

# 그래픽스 강의노트 03 - OpenGL 소개

강영민

동명대학교

2015년 2학기

# OpenGL

- OpenGL은 특정한 하드웨어나 운영체제에 의존하지 않고 다양한 시스템에 이식(移植)될 수 있는 개방형 라이브러리
- OpenGL을 통한 학습은 실시간 그래픽스에 대한 이해를 돋고, 다양한 시스템에 적용가능한 그래픽스 프로그래밍 기술을 습득하게 함

# OpenGL을 사용하기 위한 준비

- Mac OS X, Linux - 특별한 준비가 필요 없음
- MS Windows - 플랫폼 독립적 윈도우 생성을 위해 glut를 따로 설치해야 함
- 수업에 사용할 glut 라이브러리
  - freeglut
  - 다운로드 - precompiled binary는 64비트
  - 32비트와 64비트 용으로 컴파일한 결과를 수업 홈페이지에 게시

# MS Windows 환경에서 freeglut 설치

- 32비트
  - freeglutd.dll, freeglut.dll → C:\Windows\System32
  - 헤더 파일들 → (Windows SDK)\include\GL
  - 라이브러리 파일들 (freeglutd.lib, freeglut.lib) → (Windows SDK)\lib
- 64비트
  - freeglutd.dll, freeglut.dll → C:\Windows\SystemWOW64
  - 헤더 파일들 → (Windows SDK)\include\GL
  - 라이브러리 파일들 (freeglutd.lib, freeglut.lib) → (Windows SDK)\lib\x64
- 프로젝트의 플랫폼을 Win32가 아니라 64비트로 변경하여 생성하여야 함

# 간단한 OpenGL 프로그램 테스트

```
#include "headers.h"

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main ( int argc , char * argv [] ) {
    glutInit(&argc , argv );
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
    glutInitWindowPosition(0 , 0);
    glutInitWindowSize(512 , 512);
    glutCreateWindow("A Triangle"); // 윈도우 생성

    glClearColor(1.0 , 0.0 , 0.0 , 1.0);
    glutDisplayFunc(myDisplay); // 디스플레이 콜백 등록
    glutMainLoop(); // 이벤트 루프
    return 0;
}
```

# OpenGL의 특징

- 실시간 그래픽 라이브러리로서 사실상의 (*de facto*) 산업 표준
- 플랫폼에 독립적
  - OpenGL을 이용하여 작성한 그래픽 프로그램은 여러 종류의 다른 운영체제를 가진 시스템에 쉽게 이식
- OpenGL은 상태 기계(state machine)
  - OpenGL의 동작은 현재의 상태에 의해 결정
  - 이 상태는 변경하지 않으면 계속해서 유지
  - OpenGL에는 많은 종류의 상태 변수가 있으며, 이 상태는 한 번 설정하면 다시 변경하지 않는 한 계속해서 설정된 상태를 유지
  - 예: 선의 두께를 결정했다면, 이후에 이 두께를 변경하지 않는 이상 모든 선이 설정된 두께로 그려짐

# OpenGL의 관례

- OpenGL API의 명령들은 gl-Command-dimension-type의 꼴
  - glVertex3f는 정점(Vertex)의 위치를 설정하는 OpenGL 명령으로 3차원 데이터이며, 각 차원의 값을 부동소수점(floating point)로 표현한다는 의미
- OpenGL 명령어의 뒤에 붙어 입력 파라미터의 자료형을 표시하는 접미사는 표와 같음

f	32 비트 부동소수점	float	GLfloat
d	64 비트 부동소수점	double	GLdouble
b	8 비트 정수	char	GLbyte
ub	8 비트 부호 없는 정수	unsigned char	GLubyte
i	32 비트 정수	int or long	GLint
ui	32 비트 부호 없는 정수	unsigned int long	GLuint, GLenum
s	16 비트 정수	short	GLshort

# OpenGL의 관례

- OpenGL API의 명령들은 gl-Command-dimension-type의 꼴
  - glVertex3f는 정점(Vertex)의 위치를 설정하는 OpenGL 명령으로 3차원 데이터이며, 각 차원의 값을 부동소수점으로 표현한다는 의미
- 입력 파라미터의 자료형을 표시하는 접미사는 표와 같음

f	32 비트 부동소수점	float	GLfloat
d	64 비트 부동소수점	double	GLdouble
b	8 비트 정수	char	GLbyte
ub	8 비트 부호 없는 정수	unsigned char	GLubyte
i	32 비트 정수	int or long	GLint
ui	32 비트 부호 없는 정수	unsigned int long	GLuint, GLenum
s	16 비트 정수	short	GLshort

## GLU(GL Utility)와 GLUT(GL Utility Toolkit)

카메라 등을 다루는 OpenGL 유ти리티 라이브러리의 명령들은 glu로 시작한다. 또한 서로 다른 윈도 환경에 독립적인 윈도 관리를 지원하는 OpenGL 유ти리티 툴킷 (utility toolkit)을 사용하는데, 이 툴킷의 명령들은 glut로 시작한다.

