

□ 3D 그래픽스 프로그래밍

* Graphics (그래픽스) : 그래픽 학문

- * Modeling (모델링) : 모델을 만드는 것 / 만들고 옮기는 것 까지가 모델링
 - Mesh : 철사로 이루어진 것 (양파담는 망)
- * Animation (애니메이션) : 모델링을 움직인 것
 - keyFrame Animation : 중요한 프레임의 애니메이션
- * Rendering (렌더링) : 색을 칠하는 것, 빛을 어떻게 반사하는지 등을 표현하는 것이 렌더링

* 그래픽스 응용

- * 오프라인 렌더링 : 고퀄리티 그래픽(시간이 엄청 오래 걸린다)
- * 실시간 렌더링 : 게임 등에서 쓰이는 렌더링
- * 래스터 그래픽스 : 면 렌더링이 가능 (면의 색들을 한번에 ON)
- * 지역 조명 : 지역 내에만 빛을 사용
- * 전역 조명 : 전역으로 빛을 사용

□ 응용 함수

* display() 함수

- * glPointSize(5); : 점의 사이즈 조절 (glBegin() 전에 써줘야 함)

* main() 함수

- * glutIdle(display); : idle 상태를 설정할 수 있음

□ OpenGL 기초

OpenGL 기본형태

```
#include whatever you want

void display(display){
    glMatrixMode(GL_PROJECTION); // 카메라 렌즈 설정
    [[투영 행렬 설정:]]
    glMatrixMode(GL_MODELVIEW); //
    [[카메라의 위치와 방향 잡기:]]

    for(그려질 모든 객체에 대해){
        변환 설정;
        glBegin(그리기 프리미티브 지정);
        [[정점(vertex) 정보 제공:]]
        glEnd();
    }
    glFlush() 또는 glutSwapBuffers(); //메모리에 있는 것을 디스플레이로 전송
    // buffer : 메모리에 대해 임시저장하는 것
}

void main(int argc, char **argv){ // 콘솔 프로그램에서 입력을 받는 것
    [[윈도우 초기화:]] // GPU에 어떤 메모리를 잡을 것인지 설정
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("OpenGL Start");

    [[콜백 함수의 등록:]]
    glutDisplayFunc(display);

    [[메인 루프로 들어가기:]]
    glutMainLoop();
}
```

* OpenGL의 특징 : 운영체제 독립적

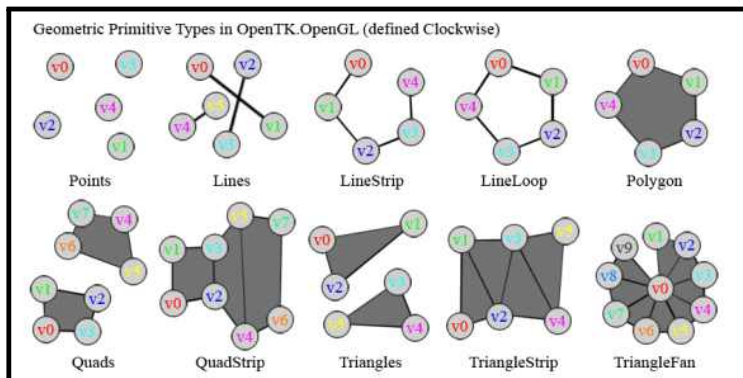
- * OS dependent : 운영체제 의존적, 종속적
- * OS independent : 운영체제 독립적 <= API GLUT lib가 운영체제 독립적으로 띄울 수 있게 해준다
- * window의 특징 : loop를 돌게 한다
- * event driven : 이벤트를 운전함 (call back)
 - Draw event : 그리는 이벤트 / Idle event : 아무것도 안할 때의 이벤트 / size event : 크기 조정

* 프리미티브(primitives/원시적인) : 크기 조정

- * 그래픽 하드웨어는 프로그래머가 지정한 프리미티브 설정에 따라 정점의 리스트를 처리
- * 프리미티브 사용 방법

| 프리미티브 |
|---|
| <pre>glBegin (drawing primitive); // vertex position, color, normal, etc glVertexInfo(); glEnd();</pre> |

- GL_POINT : 입력된 정점을 하나씩 점으로 가시화
- GL_LINES : 입력된 정점을 두 개씩 묶어 선분으로 표현 // 조합이 안되는 것은 버림
- 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_QUADS : 정점 네 개씩을 묶어 사각형 그리기
- GL_QUAD_STRIP : 처음 네 개 정점으로 사각형을 그리고, 이후 두 개씩 묶어 직전 두 개 정점과 함께 사각형 그리기
- GL_POLYGON : 입력된 모든 정점으로 다각형을 그림



* Matrix

- glPushMatrix(); ~ glPopMatrix(); : 한 단락으로 명령을 적용 (진행 시 사용)
- glBegin(GL_POLYGON); ~ glEnd(); : 범위 안의 모든 것에 적용시킨다. (생성 시 사용)

* 오브젝트 변환

- * glTranslatef(GLfloat dx, GLfloat dy, GLfloat dz); : dx, dy, dz 만큼 이동하는 함수
- * glScalef(GLfloat sx, GLfloat sy, GLfloat sz); : sx, sy, sz배 만큼 확대하는 함수
- * glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z); : 정해진 축을 중심으로 회전
 - angle은 라디안 각이 아니라 일반적으로 사용하는 60분법의 도
 - x축을 중심으로 회전할 때는 x를 1, y는 y를 1... 로 하면 됨
 - 2D를 할 때는 z만 1로 하면 됨

