3 . OpenGL 다루기

3.1 Setting Up a 2D Drawing Canvas : Coordinate System

2차원 평면에 그림을 그리기 위해서는 좌표계(Coordinate System)가 필요로 하다. 실제로 사용자의 마우스 입력에 의해 그려지는 곳은 모니터의 윈도우이기 때문에 윈도우 좌표계를 알아야 하며, 가상공간을 다루는 것은 OpenGL이기 때문에 OpenGL에서 사용되는 좌표계를 알아야 한다.

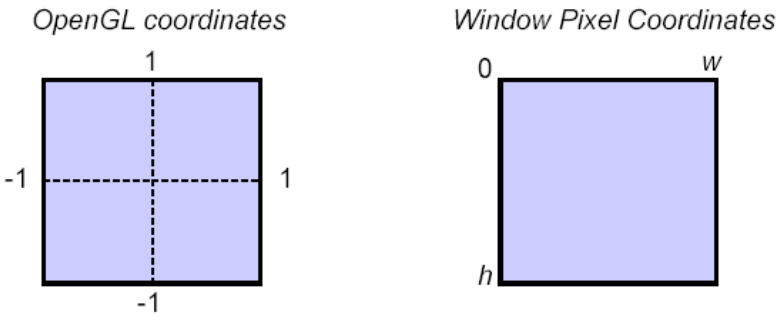


Fig 3.1 OpenGL 좌표계와 Window 좌표계

OpenGL 좌표계는 우리가 알고 있는 좌표계와 동일하다. 즉, 우축으로 갈수록 x 값이 증가하고, 위로 갈수록 y 값이 증가한다. 반대로 window 좌표계에서는 실제로 우축으로 갈수록 x 값이 증가하지만 아래로 향할수록 y 값이 증가하는 시스템을 따른다. OpenGL 좌표계 영역 [-1,1]x[-1,1]과 window 좌표계 영역 [0,w]x[0,h]이 대응된다고 가정하자. 윈도우에서 특정한 위치에 점을 찍고자 하여 마우스의 커서를 그 위치에 놓고 클릭을 하여 위치를 (x, y)로 받아들인다면, OpenGL 좌표계에서는 아래의 (s,t)로 바뀌게 된다.

$$2:w = s+1:x, \qquad 2:h = t+1:h-y$$
$$s = \frac{2}{w}x - 1, \qquad t = \frac{2}{h}(h-y) - 1$$

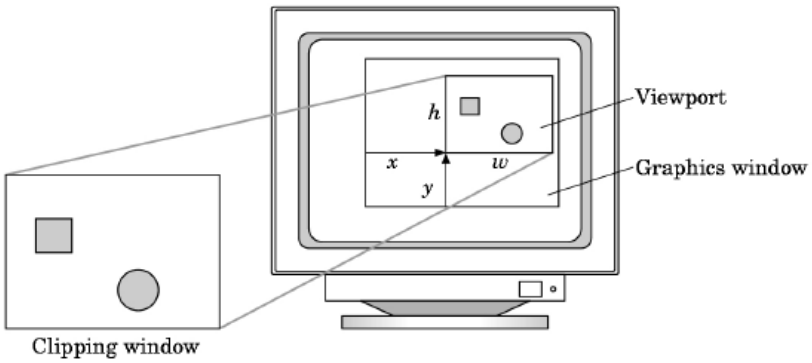
윈도우시스템에 사용되는 윈도우좌표계는 윈도우 위치와 크기를 조절하는 GLUT 함수에서 사용된다. 일반적으로 main() 함수에 다음과 같은 함수가 있다고 한다면,

```
glutInitWindowPosition(50, 100);
glutInitWindowSize(500, 800):
```

모니터의 좌측상단 모서리에서 우측으로 50픽셀, 아래로 100픽셀 되는 지점에 윈도우의 좌측상단 점이 놓이게 되고 가로로 500픽셀, 세로로 800픽셀 크기의 윈도우가 만들어지게 됨을 의미한다. 즉, 그림을 나타낼 수 있는 판이 만들어진 셈이다.

컴퓨터그래픽스는 가상의 공간에 물체를 만들어서 카메라로 찍어 컴퓨터 모니터에 보이게 하는 것이라고 했었다. 가상공간에 사용되는 좌표계는 실세계와 같은 오른손좌표계를 따르며, 이에 따라 공간에 놓여 있는 물체들의 좌표는 OpenGL 좌표계로 표현할 수 있다. 이렇게 가상공간에 놓여 있는 물체들을 카메라로 찍는 과정을 거치게 되는데 가상공간에 있는 모든 물체를 볼 수 있는 것이 아니라 카메라 렌즈로

통해서 보이는 영역만 볼 수 있다. 그 영역은 사각형 형태이며 이를 clipping window 라 한다. 카메라로 찍은 장면은 필름에 담겼다가 원하는 크기의 종이에 현상되는 과정을 거치듯이 클리핑사각형 안에 있는 물체를 윈도우의 적당한 영역의 사각형에 그리는 과정을 하여야 한다. 경우에 따라서는 윈도우에 가득하게 그릴 수도 있고 보다 작은 사각형에 그릴 수도 있을 것이다. 윈도우에 그려지는 사각형을 Viewport 라고 한다. 그림 3.1은 clipping rectangle, viewport, window들 간의 상관관계를 설명하고 있다.