# OpenGL 프로그램의 기본 형태

```
#include <whatever you want>
callback_for_display() {
    for(그려질 모든 객체에 대해) {
        변환 설정;
        glBegin(그리기 프리미티브 지정);
        [[정점(vertex) 정보 제공;]]
        glEnd();
    }
    glFlush() 또는 glutSwapBuffers();
}

void main(int argc, char **argv) {
    [[윈도우 초기화;]]
    glMatrixMode(GL_PROJECTION);
    [[투영 행렬 설정;]]
    glMatrixMode(GL_MODELVIEW);
    [[카메라의 위치와 방향 잡기;]]
    [[콜백 함수의 등록;]]
    [[메인 루프로 들어가기;]]
}
```

# 간단한 OpenGL 프로그램의 예

Lines 1–25 / 50

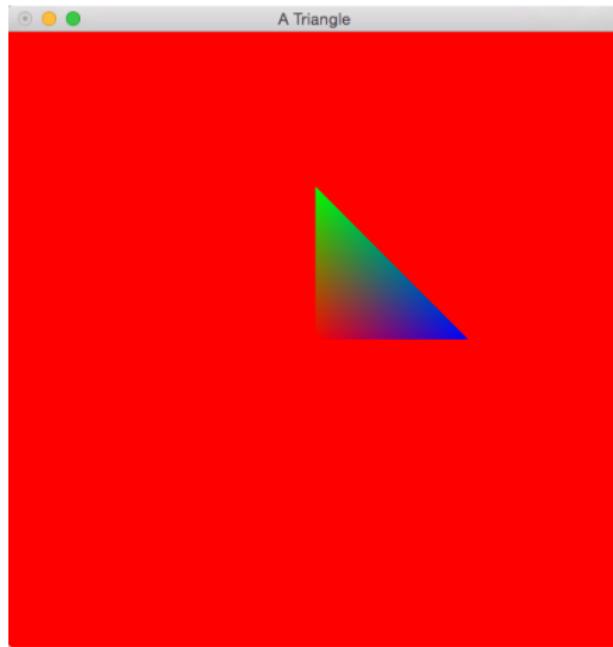
```
#ifdef WIN32 // window
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#else // mac
#include <OpenGL/OpenGL.h>
#include <GLUT/GLUT.h>
#endif
void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glColor3f(0.0, 0.0, 1.0);
    glVertex3f(0.5, 0.0, 0.0);
    glColor3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.5, 0.0);
    glEnd();
    glFlush();
}
int main ( int argc, char * argv [] ) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(512, 512);
```

# 간단한 OpenGL 프로그램의 예

Lines 26–50 / 50

```
glutCreateWindow("A Triangle"); // 윈도우 생성  
glClearColor(1.0, 0.0, 0.0, 1.0);  
glutDisplayFunc(myDisplay); // 디스플레이 콜백 등록  
glutMainLoop();           // 이벤트 루프로  
return 0;  
}
```

# 간단한 OpenGL 프로그램의 예



# 프리미티브(primitives) - 1/3

- 그래픽 하드웨어는 프로그래머(programmer)가 지정한 프리미티브(primitive) 설정에 따라 정점의 리스트를 처리
- 프리미티브는 OpenGL이 제공하는 그리기 기본요소
- 입력 정점들을 어떻게 조합할 것인가를 결정
- 프리미티브를 사용하는 방법은 다음과 같다.