* 기타 Draw함수

- * `glutWireSphere(0.5, 10, 10);` : 스페어 (반지름, 경도, 위도), 선
- * `glutSolidSphere(0.5, 10, 10);` : 스페어 (반지름, 경도, 위도), 면
- * `glutWireTeapot(0.5, 10, 10);` : 주전자 (크기) 선으로 이루어짐
- * `glutSolidTeapot(0.5, 10, 10);` : 주전자 (크기) 면(색)으로 이루어짐
- * `glLineWidth(1);` : 선의 굵기를 설정

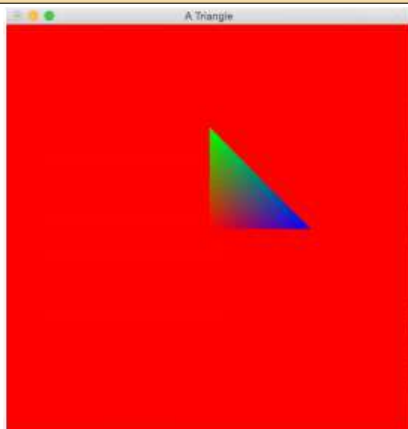
소스코드

```
#ifdef WIN32 // 윈도우 일 경우
#include <windows.h>
#include <Gl/gl.h>
#include <Gl/glut.h>
#else // 윈도우가 아닐 경우
#include <OpenGL/OpenGL.h>
#include <GLUT/GLUT.h>
#endif

void myDisplay() { // 오브젝트를 그림
    glClear(GL_COLOR_BUFFER_BIT); // 그림그리기 전에 사용
    glBegin(GL_POLYGON); // 폴리곤을 사용(입력된 모든 정점으로 다각형을 그림)
    glColor3f(1.0, 0.0, 0.0); // 점~면적 색 지정
    glVertex3f(0.0, 0.0, 0.0); // Vertex3f(세 점)의 값 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(); // 기본 색이 검은 색
}

int main (int argc, char * argv[]) { // 배경을 그림
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA); // glut의 RGBA 사용
    // 칼라와 깊이 칼라 버퍼 비트의 설정 하거나 0001 0010 방식으로 확인
    glutInitWindowPosition(0, 0); // glut 윈도우 위치 설정
    glutInitWindowSize(512, 512); // glut 창 사이즈 설정
    glutCreateWindow("12510096 조광민"); // 윈도우 이름 설정
    glClearColor(1.0, 0.0, 0.0, 1.0); // 지울 때 무슨 색으로 지울 것인지 설정
    glutDisplayFunc(myDisplay); // 디스플레이 콜백 등록
    glutMainLoop(); // 이벤트 루프로
    return 0;
}
```

실행화면

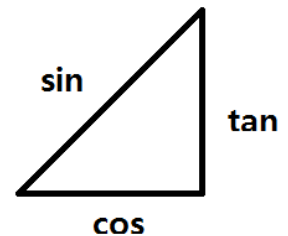


- * **법선벡터(nomal)** : 음영을 표시할 때 사용 // 면에 수직인 선 // $|a \cdot b| = N$ 벡터 a 와 b 를 외적
 - `glNormal3f()`: 법선 벡터는 무조건 3f
- * **smooth shading** : 삼각형의 점 3개의 법선벡터를 다 다르게 함 // 면과면이 만나는 점의 법선의 중간
- * **정점 데이터 설정 방법**
 - 정점 데이터는 위치와 법선벡터, 색 등을 표현
 - 정점의 위치만을 입력한다면 `glVertex[dim-type]`으로 입력
 - 3차원 정점의 각 성분을 부동소수점 표현으로 넣는다면 `glVertex3f(x, y, z)`로 입력

| 정점 데이터 |
|--|
| <pre>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();</pre> |

- * **프리미티브를 이용한 풍경 그리기**
 - **원 그리기** : 반지름이 1인 원 $\cos(\text{시타}) \sin(\text{시타})$

| 원 2개 그리기 |
|---|
| <pre>/* 원1 */ glBegin(GL_POLYGON); int nPoints=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 계산 while (angle < 3.14159*2.0) { glVertex2f(radius*cos(angle), radius*sin(angle)+0.75); angle += step; } /* 원2 */ //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();</pre> |



