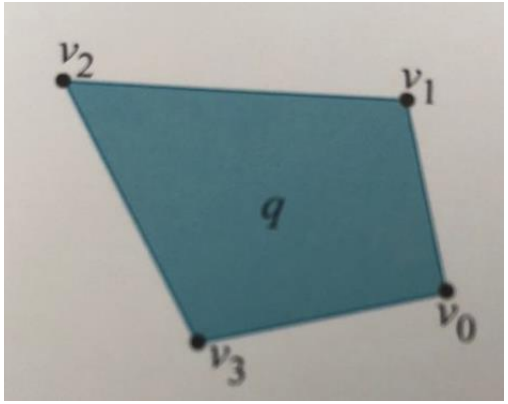
```
#include <GL/glut.h>
void myInit(void)
 {
    glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
    glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
    glLoadIdentity();// to be explained
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);// to be explained
 }
      void myDisplay(void)
 {
            glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
    glColor3f(0.6, 0.6, 0.6);
    glBegin(GL_POLYGON);
    glVertex2f(0.2, 0.2);
    glVertex2f(0.2, 0.8);
    glVertex2f(0.8, 0.2);
    glEnd();
            glFlush();                                  // send all output to display
 }
 void main(int argc, char **argv)
 {
            glutInitWindowSize(400,400);     // set the window size
            glutInitWindowPosition(100, 150); // set the window position on the screen
            glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
            glutDisplayFunc(myDisplay);     // register the redraw function
            myInit();
            glutMainLoop();                          // go into a perpetual loop
 }
```

- **You will be curious whether the given order of vertices will affect the drawing result.** 이를 **orientation** 이라 한다
- **Orientation means whether a cycle goes around**

clock wise (CW, 시계 방향) or counter-clockwise (CCW, 반시계)



- 정의: **Two orders of the vertices of a polygon are said to be equivalent (동일) if one can be cyclically rotated into the order**



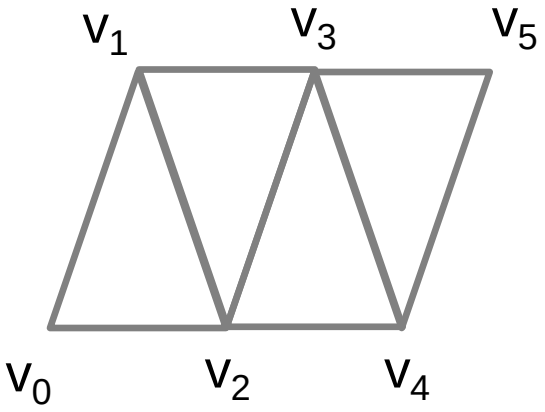
- **v0v1v2v3 , v1v2v3v0 , v2v3v0v1 , v3v0v1v2 (동일)**
- **v0v3v2v1 , v3v2v1v0 , v2v1v0v3 , v1v0v3v2 (동일)**

- **Orientation**은 왜 중요할까?
- **Viewer (camera)** 입장에서 특정 **polygon (triangle)**이 **viewer**에게 앞면 **(front-face)**인지 후면 **(back-face)**인지 판단하는 기준이 된다
- **OpenGL**에서는 **triangle**의 **orientation**이 **viewer**가 보기에 **CCW (**반시계 방향**)**이면 앞면**, CW (**시계 방향**)**이면 후면으로 판단한다
- 보다 자세한 내용은 **back-face culling** 부분에서 배울 예정

- **Triangle**과 같은 **polygon**에 색깔을 줄 때에 각 **vertex**별로 **color**를 줄 수도 있다
- **glBegin(GL_POLYGON);**
- **glColor3f(0.0, 1.0, 0.0);**
- **glVertex2f(0.2, 0.2);**
- **glColor3f(0.0, 0.0, 1.0);**
- **glVertex2f(0.2, 0.8);**
- **glColor3f(1.0, 1.0, 0.0);**
- **glVertex2f(0.8, 0.2);**
- **glEnd();**

- 이렇게 **vertex** 별로 **color**를 정하면 **polygon** 내부의 색은 어떻게 정해질까**?**
- 내부 색은 후에 배울 **interpolation (보간)** 기법에 의해서 색깔이 정해진다

- **GL_TRIANGLE_STRIP**
- **Draws a sequence of triangles called a triangle strip**
- **If there are n vertices ($v_0v_1v_2v_3v_4v_5,\ldots.v_{n-1}$)**
- **Draws a series of using vertices v0, v1, v2, then v2, v1, v3 (note the order), then v2, v3, v4, and so on. The ordering is to ensure that the triangles are all drawn with the same orientation**
- **E.g., Say we have 6 vertices**