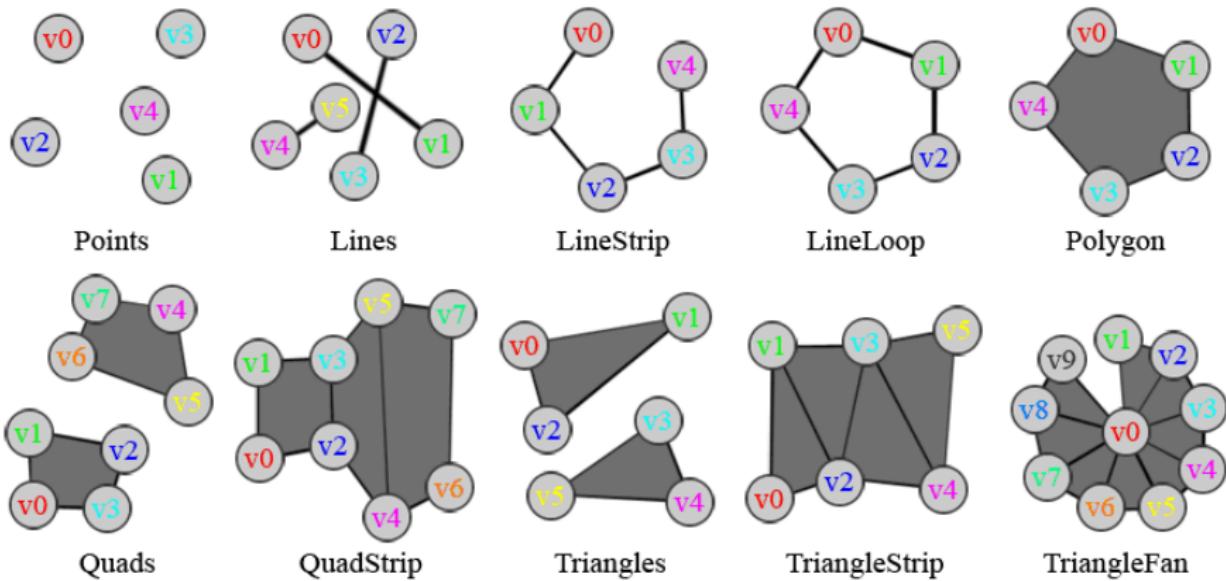
```
glBegin (drawing primitive);
    // vertex position , color , normal , etc
    setVertexInfo ();
glEnd ();
}
```

# 프리미티브(primitives) - 2/3

- GL\_POINTS: 입력된 정점을 하나씩 점으로 가시화
- GL\_LINES: 입력된 정점을 두 개씩 묶어 선분으로 표현
- GL\_LINE\_STRIP: 입력된 정점을 차례대로 연결하여 하나의 폴리라인(polyline)을 구성
- GL\_LINE\_LOOP: 입력된 정점을 차례로 연결한 뒤에 마지막 점을 시작점으로 연결
- GL\_TRIANGLES: 입력된 정점을 세 개씩 묶어 삼각형을 그림
- GL\_TRIANGLE\_STRIP: 처음 세 개 정점으로 삼각형을 그린 뒤, 정점이 추가될 때마다 삼각형을 직전 두 개 정점과 연결하여 삼각형 추가
- GL\_TRIANGLE\_FAN: 부채 모양으로 삼각형을 추가해 나감
- GL\_QUADS: 정점 네 개씩을 묶어 사각형 그리기
- GL\_QUAD\_STRIP: 처음 네 개 정점으로 사각형 그리고, 이후 두 개씩 묶어 직전 두 개 정점과 함께 사각형 그리기
- GL\_POLYGON: 입력된 모든 정점으로 다각형을 그림

# 프리미티브(primitives) - 3/3

Geometric Primitive Types in OpenTK.OpenGL (defined Clockwise)



# 정점 데이터 설정 방법

- 정점 데이터는 위치와 법선벡터, 색 등
- 정점의 위치만을 입력한다면 glVertex[dim-type]으로 입력
- 3 차원 정점의 각 성분을 부동소수점 표현으로 넣는다면,  
glVertex3f(x,y,z)와 같이 입력
- 다음과 같은 같은 여러 표현이 가능하다.

```
float x, y, z;
double dx, dy, dz;
int ix, iy, iz;
float verts = {1.0f, 2.0f, 1.0f};
glBegin(drawing primitive);
    glVertex3f(x, y, z);
    glVertex3d(dx, dy, dz);
    glVertex3i(ix, iy, iz);
    glVertex3fv(verts);
glEnd();
```

# 다양한 프리미티브를 이용한 풍경 그리기

Lines 1–25 / 75

```
...
#include <math.h>
void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    // 색상의 지정
    glColor3f(0.0, 1.0, 0.0);

    // 삼각형 그리기로 지정
    glBegin(GL_TRIANGLES);
    // mountain 1
    glVertex2f(-0.75, -0.25);
    glVertex2f(0.0, 0.25);
    glVertex2f(0.25, -0.25);
    // mountain 2 - 그리기 색상 변경
    glColor3f(0.5, 0.5, 0.1);
    glVertex2f(-0.25, -0.25);
    glVertex2f(0.75, 0.25);
    glVertex2f(1.0, -0.25);
    glEnd();

    // 사각형 그리기로 지정
    glBegin(GL_QUADS);
    // roof 파란색으로 색상 변경
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f(-1.0, 0.25);
```