2주차 실습1

```
#define GLUT_DISABLE_ATEXIT_HACK

#include <Windows.h>
#include <gl/GL.h>
#include <gl/glut.h>
#include <math.h>

double rotation = 30;

void drawCircle(float setradius, float x, float y){
    glBegin(GL_POLYGON);
    int Points = 20;
    float radius = setradius;
    glColor3f(1.0, 1.0, 0.0);
    float angle = 0.0;
    float step = (3.14159*2.0) / Points;
    while (angle < 3.14159*2.0) {
        glVertex3f(radius*cos(angle) + x, radius*sin(angle) + y, 0);
        // cos, sin에 크기 비율을 곱해줌
        angle += step;
    }
    /*
    for(float x=0; x<6.28; x+0.001f){
        glVertex3f(cos(x)*radius, sin(x)*radius, 0);
    }
    */
    glEnd();
}

void drawRectangle(){
    glBegin(GL_QUADS);

    glColor3f(1, 0, 1);
    glVertex3f(-0.5, -0.3, 0);
    glVertex3f(0.5, -0.3, 0);
    glVertex3f(0.5, 0.3, 0);
    glVertex3f(-0.5, 0.3, 0);

    glColor3f(1, 1, 0);
    glVertex3f(0.4, 0.1, 0);
    glVertex3f(0.5, 0.1, 0);
    glVertex3f(0.9, 0.3, 0);
    glVertex3f(0.8, 0.3, 0);

    glEnd();
}

void myDisplay() {
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    glPushMatrix();
    drawRectangle();
    glPopMatrix();

    glPushMatrix();
    drawCircle(0.2, -0.3, -0.3);
    drawCircle(0.2, 0.3, -0.3);
    glPopMatrix();

    glFlush();
    glutSwapBuffers();
}

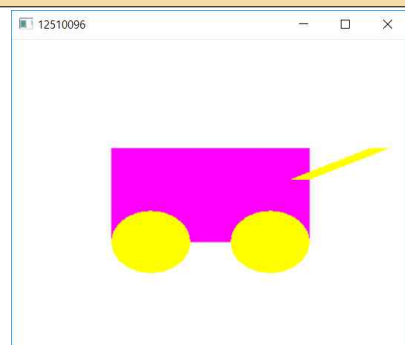
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("12510096");
    glutDisplayFunc(myDisplay);

    glClearColor(1, 1, 1, 1);

    glutMainLoop();
    return 1;

    return 0;
}
```

실행화면



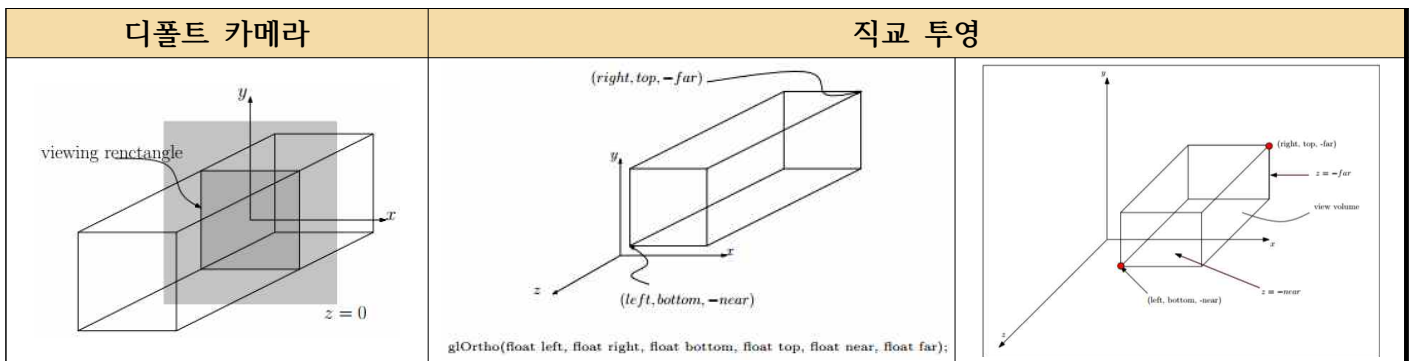
□ 카메라 설정

* 디폴트 카메라 (Default Camera)

- * OpenGL의 디폴트 카메라(직교 투영 카메라) : 원점 위치
- * z축 음의 방향

* 직교 투영 설정 (Orthographics Projection)

- * 원근 투영을 적용하지 않아 카메라에 잡히는 공간은 상자 형태
- * 상자 모양의 가시화 공간 내의 객체를 상자를 절단하는 면에 투영
- * `glOrtho(-1,1, -1,1, -1,1)` : 상자의 위치와 길이를 변경하는 함수



* 원근 투영(perspective projection) 설정

- * 실제 카메라나 우리 눈은 원근이 없는 평행 투영이 불가능
- * `glFrustum` 과 `gluPerspective`를 이용하여 원근 투영을 설정
- * `gluPerspective(float fovy, float Aspect, float near, float far);`
 - **fovy** : y축 방향으로의 시야각을 도(degree)로 나타낸 것 // 좀 인, 아웃 할 때 이 값을 변경
 - **Aspect** : 가시화 볼륨의 종횡비(aspect ratio) // 가로, 세로 비율) 0~1까지 : 가로비율, 1~ : 세로비율
 - **near** : 카메라에서 상이 맺히는 가까운 평면까지의 거리
 - **far** : 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 때만 불린다.

* 행렬 모드(matrix mode)

- * OpenGL 행렬의 종류 : 텍스처 행렬, 모델뷰 행렬, 투영행렬
 - **GL_MODELVIEW (모델뷰 행렬)** : 공간 내에서 가상 객체의 좌표를 변경 (물체의 위치) ,
 - **GL_PROJECTION (투영 행렬)** : 가상 객체를 투영면에 옮겨 놓음 // **GL_PROJECTION**
 - **GL_TEXTURE**
 - 광원(광각) 렌즈 : 원근감이 많다. (풍경)
 - 협각(망원) 렌즈 : 원근감이 없어진다. (인물)

| 행렬 모드 |
|--|
| <pre>glMatrixMode(GL_PROJECTION); // 렌즈를 설정함(렌즈속성만 바꾸는게 PROJECTION) glLoadIdentity(); // 항등행렬(1행렬)로 만듦 (아무런 변환이 없게 만듦) 초기화라고함. glOrtho(-2, 2, -2, 2, -2, 2); // -1, 1 일 때 보다 크기가 줄어든다. gluPerspective(60, 1.0, 0.1, 100.0); // 투영의 특성을 변경하는 것이므로 투영 행렬모드 glMatrixMode(GL_MODELVIEW); // 카메라의 위치를 바꿈 glLoadIdentity(); // 항등행렬(1행렬)로 만듦 (아무런 변환이 없게 만듦) gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); // 카메라의 위치를 옮기는 것이고, 이는 바꾸어 말해 물체의 위치를 카메라 기준에서 옮기는 것이므로 모델뷰 행렬을 변경 [[draw something]]</pre> |