이러한 사각형들의 관계를 이해하기 위하여 다음과 같은 예제를 생각해 보자. 가상공간에 원점을 중심을 하고 반지름이 1인 원이 있고, 점(5,5)를 중심으로 하고 한 변의 길이가 2인 정사각형이 놓여 있다고 하자. 이러한 장면을 Clipping window, Viewport, Graphics window의 크기와 위치를 달리하는 여러 예제를 살펴보도록 하자. Clipping window를 설정하는 OpenGL 함수는 카메라 설정에서 상세히 설명을 하겠지만 우선 수직투영으로 장면을 촬영한다고 가정한다면, gluOrtho2D(xmin, xmax, ymin,ymax) 를 사용하며, Viewport를 설정하는 함수는 glViewport(x, y, w, h)이며, 그림판에 해당하는 Graphics window를 설정하기 위해서는 glutInitWindowPosition(win_x, win_y)와 glutInitWIndowSize(win_w, win_h)를 사용한다. 다음은 이러한 빨강색의 원과 파랑색의 사각형을 그리는 코드를 보이고 있다.

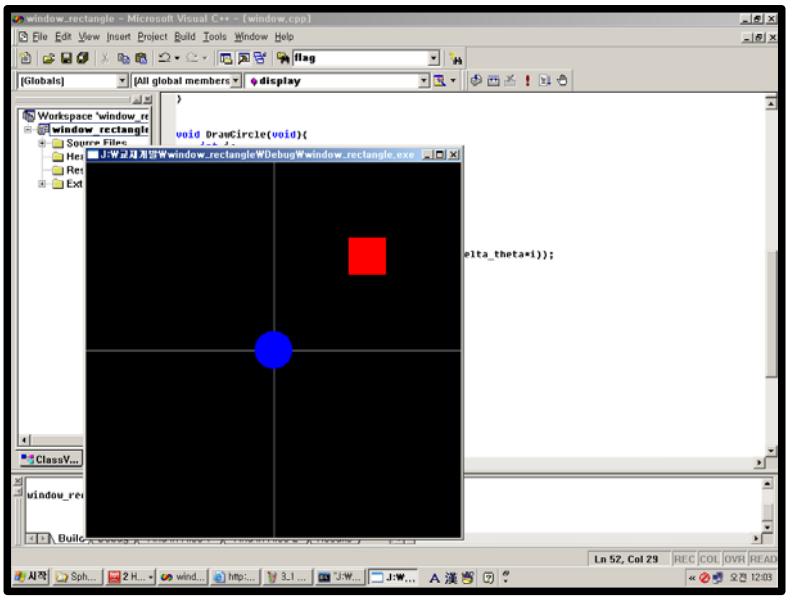
```
#include <gl/glut.h>
#include <iostream.h>
#include <math.h>
#define                 PI              3.1415926
#define                 Vertex_No       36

void init (void){
      glClearColor (0.0, 0.0, 0.0, 0.0) ;
      glShadeModel(GL_FLAT);
}

void DrawAxis(void){
      glColor3f(0.3f, 0.3f, 0.3f);
      glLineWidth(2.0);
      glBegin(GL_LINES);
            glVertex2f(-10.0, 0.0);
```

```
                glVertex2f(10.0,  0.0);
                glVertex2f(0.0,  -10.0);
                glVertex2f(0.0,  10.0);
        glEnd();
}

void DrawSquare(void){
        glColor3f(1.0f,  0.0f,  0.0f);
        glBegin(GL_POLYGON);
                glVertex2f(4.0,  4.0);
                glVertex2f(6.0,  4.0);
                glVertex2f(6.0,  6.0);
                glVertex2f(4.0,  6.0);
        glEnd();
}

void DrawCircle(void){
        int         i;
        float       delta_theta;
        delta_theta = 2*PI/Vertex_No;
        glColor3f(0.0f,  0.0f,  1.0f);
        glBegin(GL_POLYGON);
                for ( i = 0; i< Vertex_No; i++)
                        glVertex2f(cos(delta_theta*i),  sin(delta_theta*i));
        glEnd();
}

void display(void){
        glClearColor(0.0,  0.0,  0.0,1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        DrawAxis();
        DrawSquare();
        DrawCircle();
        glFlush();
}

void reshape(int w,  int h){
        glViewport(0,  0,  (GLsizei)  w,  (GLsizei)  h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-10,  10,  -10,  10);
        glMatrixMode  (GL_MODELVIEW);
}
```

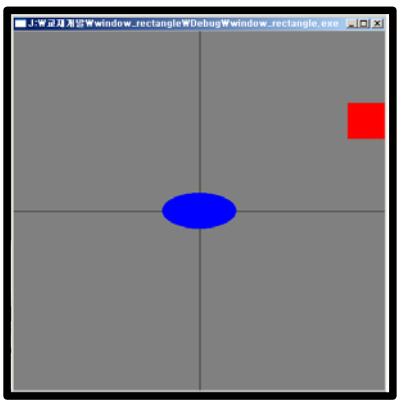
```
void main (int argc, char** argv){
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (100, 200);
    glutInitWindowSize (500, 500);
    glutCreateWindow (argv[0]);
    init();
    glutDisplayFunc (display);
    glutReshapeFunc (reshape);
    glutMainLoop();
}
```

예제 1)

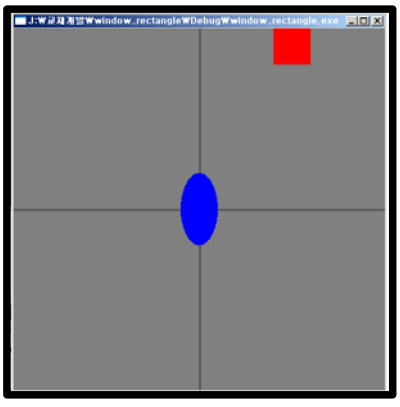
```
gluOrtho2D(-10, 10, -10, 10)
glViewport(0, 0, 500, 500)
glutInitWindowPosition(100, 200);
glutInitWIndowSize(500, 500);
```



예제 2) gluOrtho2D(-5, 5, -10, 10)