# 다양한 프리미티브를 이용한 풍경 그리기

Lines 26–50 / 75

```
glVertex2f( -0.75,  0.5);
glVertex2f( -0.25,  0.5);
glVertex2f( 0.0,   0.25);
// house 노란색으로 색상 변경
glColor3f(1.0, 1.0, 0.0);
glVertex2f( -0.75,  0.25);
glVertex2f( -0.75, -0.25);
glVertex2f( -0.25, -0.25);
glVertex2f( -0.25,  0.25);
// tree 갈색으로 변경
glColor3f(0.7, 0.5, 0.0);
glVertex2f( 0.5,  0.25);
glVertex2f( 0.75,  0.25);
glVertex2f( 0.75, -0.25);
glVertex2f( 0.5, -0.25);
glEnd();

// 입력된 정점을 모두 이용하는 다각형 그리기로 지정
glBegin(GL_POLYGON);
int n=20;
float radius=0.1;
glColor3f(1.0, 1.0, 0.0);
float angle = 0.0; float step=(3.14159*2.0)/n;
// 반복문 내에서 여러 개의 정점 좌표를 계산한 뒤에 지정하는 방식
// 여기서는 원을 이루는 정점들을 계산하고 있음
```

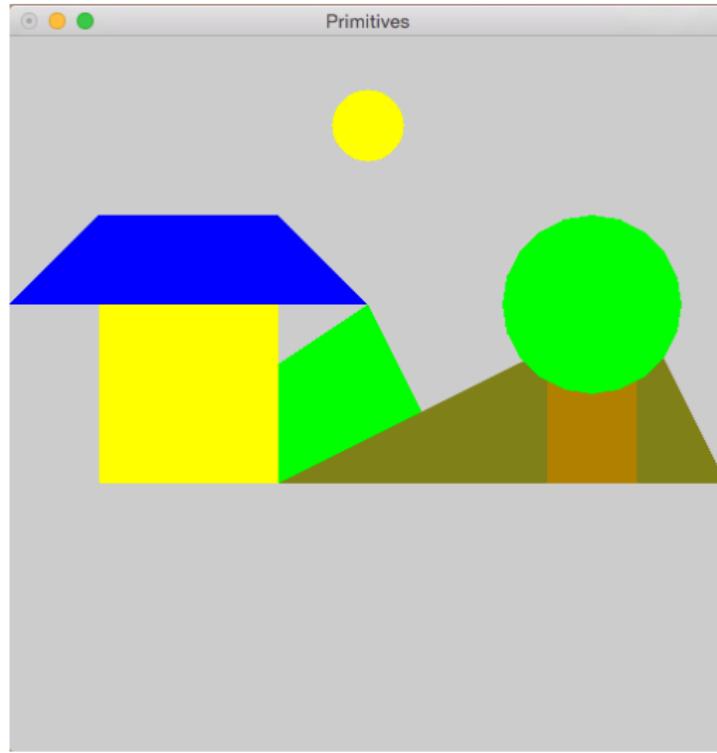
# 다양한 프리미티브를 이용한 풍경 그리기

Lines 51–75 / 75

```
while ( angle < 3.14159*2.0) {
    glVertex2f( radius*cos( angle ) ,  radius*sin( angle )+0.75 );
    angle += step ;
}
glEnd();

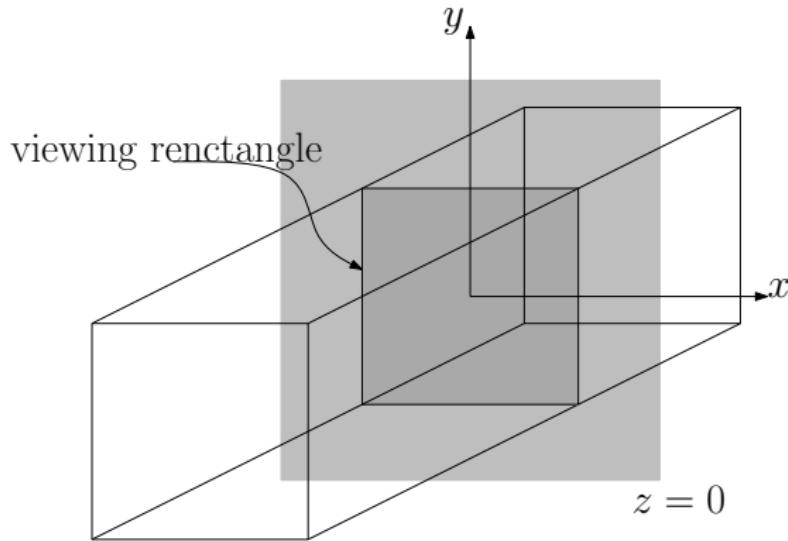
// 원의 중심을 옮기고 반지름을 바꾼 뒤에 다시 그림
glBegin(GL_POLYGON);
n=20;
radius=0.25;
	glColor3f(0.0, 1.0, 0.0);
angle = 0.0; step=(3.14159*2.0)/n;
while (angle < 3.14159*2.0) {
    glVertex2f( radius*cos( angle )+0.625 ,  radius*sin( angle )+0.25 );
    angle += step ;
}
glEnd();
glFlush();
}
```