* 카메라 위치, 회전 변경

* gluLookAt() : 카메라 위치 변경

- 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);
- **eye** : 보는 위치(카메라의 위치)
- **at** : 초점(바라볼 물체)의 위치
- **up** : 카메라를 돌리는 것 (상향 벡터)

* 카메라 이동의 반대로 물체를 옮겨 놓음

* 카메라 위치가 매 프레임마다 변경될 수 있으므로, 그리기 함수 내에 매번 불림

* gluRotatef() : 물체 회전 (축도 같이 회전함)

- gluRotatef(theta, 1, 1, 0);

행렬 모드

```
#include <stdio.h>
#include <stdlib.h>
#define GLUT_DISABLE_ATEXIT_HACK
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/glu.h>

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("12510096 조광민");
    glClearColor(1.0, 1.0, 1.0, 1.0);
    // 카메라 투영 특성 설정(glPerspective 사용), 이 때는 GL_PROJECTION 행렬모드여야 한다.
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1.0, 0.1, 100.0);
}

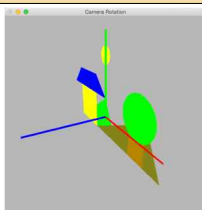
void drawScene() {
    [[앞서 사용한 코드 ??의 그리기 코드를 넣음. 단, glFlush는 여기서 사용하지 않음]]
}

void drawAxes() { // 3D를 구분하기 위해 3D 줄 사용
    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);    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. 이 때는 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;
    drawScene();
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(5);
    drawAxes();
    glLineWidth(1);
    glFlush();
};

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

실행 결과



네 개의 면으로 상자 모양 그리기

```
#include <stdio.h>
#include <stdlib.h>
#define GLUT_DISABLE_ATEXIT_HACK
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/glu.h>
```

```
void drawScene() { // drawing code
```

```
glBegin(GL_QUADS);
```

```
// 천장
```

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

```
glEnd();
```

```
}
```

```
void drawAxes(){ // 3차원 좌표 선을 생성
```

```
glBegin(GL_LINES);
```

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

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

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

```
glEnd();
```

```
}
```

```
void draw() {
```

```
drawScene();
```

```
drawAxes();
```

```
}
```

```
int main (int argc, char **argv){
```

```
/*init(argc, argv);
```

```
glutDisplayFunc(display);
```

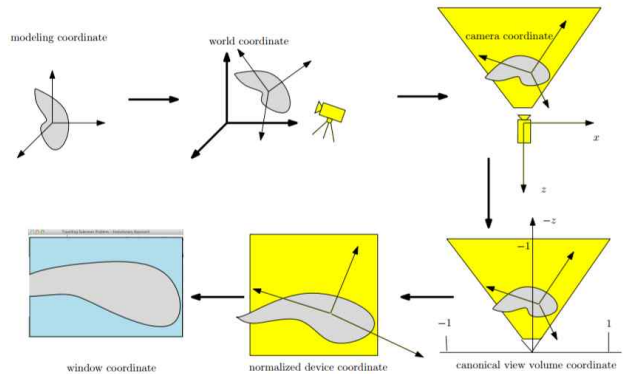
```
glutIdleFunc(display);
```