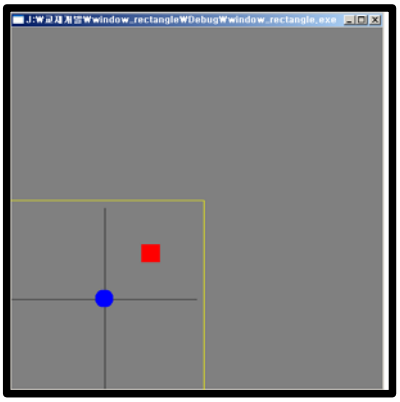


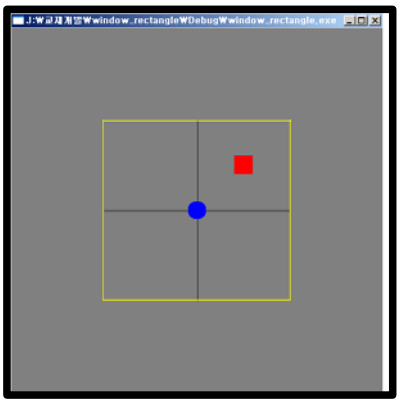
gluOrtho2D(-10, 10, -5, 5)



예제 3) gluOrtho2D(-10, 10, -10, 10)
glViewport(0, 0, 250, 250)

gluOrtho2D(-10, 10 -10, 10)
glViewport(250, 250, 250, 250)





3.2 Geometric Primitives

OpenGL에서 사용되는 기본적인 기하학적 프리미티브들은 다음의 세 개로 구성된다.

- **Points**
- **Line Segments**
- **Filled Polygon**

점은 그 자체 하나의 점으로 그 프리미티브를 나타낼 수 있으며, 선분은 두 개의 점에 의해서 정의되며, 다각형은 여러 개의 점들의 집합으로 구성된다. 그러므로, OpenGL에서는 점들로 표현하는 방법으로 여러 종류의 프리미티브를 나타낼 수 있다. glBegin()함수와 glEnd()함수 사이에 원하는 수의 정점을 정의하면 원하는 형태의 프리미티브를 정의하게 되며 윈도우에 그려지게 된다. 먼저 점들을 그리는 코드를 살펴보자.

Drawing 2-D Points:

```
extern Vec2 v[k];

glBegin(GL_POINTS);
  for(int i=0; i<k; i++)
    glVertex2dv(v[i]);
glEnd();
```

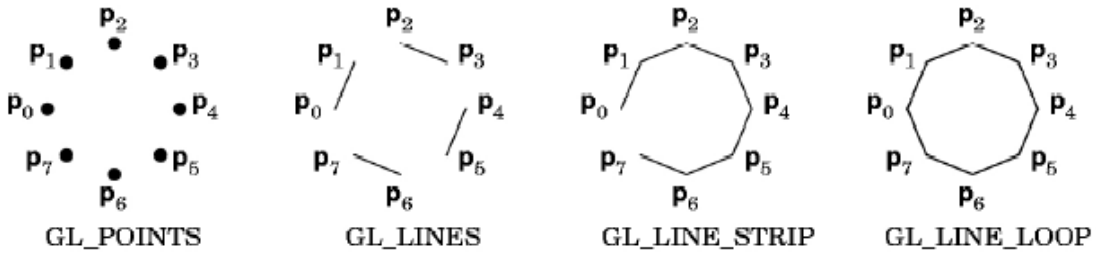
Drawing 3-D Points:

```
extern Vec3 v[k];

glBegin(GL_POINTS);
  for(int i=0; i<k; i++)
    glVertex3dv(v[i]);
glEnd();
```

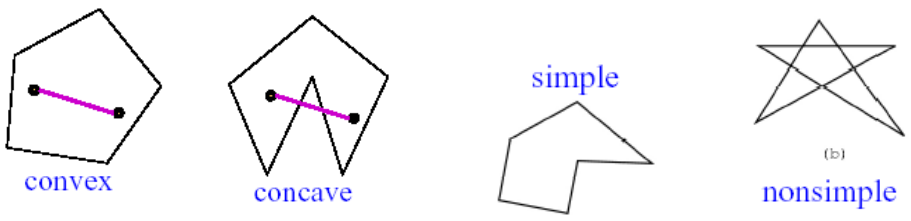
그리려는 것이 점집합이기 때문에 glBegin() 함수의 인자가 **GL_POINTS** 가 되어야 한다. 위 예제의 코드에서는 점들이 벡터 형태로 표현되어 있기 때문에 점을 정의하는 함수가 벡터타입을 사용하고 있음을 확인할 수 있다. 선분을 그리려고 한다면 다음의 코드로 변경하면 된다. 점을 그리는 코드와 달라진 부분이 무엇인지 살펴보자. 우선 인자가 **GL_LINES** 로 바뀌었고 정점들이 짝수 개 정의되어 있어야 한다.

```
glBegin(GL_LINES);
   glVertex2f(0.0, 0.0);
   glVertex2f(1.0, 0.0);
   glVertex2f(0.0, 0.0);
   glVertex2f(0.0, 1.0);
glEnd();
```