# 다양한 프리미티브를 이용한 풍경 그리기



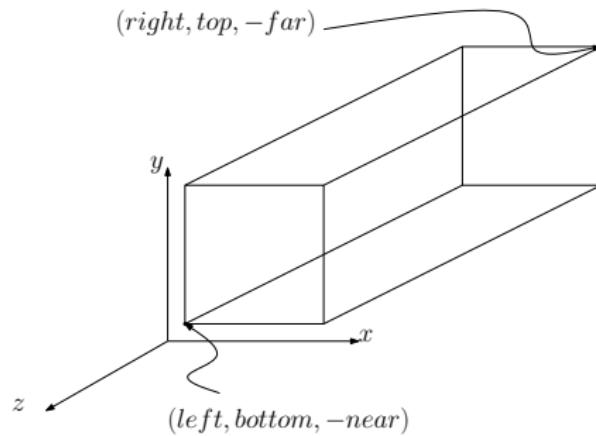
# 디폴트 카메라

- OpenGL의 디폴트 카메라는 원점 위치
- z축 음의 방향
- 원근 투영을 적용하지 않아 카메라에 잡히는 공간은 상자 형태



# 직교 투영 (orthographics projection) 설정

- 디폴트 카메라는 직교 투영 카메라
- 상자 모양의 가시화 공간내의 객체를 상자를 절단하는 면에 투영
- 이 상자의 위치와 길이를 변경하는 함수가 glOrtho



```
glOrtho(float left, float right, float bottom, float top, float near, float far);
```

- 결국 OpenGL의 디폴트 카메라는 다음과 같은 설정
  - glOrtho(-1,1, -1,1, -1,1);

# 원근 투영 (perspective projection) 설정

- 실제 카메라나 우리 눈은 원근이 없는 평행 투영이 불가능
- 멀리 있는 것은 작게 보이고 가까이 있는 것은 크게 보이는 이유는 원근투영
- 이러한 원근 투영을 설정하는 방법은 glFrustum과 gluPerspective
- glFrustum에 대한 이해는 뒤로 미루고 우선 직관적인 gluPerspective 함수를 먼저 이해

`gluPerspective(float fovy, float Aspect, float near, float far);`

- fovy는  $y$ 축 방향으로의 시야각을 도(degree)로 나타낸 것
- Aspect는 가시화 볼륨의 종횡비(aspect ratio)
- near는 카메라에서 상이 맷히는 가까운 평면까지의 거리
- far는 가시화 공간을 결정하는 평면 중 카메라에서 가장 먼 쪽 평면과 카메라 사이의 거리
- 이 함수는 그리기 동작이 일어날 때마다 불리는 것이 아니라 초기에 한 번, 혹은 렌즈를 바꿀 필요가 있을 때에만 불린다.

# 행렬 모드(matrix mode)

- OpenGL은 세 종류의 행렬: 텍스처 행렬, 모델뷰 행렬, 투영행렬
- 모델뷰 행렬은 공간 내에서 가상 객체의 좌표를 변경
- 투영행렬은 가상 객체를 투영면에 옮겨 놓음

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60, 1.0, 0.1, 100.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0);

[[draw something]]
```

- 여기서는 gluPerspective와 gluLookAt이 어떠한 행렬 모드를 변경하는가 하는 것이 관심
- gluPerspective 함수는 투영의 특성을 변경하는 것이므로 투영행렬 모드
- gluLookAt은 카메라의 위치를 옮기는 것이고, 이는 바꾸어 말해 물체의 위치를 카메라 기준에서 옮기는 것이므로 모델뷰 행렬을 변경

# 카메라 위치 변경

- 카메라를 원하는 곳으로 옮겨주는 함수: gluLookAt

```
gluLookAt(float eye_x, float eye_y, float eye_z,  
          float at_x, float at_y, float at_z,  
          float up_x, float up_y, float up_z);
```

- 카메라의 위치를 옮겨 놓는 역학
- 카메라 이동의 반대로 물체를 옮겨 놓는 것
- 카메라의 위치는 매 프레임마다 변경될 수 있으므로 이 함수는 그리기 함수 내에서 매번 불리는 것이 일반적

# 카메라를 설정하는 기본적인 프로그램

Lines 1–25 / 50

[[header 파일 포함하기...]]