```
glutMainLoop();*/
```

```
}
```

카메라 좌표 1/2

- 3차원 그래픽스에서 모든 정점은 카메라를 기준으로 좌표가 재배치
- OpenGL에서 이 카메라 좌표계의 원점은 카메라의 위치가 되고, 카메라가 바라보는 방향이 z축의 음의 방향



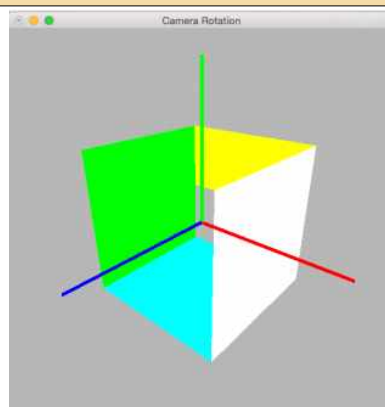
강영민 (동명대학교)

3D 그래픽스 프로그래밍

2015년 6학기

2 / 1

실행 화면



□ 상태(Function)

- * 메인함수의 func 기능
- * `glutDisplayFunc(display);` : 사용자가 기능을 수행하고 있는 상태일 때를 그려줌
- * `glutIdleFunc(display);` : 사용자가 아무것도 하지 않은 상태에서 그려줌
- * `glutReshapeFunc(reshape);` : 윈도우의 창 크기가 바뀌어도 모델의 크기는 바뀌지 않음
 - `glViewport(0, 0, w, h);` : 윈도우 창의 비율을 바꾼 상태의 비율을 맞춰줌

reshape

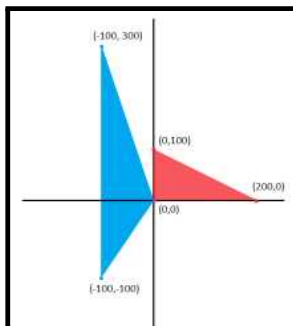
```
void reshape(int w, int h){
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    float asp = float(w)/h;
    glOrtho(-asp*2,asp*2,-1*2,1*2,-2,2);
    glViewport(0, 0, w, h);
}
```

- * `glutKeyboardFunc(keyboard);` : 윈도우의 창 크기가 바뀌어도 모델의 크기는 바뀌지 않음
 - `glPostRedisplay();` : 윈도우를 그려야 할 때만 그려줌 (`glutIdleFunc`를 안써도됨)

6주차 (16.10.07)

□ 변환

- * **어파인(affine) 변환** : 직선은 직선으로, 평행선은 평행선으로 유지
 - 이동(translate) : 주어진 변위 벡터만큼 좌표를 동일하게 옮겨 놓는다
 - 회전(rotate) : 2차원에서는 기준점, 3차원에서는 기준축을 중심으로 주어진 각도만큼 돌아간다.
 - 크기변경(scale) : 각 축 방향으로 주어진 비율에 따라 좌표 값이 커지거나 줄어든다.



- * **동차 좌표계(Homogeneous coordinate)** : 이동과 회전 모두 4x4 행렬의 곱으로 표현 가능
 - n차원의 사영공간을 n+1 차원의 좌표로 나타내는 좌표계
 - 무한의 위치에 있는 점을 유한 좌표로 표현하는 데 적합
 - $(x, y, z) \Rightarrow (kx, ky, kz, k)$ // k가 1일 때, $(x, y, z, 1)$ // k가 2일 때, $(2x, 2y, 2z, 2)$
 - 사영기하학에서 사용하며, 4차원 좌표에서 3차원으로 한 점을 바라보는 떨어지는 점이기에 때문에 $(1,1,1,1)$ 과 $(2,2,2,2)$, $(3,3,3,3)$ 는 같은 점이 된다. // $(x, y, z, 0)$ 는 무한

* 동차좌표계를 사용하는 이유

- 3차원 데카르트 좌표를 사용할 경우 이동은 벡터의 덧셈으로 표현되고, 회전은 3 x 3 행렬의 곱으로 표현
- 이동과 회전이 누적되면 벡터 덧셈과 행렬 곱셈이 연속적 적용됨
- 동차좌표를 사용하면 이동과 회전 모두 4 x 4 행렬의 곱으로 표현 가능
- 누적된 이동, 회전 변환을 하나의 행렬로 표현 가능

- 동차좌표계를 사용한 이동 행렬곱 :
$$\begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

* CTM (Current Transform Matrix)

- 처음 카메라가 만들어진 것 : I
 - 카메라를 이동하여 그려줌 : C (glLookat(x,y,z eye, x,y,z at, x,y,z up))
 - 물체 이동 : T
 - 물체 회전 : R
 - ex) glTranslatef(1, 0, 0)일 경우 : I*T(1,0,0)
 - ex) I(CTM) * R(90도) * T(1,0,0) 일 경우/ y방향의 1위치에 있는 위로보는 주전자가 된다.
- * 물체가 Rotatef() 되면 축자체도 전부 회전한다.

* gluRotatef() : 물체 회전 (축도 같이 회전함)

- gluRotatef(theta(각도), 1, 1, 0);

glRotatef() 실습 코드 - 1

```
#define GLUT_DISABLE_ATEXIT_HACK

#include <Windows.h>
#include <gl/GL.h>
#include <gl/glut.h>
#include <math.h>

float range = 1.0;
float aspRatio = 1.0;
float dX = 0.0;

void drawAxis(){
    glBegin(GL_LINES);

    glColor3f(1, 0, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(1, 0, 0);

    glColor3f(0, 1, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 1, 0);

    glColor3f(0, 0, 1);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 0, 1);

    glEnd();
}

void myDisplay() {
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(2, 2, 2, 0, 0, 0, 0, 1, 0);

    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    glRotated(angle, ax, ay, 0.0);

    glLineWidth(5);
    drawAxis();
    glLineWidth(1);

    glColor3f(1, 0, 0);
    glutWireTeapot(0.5); // 스페어 (반지름, 경도, 위도)

    glColor3f(0, 1, 0);
    glTranslatef(1, 0, 0);
    glRotatef(90, 0, 0, 1);
    glutWireTeapot(0.5); // 스페어 (반지름, 경도, 위도)

    glColor3f(1, 1, 1);
    glTranslatef(1, 0, 0);
    glRotatef(90, 0, 0, 1);
    glutWireTeapot(0.5); // 스페어 (반지름, 경도, 위도)

    glColor3f(0, 0, 1);
    glTranslatef(1, 0, 0);
    glRotatef(180, 0, 0, 1);
    glutWireTeapot(0.5); // 스페어 (반지름, 경도, 위도)

    glutSwapBuffers();
}
```

glRotatef() 실습 코드 - 2

```
void SetCamera() { // 렌즈를 설정
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, aspRatio, 0.1, 1000);
}

void reshape(int w, int h){
    aspRatio = float(w)/h;
    SetCamera();
    glViewport(0, 0, w, h);
}