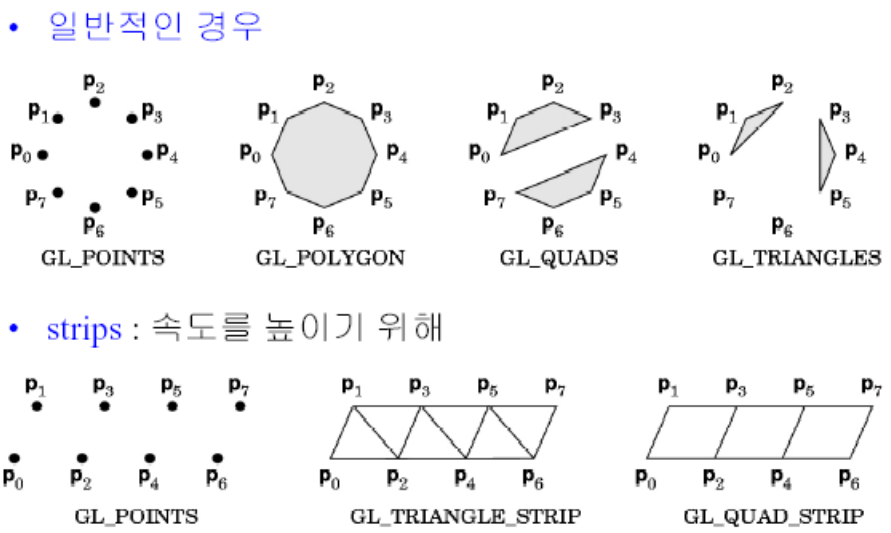
다음은 다양한 종류의 선분 프리미티브를 정의하는 방법에 대해서 살펴보도록 한다.



다각형은 일반적으로 Loop과 Interior을 합한 것을 말하게 되는데 내부가 존재한다고 보면 된다. 즉, 어떠한 색상으로 채워질 수 있는 프리미티브이다. 컴퓨터그래픽스에서는 3차원에서의 렌더링 Quality를 위하여 몇 가지 가정을 적용한다. 사용하는 다각형은 simple하고 flat하며 convex하다고 가정한다. 다각형이 simple 하다는 것은 다각형의 선분들끼리의 교차는 정점에서만 일어난다는 것인데, 별 모양을 한붓그리기로 그리게 되면 선분의 내부 점에서 교차가 일어나는데, 이와 같은 다각형은 피하겠다는 의미이다. 다각형이 Flat하다는 것은 2차원 공간에서는 사실이지만 3차원 공간에서는 사실이 아닐 수도 있다. 예를 들면, 3차원 공간에서 4개의 정점에 의해서 만들어지는 다각형은 일반적으로 하나의 평면에 놓여 있다고 보기는 힘들다. 왜냐하면, 3개의 점정이 하나의 평면을 구성하기 때문에 특별히 나머지 1개의 정점이 그 평면에 놓일 수도 있지만 그렇지 않을 확률이 더 많다고 볼 수 있다. 그러므로 하나의 다각형이 2개의 평면으로 이루어지는 경우가 발생하게 되는 셈이다. 따라서 이런 문제를 미리 방지하기 위해서 다각형이 하나의 평면에 놓인다고 가정하게 된다. 또한 다각형이 오목(concave)하다면 모양이 여러 다각형의 합집합으로 인식될 수도 있기 때문에 볼록 형태의 모양을 사용한다. 따라서 convex, simple, flat 하다는 조건을 다 만족하는 것은 삼각형이므로 컴퓨터그래픽스에서는 대부분의 물체들을 삼각 메수 형태로 물체를 표현하는 데이터 구조를 사용한다.



다각형을 그리는 일반적인 프리미티브 코드는 기본적으로 glBegin()함수와 glEnd()함수로 사용되며 glBegin()함수의 인자로 들어가는 Type에 따라 사용된 정점들로부터 다양한 형태의 다각형을 만들 수 있다.



3.3 Geometric Attributes

점, 선분, 다각형을 그리기 위해서는 glBegin()함수와 glEnd()함수 사이에 원하는 정점의 좌표를 glVertex[n][f]() 타입의 함수를 사용하여 입력을 하면 외형은 일단 만들어지는 것이라 생각할 수 있다. 그런데 사용자의 취향이나 용도에 따라 다른 속성을 가질 때가 있다. 이러한 속성을 **Attribute** 라고 한다. 정점에 대한 속성들에는 점의 크기와 점의 색상이 있고, 선분의 속성에는 선분의 색상, 선분의 굵기, 선분의 종류(실선, 점선, 등)가 있다.

3.3.1 Point의 Attributes

모든 프리미티브에 색상을 지정하기 위해서는 나타내고자 하는 정점 바로 앞에 원하는 색상을 다음과 같은 함수를 사용하여 지정한다.

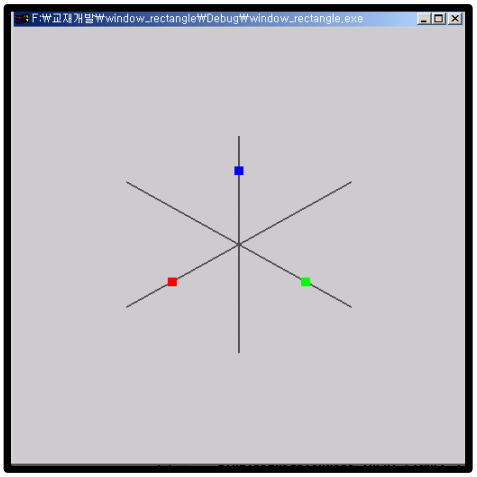
```
glColor3f(float r, float g, float b);
```

모든 인자의 값의 범위는 [0.0, 1.0]이며 첫 번째 인자는 빨강색을 나타내며, 두 번째에는 초록색, 세 번째 인자에는 파랑색을 나타내는 인자들이며 모든 인자가 1.0이면 흰색에 해당되며 모든 인자가 0.0이면 나타내고자 하는 색상은 검정색이다. 다음 코드는 세 정점을 빨강, 초록, 파랑색으로 표현하는 코드이다.

```
glBegin(GL_POINTS);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
```

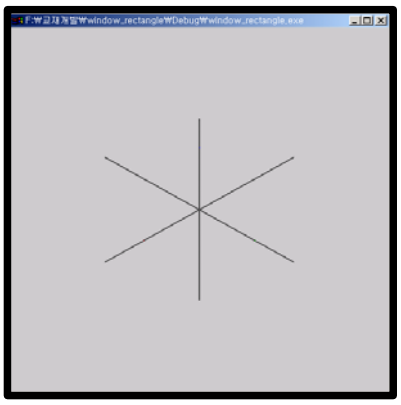
```
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, 1.0);
    glEnd();
```

x축의 (1.0, 0.0, 0.0)에 빨강색의 정점이 있고, y축의 (0.0, 1.0, 0.)에 초록색의 정점이 있고, z축의 (0.0, 0.0, 1.0)에 파랑색의 정점이 있다.