```
void init( int argc, char **argv ) {
    // 윈도우 생성, 버퍼 설정
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(512,512);
    glutCreateWindow("Dr. Kang's Graphics Lecture");
    glClearColor(1.0, 1.0, 1.0, 1.0);
    // 카메라 투영 특성 설정 (glPerspective 사용). 이때는 GL_PROJECTION 행렬모드여야 한다.
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1.0, 0.1, 100.0);
}
void drawScene() {
    [[앞서 사용한 코드 ??의 그리기 코드를 여기에 넣는다. (단, glFlush는 여기서 사용하지 않는다)]]
}
void drawAxes() {
    glBegin(GL_LINES);
    glColor3f(1.0, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);    glVertex3f(1.0, 0.0, 0.0);
    glColor3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);    glVertex3f(0.0, 1.0, 0.0);
    glColor3f(0.0, 0.0, 1.0);
}
```

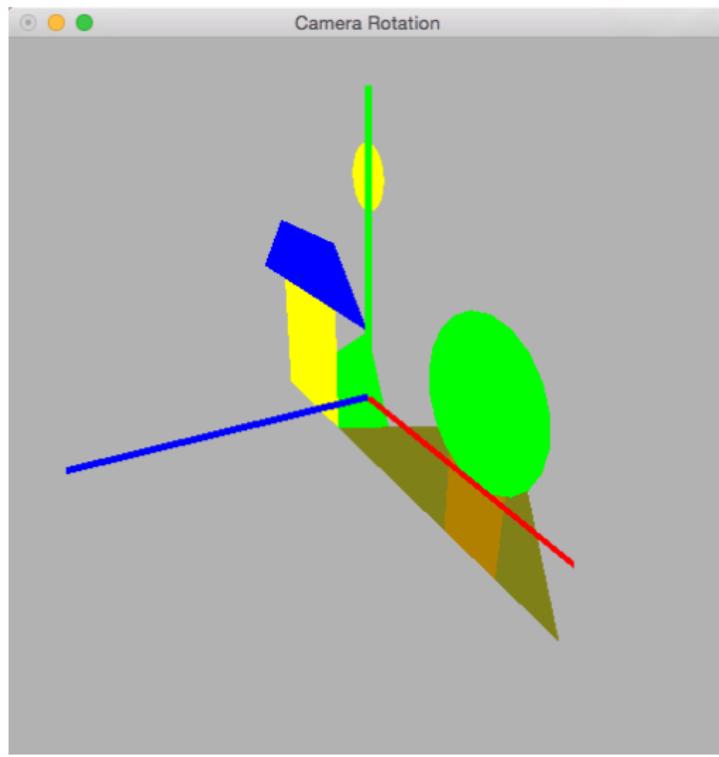
# 카메라를 설정하는 기본적인 프로그램

Lines 26–50 / 50

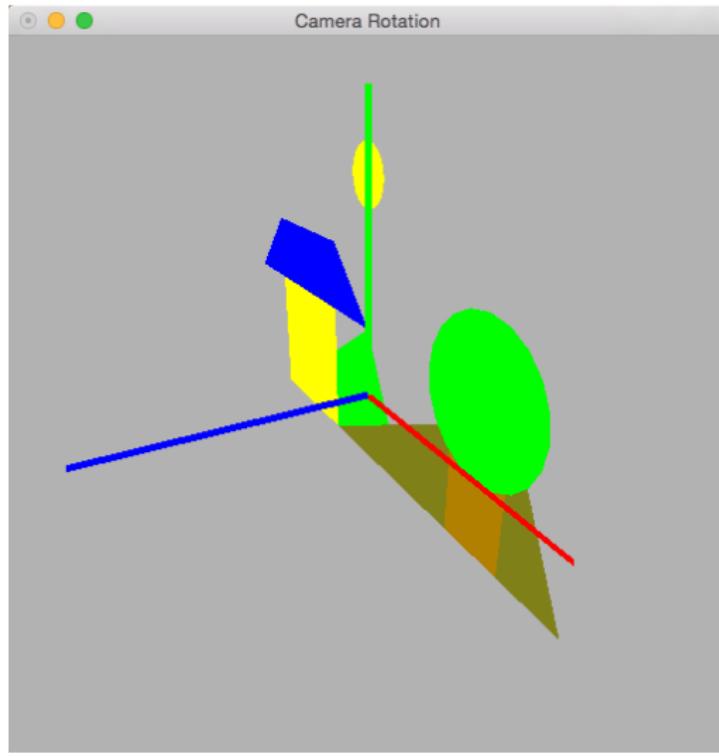
```
glVertex3f(0.0, 0.0, 0.0);    glVertex3f(0.0, 0.0, 1.0);
glEnd();
}
void display() {
    static float t=0.0;
    // 카메라의 위치와 방향을 설정한다. 이때는 GL_MODELVIEW 행렬 모드여야 한다.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt( 2.0*sin(t), 1, 2.0*cos(t), 0, 0, 0, 0, 1, 0);
    t+=0.001;
    glClear(GL_COLOR_BUFFER_BIT);
    drawScene();
    glLineWidth(5);
    drawAxes();
    glLineWidth(1);
    glFlush();
};