- **6개의 vertex를 정하고 triangle strip과 interpolation을 이용한 예제**

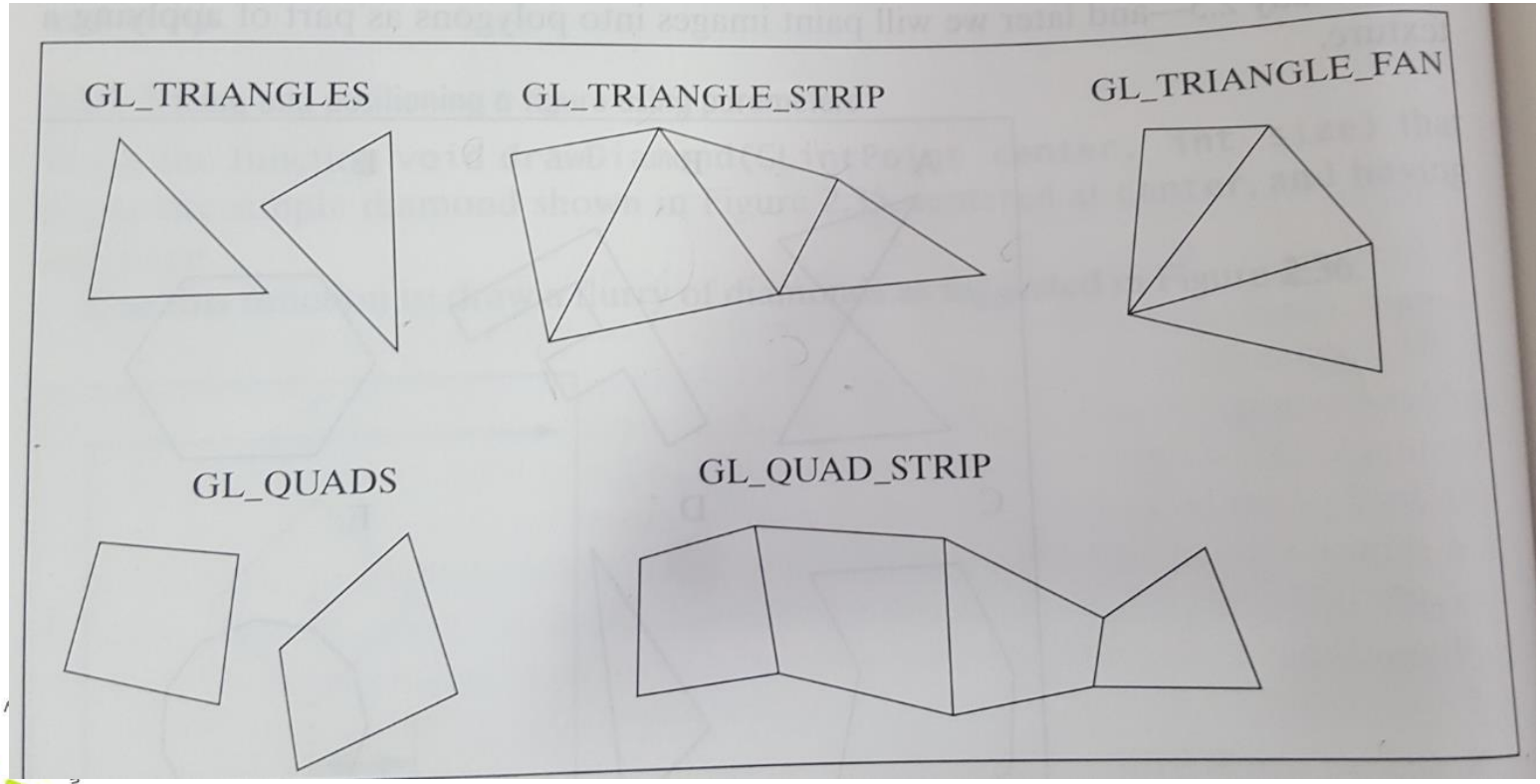
- **https://www.dropbox.com/s/mrw1wouank3p48k/triangle_strip.txt?dl=0**

- **Triangle strip**을 만들 때 주의해야할 점은 없을까?
- **1. orientation (**각 삼각형의 **orientation)**
- **2. triangulation (**삼각형 모양**)**

- **GL_TRIANGLE_FAN: draws a series of connected triangles based on triplets of vertices**
- **v0v1v2, v0,v2,v3, v0,v3,v4,…**