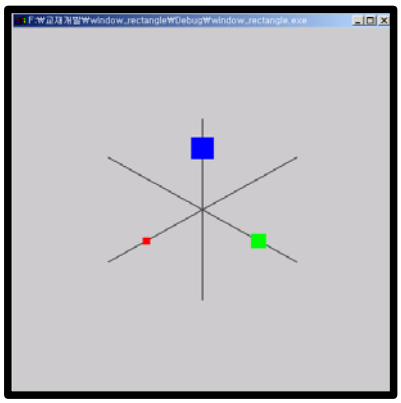
정점의 크기가 동일한 것을 알 수 있다. 정점의 크기를 조절하는 함수는 glPointSize(float size)이다. 이를 이용하여 빨강색 점의 크기를 10, 초록색 점의 크기를 20, 파랑색 점의 크기를 30으로 하는 결과를 얻고자 한다. 쉽게 생각하여 각 정점 앞에 크기조절 함수를 다음과 같이 위치시키면 원하지 않는 결과를 얻게 된다.

```
glBegin(GL_POINTS);
        glColor3f(1.0, 0.0, 0.0);
        glPointSize(10.0);
        glVertex3f(1.0, 0.0, 0.0);
        glColor3f(0.0, 1.0, 0.0);
        glPointSize(20.0);
        glVertex3f(0.0, 1.0, 0.0);
        glColor3f(0.0, 0.0, 1.0);
        glPointSize(30.0);
        glVertex3f(0.0, 0.0, 1.0);
glEnd();
```



올바른 결과인 다른 크기의 세 정점을 얻기 위해서는 다음과 같은 코드로 작성해야 한다.

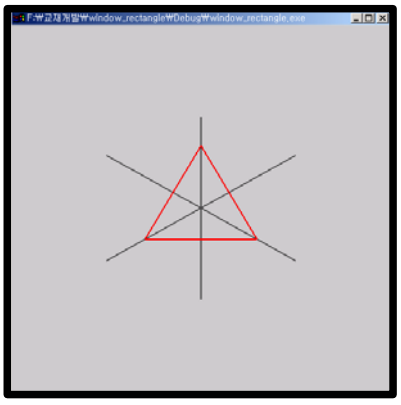
```
glPointSize(10.0);
glBegin(GL_POINTS);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
glEnd();
glPointSize(20.0);
glBegin(GL_POINTS);
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
glEnd();
glPointSize(30.0);
glBegin(GL_POINTS);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, 1.0);
glEnd();
```



3.3.2 Line 의 Attributes

이제 세 정점을 이어 삼각형의 외형을 만들어보자. 세 개의 선분은 빨강색으로 이루어지도록 하자. 이를 위한 코드는 다음과 같다.

```
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_LINES);
        glVertex3f(1.0, 0.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, 1.0);
        glVertex3f(1.0, 0.0, 0.0);
glEnd();
```

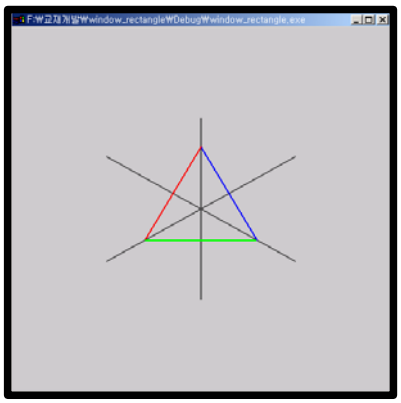


이 코드를 살펴보면 각 정점의 좌표를 이중으로 사용하고 있다. 이를 효율적으로 처리하기 위하여 LINE_LOOP 타입을 사용하면 다음과 같은 간단한 코드로 교체될 수 있다.

```
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_LINE_LOOP);
        glVertex3f(1.0, 0.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 0.0, 1.0);
glEnd();
```

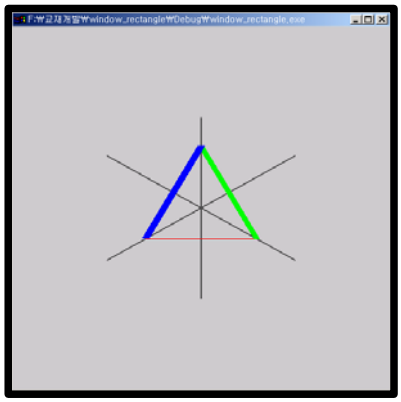
이제 각 선분에 다른 색을 입혀 보자

```
glBegin(GL_LINE_LOOP);
        glColor3f(1.0,  0.0,  0.0);
        glVertex3f(1.0,  0.0,  0.0);
        glColor3f(0.0,  1.0,  0.0);
        glVertex3f(0.0,  1.0,  0.0);
        glColor3f(0.0,  0.0,  1.0);
        glVertex3f(0.0,  0.0,  1.0);
glEnd();
```



이제 각 선분의 굵기를 달리하여보자 이를 위하여 glLineWidth(float width) 함수를 사용한다.