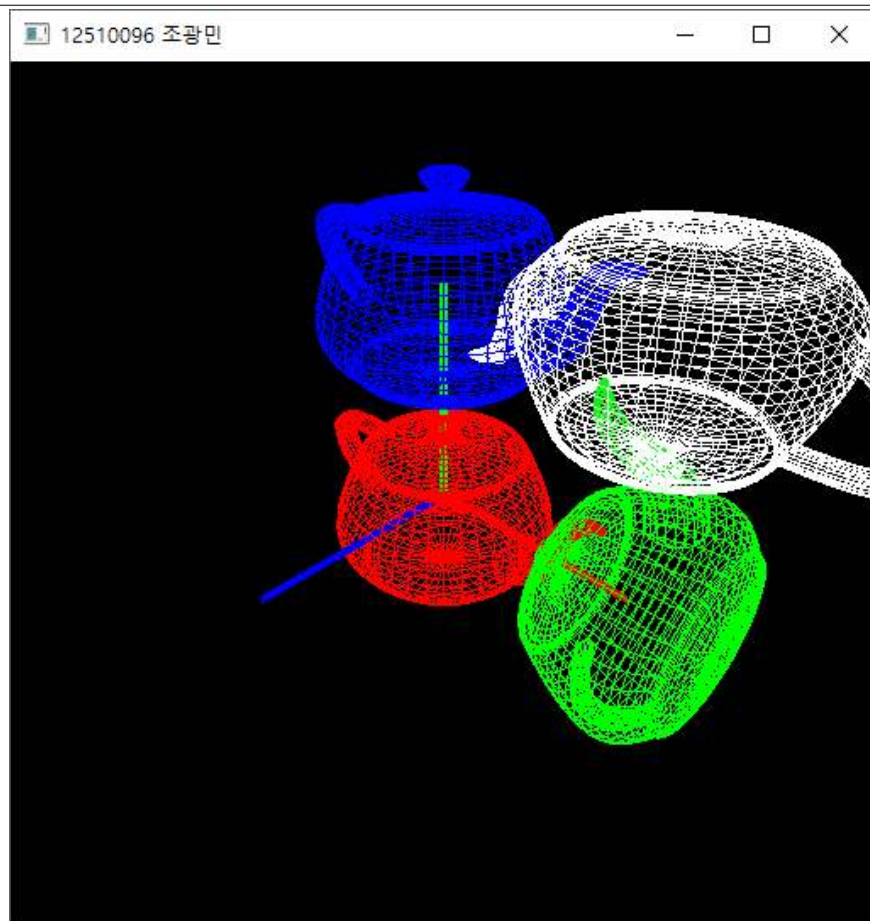
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(512, 512);
    glutCreateWindow("12510096 조광썸민이");
    glEnable(GL_DEPTH_TEST);

    glClearColor(0.0, 0.0, 0.0, 1.0);

    glutDisplayFunc(myDisplay);
    glutIdleFunc(myDisplay);
    glutReshapeFunc(reshape);

    glutMainLoop();

    return 0;
}
```



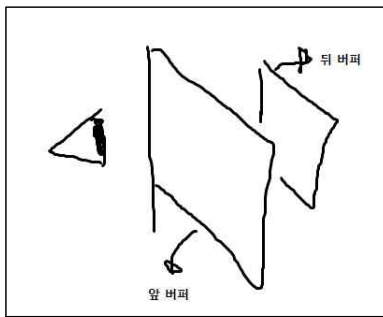
□ 깊이 버퍼와 이중 버퍼

* 깊이 버퍼 사용

- * `glutInitDisplayMode(GLUT_SINGLE | GLUT_DEPTH | GLUT_RGBA)` : 깊이 버퍼를 쉽게 사용할 수 있도록 지원
 - GLUT_DEPTH : 깊이 테스트 작업을 수행하지 않는 것이 디폴트
- * `glEnable(GLUT_DEPTH_TEST)` : 깊이 테스트를 수행 (앞, 뒤에 그려질 것을 설정해줌)
- * `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
 - : 새로운 그리기 전 색상 버퍼를 깨끗이 지우듯, 깊이 버퍼의 내용도 깨끗하게 지움
- * **단계** : main의 `glutInitDisplayMode`에 GLUT_DEPTH 버퍼 선언
 - main에 `glEnable(GL_DEPTH_TEST);` 로 깊이를 존재하게 함
 - Display에 GL_DEPTH_BUFFER_BIT 로 윈도우를 지워줌

* 깊이 버퍼와 이중 버퍼 사용

- * 단일 버퍼 환경은 애니메이션 등이 있을 때 깜빡임 발생
- * `glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA);`
 - : 앞 버퍼(front buffer)와 뒷 버퍼(back buffer)의 이중 구조
- * `glutSwapBuffers();` : 디스플레이 장치로 프레임 버퍼를 보내는 것은 `glFlush`가 아니라 `glutSwapBuffer`를 이용
 - 앞쪽 버퍼는 손을 대지 않고 뒤쪽 버퍼에만 손을 대기 때문에 자연스럽게 애니메이션을 보여줌



네 개의 면으로 상자 모양 그리기

```
#include <stdio.h>
#include <stdlib.h>
#define GLUT_DISABLE_ATEXIT_HACK
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/glu.h>

void init(int argc, char **argv) { // 윈도우 생성, 버퍼 설정
    glutInit(&argc, argv);