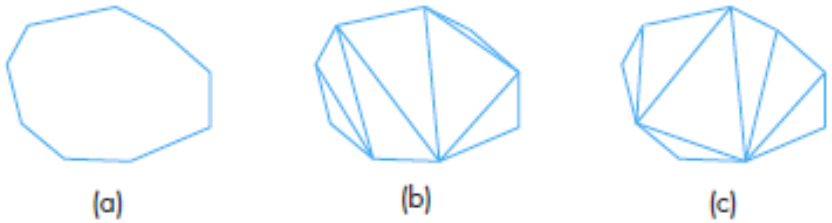
```
#include <GL/glut.h>
void myInit(void)
 {
   glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
   glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
   glLoadIdentity();// to be explained
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0);// to be explained
 }
     void myDisplay(void)
 {
            glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
   glBegin(GL_TRIANGLE_FAN);
            glColor3f(0.0, 0.0, 0.0);
   glVertex2f(0.0, 0.0);
   glVertex2f(0.5, 0.0);
            glVertex2f(0.3, 0.2);
            glVertex2f(0.0, 0.5);
            glVertex2f(-0.2, 0.3);
   glEnd();
            glFlush();                                  // send all output to display
 }
 void main(int argc, char **argv)
 {
            glutInitWindowSize(400,400);     // set the window size
            glutInitWindowPosition(100, 150); // set the window position on the screen
            glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
            glutDisplayFunc(myDisplay);     // register the redraw function
            myInit();
            glutMainLoop();                          // go into a perpetual loop
```

- 여러 가지 **geometric primitives**



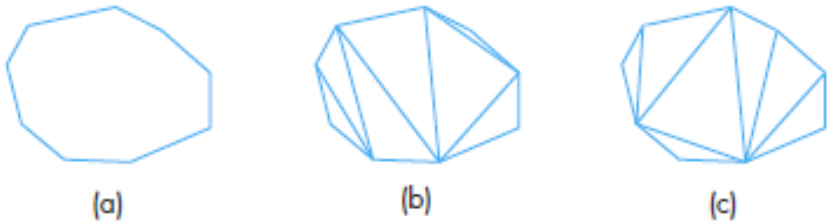
- **Triangulation (삼각 분할)**

- 많은 경우에 **polygon**을 직접 그리는 대신에 여러 개의 **triangle**을 모아서 **polygon**을 표현하는 경우가 많다
- 이를 **triangulation** 이라 한다
- **Why? Triangles are easy to draw**
- **Polygon**에 대한 **triangulation**은 **unique**하지 않다



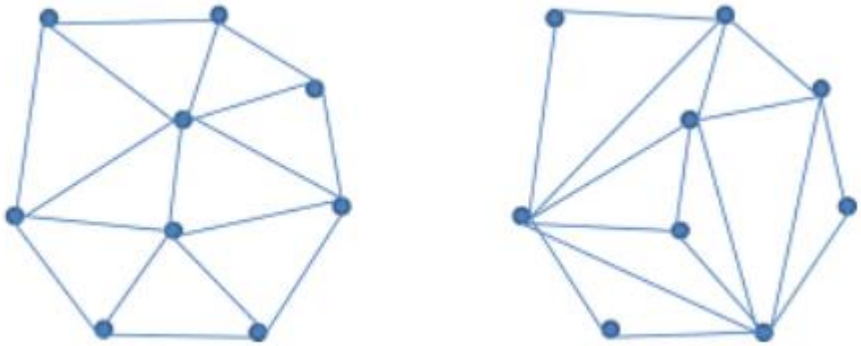
(a) Polygon (b) A triangulation (c) another triangulation

- **Long thin triangles are bad!**
- 정삼각형에 가까운 삼각형들보다 **long thin** 삼각형들은 일반적으로 **rendering** 시에 더 안좋은 영향을 미친다고 알려져 있다
- 그렇다면 앞의 **(b)**와 **(c)** 중에 더 바람직한 **triangulation**은**?**



- 가장 유명한 **triangulation** 방법중의 하나는 **Delaunay triangulation**이다

- **Delaunay triangulation algorithm finds a best triangulation in the sense that when we consider the circle determined by any triangle, no other vertex lies in this circle**



(Left) Delaunay triangulation (right) Non-Delaunay triangulation

- **We can avoid long thin triangles!**

https://en.wikipedia.org/wiki/Delaunay_triangulation

- https://en.wikipedia.org/wiki/Delaunay_triangulation
- 후에 간단히 소개하겠지만 **Triangulation**은 **tessellation**의 특별한 **case**이다
- **Polygon**을 삼각형이 아닌 **polygonal mesh**로 나누는 방법이다 **(**메쉬 자료구조는 후에 배울 예정**)**

- 이 수업에서 다루지는 않지만 **Delaunay triangulation**을 만드는 알고리즘은 계산 기하학등에서 자주 언급되는 알고리즘 중의 하나이다
- 참고: **http://www.secmem.org/blog/2019/01/11/Deluanay_Triangulation/**