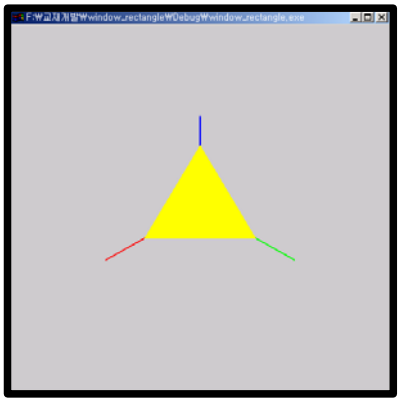
```
glColor3f(1.0,  0.0,  0.0);
glLineWidth(1.0);
glBegin(GL_LINES);
        glVertex3f(1.0,  0.0,  0.0);
        glVertex3f(0.0,  1.0,  0.0);
glEnd();
glColor3f(0.0,  1.0,  0.0);
glLineWidth(8.0);
glBegin(GL_LINES);
        glVertex3f(0.0,  1.0,  0.0);
        glVertex3f(0.0,  0.0,  1.0);
glEnd();
glColor3f(0.0,  0.0,  1.0);
glLineWidth(20.0);
glBegin(GL_LINES);
        glVertex3f(0.0,  0.0,  1.0);
        glVertex3f(1.0,  0.0,  0.0);
glEnd();
```



3.3.3 Polygon 의 Attributes

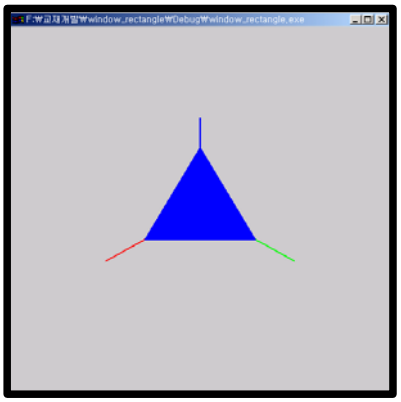
삼각형 내부를 채워서 만들어지는 것이 다각형이다. 이제 다각형의 내부에 원하는 색을 채우는 방법을 생각해보자 하나의 색상을 사용하는 경우와 여러 색을 사용하는 경우도 있을 것이다. x축은 빨강색, y축은 초록색, z축은 파랑색으로 나타내어지고 세 점에 의해서 만들어지는 삼각형에 노랑색을 입혀 본다.

```
glColor3f(1.0, 1.0, 0.0);
glBegin(GL_TRIANGLES);
        glVertex3f(1.0, 0.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 0.0, 1.0);
glEnd();
```

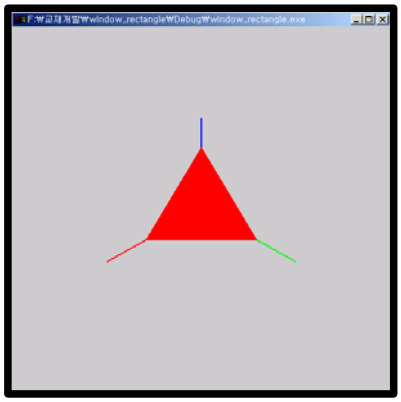


각 정점마다 다른 색을 부여한다고 하면 삼각형 내부는 어떤 색으로 채워질까 ? 먼저 각 정점 앞에 색상을 부여하는 코드를 실행시켜 보자. 프리미티브 타입을 GL_TRIANGLE 로 하게 되면 세 번째 정점의 색상을 읽어서 그 색상으로 삼각형 내부를 채우게 되고, GL_POLYGON으로 하게 되면 첫 번째 정점의 색상으로 그 다각형 내부를 채우게 됨을 알 수 있다.

```
glBegin(GL_TRIANGLES);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, 1.0);
glEnd();
```

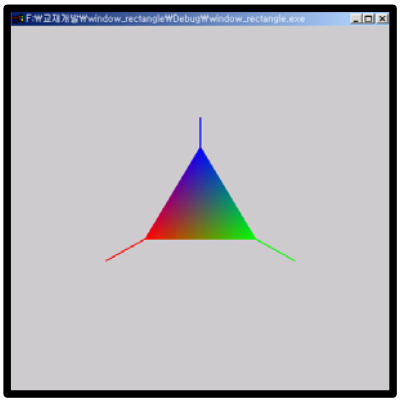


```
glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, 1.0);
glEnd();
```



다각형 내부의 각 지점에 다른 색으로 칠하고 싶다면, 즉 내부의 점들은 세 점의 색상에 의해서 적당히 섞이게 하는 방법이 있다. Shading 방법을 Smooth로 지정하게 되면 각 정점의 색상들을 bilinear interpolation을 통하여 삼각형 내부의 점들의 색상이 위치에 따라 변하게 하는 방법이다. 이에 대한 코드는 **glShadeModel(GL_SMOOTH);** 라는 함수를 한번 만 적용하면 된다.

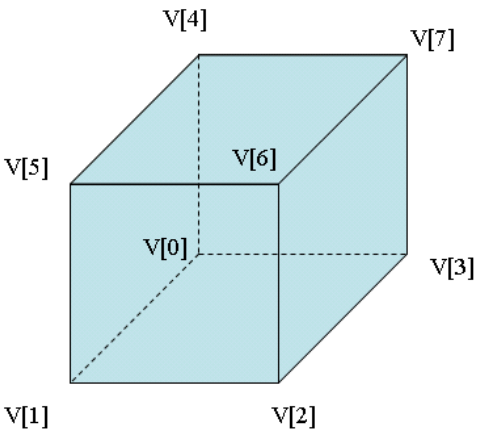
```
glShadeModel(GL_SMOOTH);
glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, 1.0);
glEnd();
```