    // 이중 버퍼링, RGBA 색상 버퍼와 함께, 깊이 버퍼를 준비하도록 함
    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 활성화
    glEnable(GL_DEPTH_TEST);

    // 카메라 투영 특성 설정 (glPerspective 사용), 이 때는 GL_PROJECTION 행렬 모드여야 함
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1.0, 0.1, 100.0);
}

void drawScene(){

}

void drawAxes(){ // 3차원 좌표 선을 생성
    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);    glVertex3f(0.0, 0.0, 0.0);    glVertex3f(0.0, 0.0, 1.0);
    glEnd();
}

void draw() {
}
```

glRotatef() 실습 코드2 - 1 (행성 자전, 공전)

```
#define GLUT_DISABLE_ATEXIT_HACK

#include <Windows.h>
#include <gl/GL.h>
#include <gl/glut.h>
#include <math.h>

float range = 1.0;
float aspRatio = 1.0;
float dX = 0.0;
float rt = 0, grt=0;

void drawAxis(){
    glBegin(GL_LINES);

    glColor3f(1, 0, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(1, 0, 0);

    glColor3f(0, 1, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 1, 0);

    glColor3f(0, 0, 1);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 0, 1);

    glEnd();
}

void myDisplay() {
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(2, 2, 2, 0, 0, 0, 0, 1, 0);

    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    glLineWidth(5);
    drawAxis();
    glLineWidth(1);

    glPushMatrix();
    glColor3f(1, 0, 0);
    glRotatef(rt, 0, 1, 0);
    glutWireSphere(0.6, 20, 20); // 스핀-페췌어ú (반öY지o름A×, 경튜i도伊i, 위§도伊i)
    glPopMatrix();

    rt += 1;
    grt += 2;
    if (rt >= 360){
        rt = 0;
    }
    if (grt >= 360){
        grt = 0;
    }

    // 수ò성彼
    glPushMatrix();
    glColor3f(0, 0, 1);
    glRotatef(rt, 0, 1, 0);
    glTranslatef(1.2, 0, 0);
    glRotatef(rt, 1, 0, 0);
    glutWireSphere(0.2, 10, 10); // 스핀-페췌어ú (반öY지o름A×, 경튜i도伊i, 위§도伊i)

    glRotatef(-rt, 1, 0, 0);
    glColor3f(1, 1, 1);
    glRotatef(rt, 0, 1, 0);
    glTranslatef(0.5, 0, 0);
    glRotatef(rt, 1, 0, 0);
    glutWireSphere(0.1, 10, 10); // 스핀-페췌어ú (반öY지o름A×, 경튜i도伊i, 위§도伊i)
    glPopMatrix();

    // 금뿔성彼
    glPushMatrix();
    glColor3f(1, 1, 0);
    glRotatef(grt, 0, 1, 0);
    glTranslatef(2, 0, 0);
    glRotatef(grt*2, 1, 0, 0);
    glutWireSphere(0.4, 10, 10); // 스핀-페췌어ú (반öY지o름A×, 경튜i도伊i, 위§도伊i)

    glRotatef(-grt, 1, 0, 0);
    glColor3f(1, 1, 1);
    glRotatef(grt, 0, 1, 0);
    glTranslatef(0.5, 0, 0);
    glRotatef(grt, 0, 1, 0);
    glRotatef(grt, 0, 0, 1);
    glutWireSphere(0.1, 10, 10); // 스핀-페췌어ú (반öY지o름A×, 경튜i도伊i, 위§도伊i)
    glPopMatrix();

    glutSwapBuffers();
}
```

glRotatef() 실습 코드2 - 2 (행성 자전, 공전)

```
void SetCamera() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //glOrtho(-aspRatio * range + dX, aspRatio * range + dX, -range, range, -2, 2);
    gluPerspective(60, aspRatio, 0.1, 1000);
}

void reshape(int w, int h){
    aspRatio = float(w)/h;
    SetCamera();
    glViewport(0, 0, w, h);
}