int main ( int argc, char **argv ) {
    init(argc, argv);
    glutDisplayFunc(display);
    glutIdleFunc(display);
    glutMainLoop();
}
```

# 카메라 조작 결과



# 카메라 조작 결과



# (깊이버퍼 없이) 네 개의 면으로 상자 모양 그리기

Lines 1–25 / 50

```
void drawScene() {  
    // drawing code  
    glBegin(GL_QUADS);  
    // 천정  
    glColor3f(1.0, 1.0, 0.0);  
    glVertex3f(-0.5, 0.5, -0.5);  
    glVertex3f( 0.5, 0.5, -0.5);  
    glVertex3f( 0.5, 0.5, 0.5);  
    glVertex3f(-0.5, 0.5, 0.5);  
    // 바닥  
    glColor3f(0.0, 1.0, 1.0);  
    glVertex3f(-0.5, -0.5, -0.5);  
    glVertex3f( 0.5, -0.5, -0.5);  
    glVertex3f( 0.5, -0.5, 0.5);  
    glVertex3f(-0.5, -0.5, 0.5);  
    // 왼쪽 벽  
    glColor3f(0.0, 1.0, 0.0);  
    glVertex3f(-0.5, 0.5, -0.5);  
    glVertex3f(-0.5, 0.5, 0.5);  
    glVertex3f(-0.5, -0.5, 0.5);  
    glVertex3f(-0.5, -0.5, -0.5);  
    // 오른쪽 벽  
    glColor3f(1.0, 1.0, 1.0);  
    glVertex3f( 0.5, 0.5, -0.5);  
    glVertex3f( 0.5, 0.5, 0.5);  
}
```

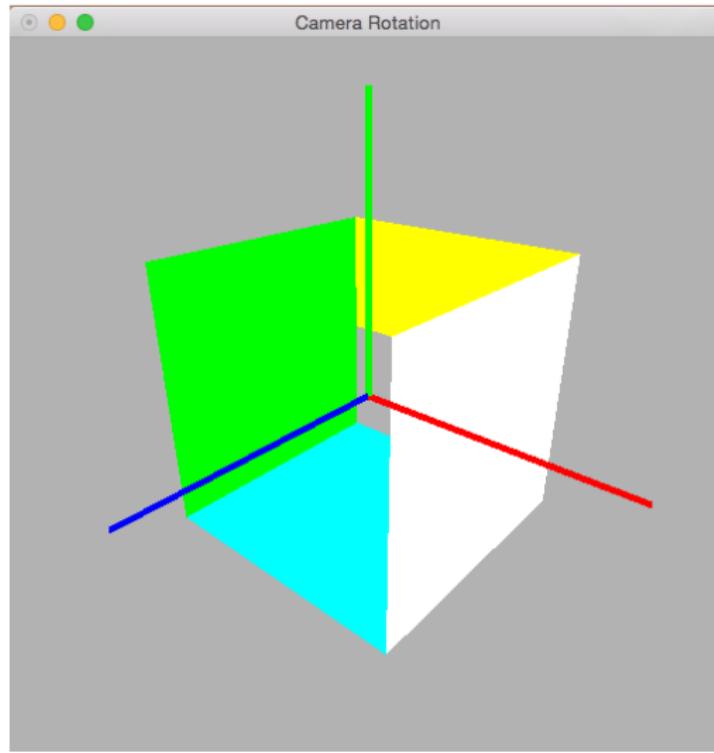
# (깊이버퍼 없이) 네 개의 면으로 상자 모양 그리기

Lines 26–50 / 50