3.4 Polyhedron Construction

2차원 개념의 다각형들을 이용하여 3차원 다면체를 만들 수 있을 것이다. 우리가 쉽게 만들 수 있다고 생각되는 정육면체를 만들어 보자. 정육면체는 정사각형이 여섯 개로 구성되는 물체이기 때문에 정점 4개로 구성되는 정사각형을 그리는 코드를 여섯 번 적용하면 될 것 같다. 정육면체에 사용되는 정점의 개수는 8이며 이들에 대한 좌표값을 알아야 한다.

```
v[0] = {-1.0, -1.0, -1.0};
v[1] = { 1.0, -1.0, -1.0};
v[2] = { 1.0,  1.0, -1.0};
v[3] = {-1.0,  1.0, -1.0};
v[4] = {-1.0, -1.0,  1.0};
v[5] = { 1.0, -1.0,  1.0};
v[6] = { 1.0,  1.0,  1.0};
v[7] = {-1.0,  1.0,  1.0};
```



여섯 개의 각 면은 정점 4개에 의해서는 만들어지는 정사각형이며 이들을 표현하기 위하여 정점의 순서가 매우 중요하다. 컴퓨터그래픽스에 사용되는 정점의 순서는 시계 반대방향인데, 이러한 방향은 면의 법선 벡터가 바깥쪽으로 향하도록 하기 위해서이기도 하다. 왜냐하면 solid와 같은 닫힌 다면체의 내부는 아무런 정보를 제공하지 않기 때문에 바깥면만 중요하게 생각한다. 경우에 따라서는 내부 정보도 필요로 하는데, 예를 들면 의료영상 같은 곳에서는 Voxel data를 사용하기 때문에 내부 정보도 매우 중요하게 작용한다. 정육면체의 여섯 개의 면은 다음과 같이 구성된다고 볼 수 있다.

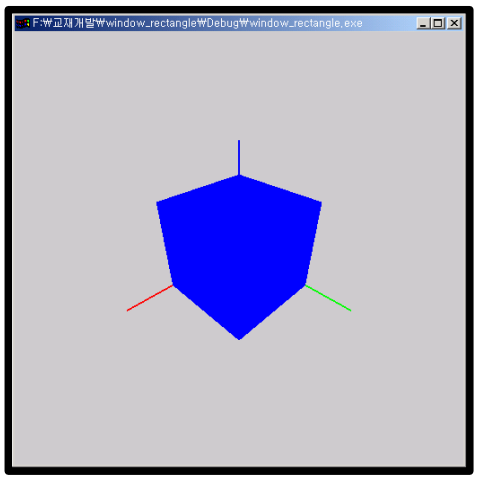
```
f[0] = {1, 2, 6, 5};
f[1] = {2, 3, 7, 6};
f[2] = {3, 0, 4, 7};
f[3] = {4, 5, 6, 7};
f[4] = {3, 2, 1, 0};
f[5] = {0, 1, 5, 4};
```

자 이제 육면체를 그려보도록 하자. 먼저 위와 같은 데이터 구조를 갖는 사각형을 그리는 함수를 다음과 같이 정의한다.

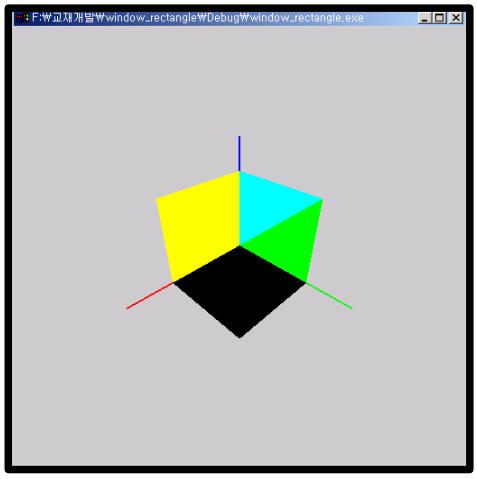
```
void Draw_Rectangle(int a, int b, int c, int d) {
        glBegin(GL_POLYGON);
                glVertex3fv(v[a]);
                glVertex3fv(v[b]);
                glVertex3fv(v[c]);
                glVertex3fv(v[d]);
        glEnd();
}
```

그러면 여섯 개의 면을 호출하는 코드를 이용하여 육면체를 그릴 수 있을 것 같다.

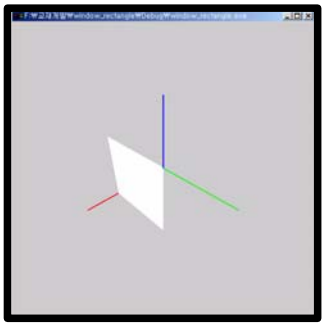
```
void Draw_Cube(void) {
        for ( int I=0 ; I < 6 ; I++ )
                Draw_Rectangle( f[I][0], f[I][1], f[I][2], f[I][3] );
}
```

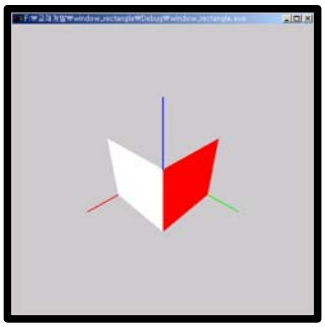


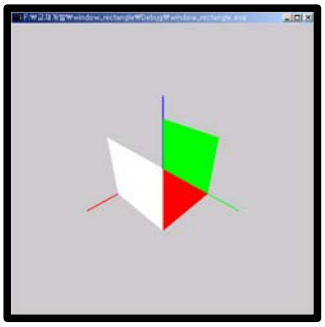
모든 면에 동일한 파랑색을 부여하게 되면 정육면체 형태가 아닌 육각형 형태의 도형만을 볼 수 있게 된다. 즉 2차원 느낌이 들지 3차원 형태는 아닌 것으로 오해 받게 되는 셈이다. 3차원 형태처럼 보이게 하기 위해서 모든 면에 다른 색상을 부여 하도록 한다. f[0]에는 흰색, f[1]에는 빨강, f[2]에는 초록, f[3]에는 파랑, f[4]에는 검정색, f[5]에는 노랑색을 부여해서 그리면 다음과 같은 결과를 얻게 된다.

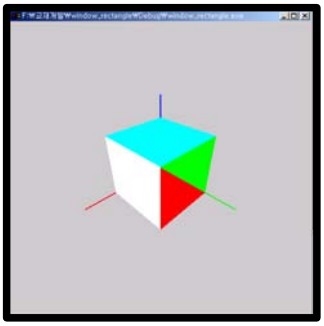


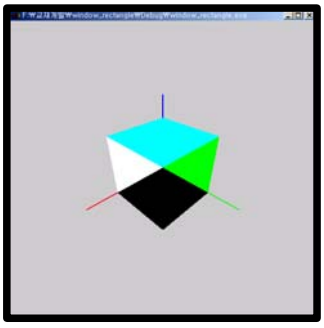
이 결과를 잘 보면 정육면체라고 믿기에는 모양이 좀 이상하다. 왜 이런 결과가 나왔을까 ? 정육면체를 그리는 함수에서 f[0], f[1], … , f[5] 순서로 그려진다.

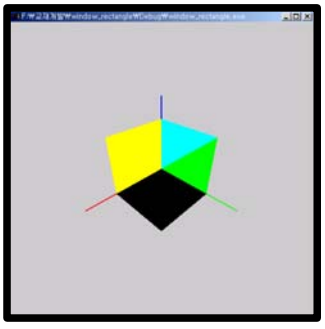

f[0]


f[0], f[1]

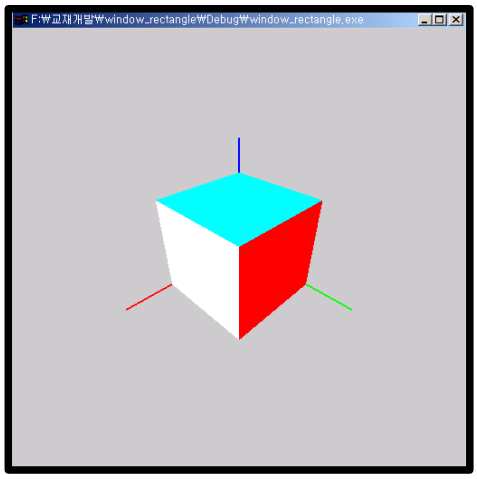

f[0], f[1], f[2]


f[0], f[1], f[2], f[3]

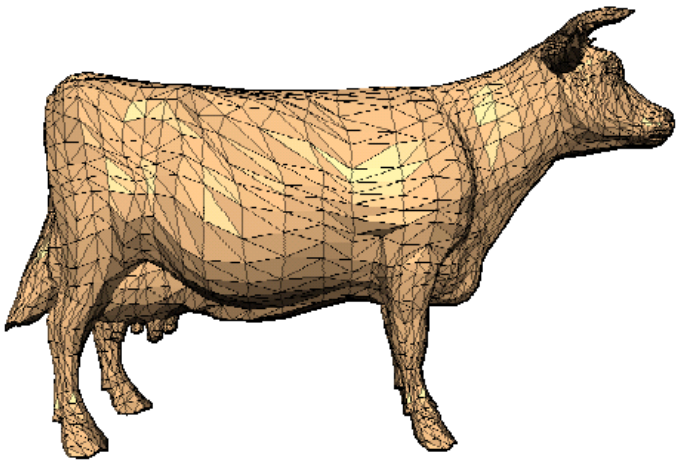

f[0], f[1], f[2], f[3], f[4]


f[0], f[1], f[2], f[3], f[4], f[5]

사각형 그리는 순서를 f[2], f[4], f[5], f[0], f[1], f[3] 으로 바꾸어 보자.



이번 실습을 통해서 우리가 얻을 수 있는 것은 OpenGL에서 그림을 그릴 때 그리는 순서에 영향을 받는다는 사실이다. 특히 3차원 물체를 그리고자 한다면 카메라 위치에서 멀리 놓여 있는 면부터 그리고 가까운 면은 나중에 그리는 노력이 필요함을 알 수 있다. 그런데 만약 다음의 소와 같은 매우 복잡한 물체는 그러한 면들의 순서를 매길 수 없기 때문에 자동으로 처리할 수 있는 방법이 필요하게 된다.



3차원 물체를 2차원 화면으로 투영시켜 표현하려고 할 때 중요하게 고려하여야 하는 것이 카메라로부터 보이는 면은 그리고 보이지 않는 면은 그리지 않아야 한다. 이런 문제를 다루는 것이 은면 제거 알고리즘(Hidden-Surface Removal Algorithm)이다. 이러한 문제에 대한 일반적인 해결책은 하나의 버퍼를 추가로 사용하여 보이는 물체의 카메라로부터의 거리값을 저장하는 Z-버퍼를 이용하는 Z-버퍼 알고리즘이 있다. 다음은 Z-버퍼 알고리즘을 적용한 육면체를 거리는 코드이다. 함수 glEnable(GL_Depth_Test)를 사용하면 그리고자 하는 면들의 순서와는 무관한 결과를 얻을 수 있다. 즉 보이지 않는 물체는 그리지 않고 보이는 면만 나타나는 결과를 얻게 된다.

```
void Draw_Rectangle(int a, int b, int c, int d) {
        glBegin(GL_POLYGON);
                glVertex3fv(v[a]);
                glVertex3fv(v[b]);
                glVertex3fv(v[c]);
                glVertex3fv(v[d]);
        glEnd();
}

void Draw_Cube(void) {
        for ( int I=0 ; I < 6 ; I++ )
                Draw_Rectangle( f[I][0], f[I][1], f[I][2], f[I][3] );
}

void Display(void) {
        glEnable(GL_Depth_Test);
        Draw_Cube();
}
```