```
glVertex3f( 0.5, -0.5,  0.5);
glVertex3f( 0.5, -0.5, -0.5);
glEnd();
}

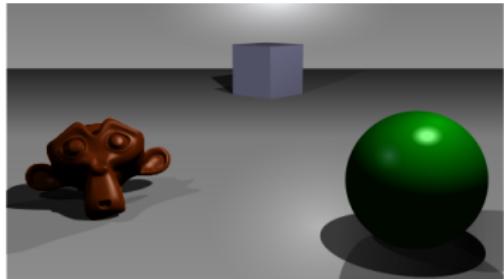
void draw() {
    drawScene();
    drawAxes(); // drawAxes 함수는 코드 ??에서 사용한 것을 그대로 쓴다.
}
```

(깊이버퍼 없이) 네 개의 면으로 상자 모양 그리기



# 깊이 버퍼 (depth buffer)

- 깊이 버퍼 혹은 z-버퍼
  - 색상 버퍼(color buffer)는 그려지는 영상의 각 화소별 색을 기록
  - 우리가 최종적으로 보게되는 것은 이 색상 버퍼의 내용
  - 색상 버퍼를 구성할 때 물체의 가려짐을 고려하여 가려진 물체를 그리지 않도록 하는 버퍼가 깊이 버퍼
  - 색상이 아니라 카메라에서 그려지는 픽셀까지의 거리를 저장하는 버퍼



A simple three-dimensional scene



Z-buffer representation

# 깊이 버퍼의 사용

- OpenGL에서는 이러한 깊이 버퍼를 쉽게 사용할 수 있도록 지원
  - `glutInitDisplayMode(GLUT_SINGLE | GLUT_DEPTH | GLUT_RGBA);`
- 기본적으로 “깊이 테스트” 작업을 수행하지 않는 것이 디폴트
- 깊이 테스트를 수행하기 위해서는 다음과 같이 상태를 변경
  - `glEnable(GL_DEPTH_TEST);`
- 또한 매번 그리기가 이뤄지면 그려진 내용에 따라 깊이 버퍼의 내용이 더러워져 있는 상태
- 새로운 그리기를 하기 전에 색상 버퍼를 깨끗하게 지우듯이 깊이 버퍼의 내용도 깨끗하게 지우는 다음과 같은 코드가 필요
  - `glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`

# 깊이 버퍼와 이중 버퍼 사용 예제

- 지금까지 색상과 깊이 버퍼가 하나만 존재하는 단일 버퍼 환경
- 단일 버퍼 환경은 애니메이션 등이 있을 때 깜빡임 발생 가능
- 해결: 앞 버퍼(front buffer)와 뒷 버퍼(back buffer)의 이중 구조
  - `glutInitDisplayMode(GLUT_DOUBLE|GLUT_DEPTH|GLUT_RGBA);`
- 디스플레이 장치로 프레임 버퍼를 보내는 것은 `glFlush`가 아니라 `glutSwapBuffers`를 이용
  - `glutSwapBuffers();`

# 깊이 버퍼와 이중 버퍼 사용 예제

Lines 1–25 / 50

[[헤더 파일 포함하기]]

```
void init( int argc , char **argv ) {
    // 윈도우 생성, 버퍼 설정
    glutInit(&argc , argv );
    // 이중 버퍼링, RGBA 색상 버퍼와 함께, 깊이 버퍼를 준비하도록 한다.
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA|GLUT_DEPTH) ;
    glutInitWindowPosition(0,0);
    glutInitWindowSize(512,512);
    glutCreateWindow("DEPTH BUFFER");
    glClearColor(0.7, 0.7, 0.7, 1.0);
    // 깊이 버퍼 검사를 활성화한다.
    glEnable(GL_DEPTH_TEST);
    // 카메라 투영 특성 설정 (glPerspective 사용). 이때는 GL_PROJECTION 행렬모드여야 한다.
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1.0, 0.1, 100.0);
}
void drawScene() { [[코드 ??와 동일]] }
void drawAxes() { [[코드 ??와 동일]] }
void draw() { [[코드 ??와 동일]] }
void display() {
    static float t=0.0;
    // 카메라의 위치와 방향을 설정한다. 이때는 GL_MODELVIEW 행렬 모드여야 한다.
    glMatrixMode(GL_MODELVIEW);
```

# 깊이 버퍼와 이중 버퍼 사용 예제

Lines 26–50 / 50

```
glLoadIdentity();
gluLookAt( 2.0*sin(t), 1, 2.0*cos(t), 0, 0, 0, 0, 1, 0);
t+=0.001;
// 컬러 버퍼 뿐만 아니라 깊이 버퍼도 매 프레임마다 지운다.
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
draw();
glutSwapBuffers();
};

};
```

# 깊이 버퍼와 이중 버퍼 사용 결과