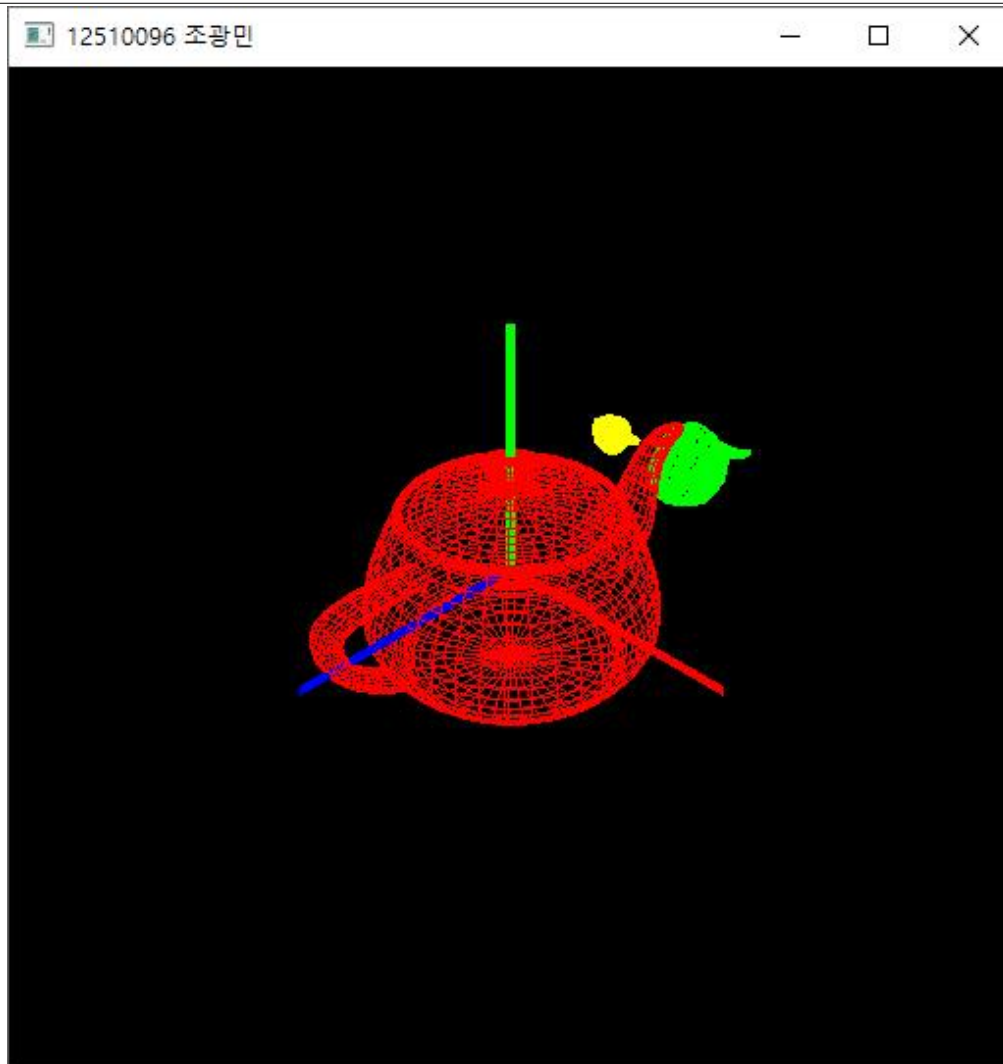
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(512, 512);
    glutCreateWindow("12510096 조광썸민호");
    glEnable(GL_DEPTH_TEST);

    glClearColor(0.0, 0.0, 0.0, 1.0);

    glutDisplayFunc(myDisplay);
    glutIdleFunc(myDisplay);
    glutReshapeFunc(reshape);

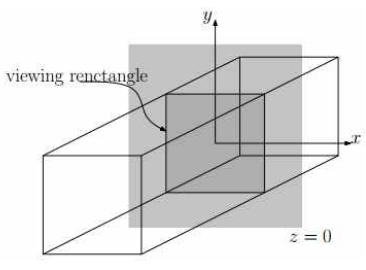
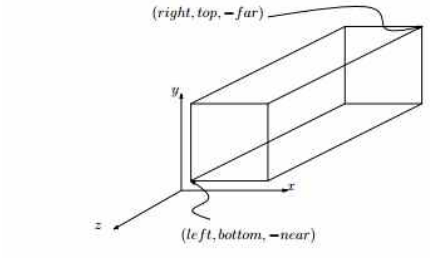
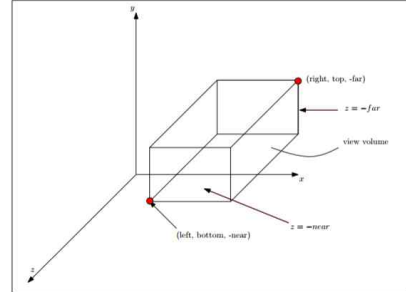
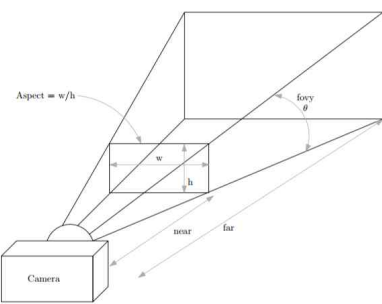
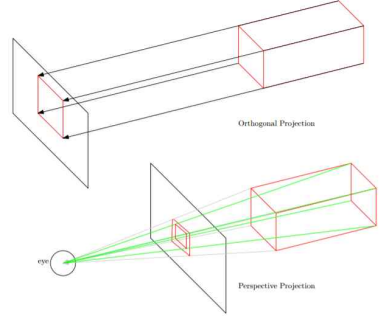
    glutMainLoop();

    return 0;
}
```

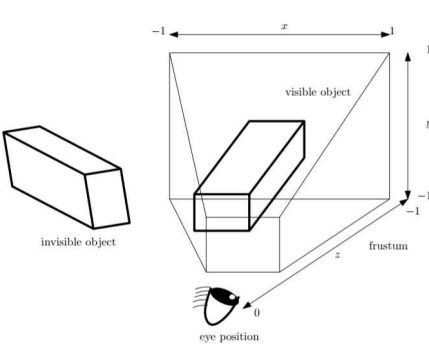
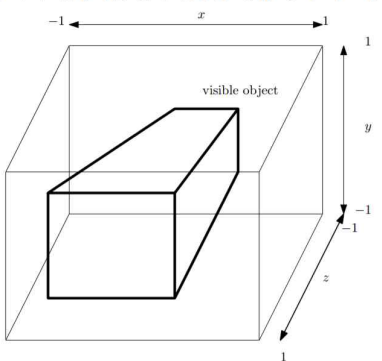
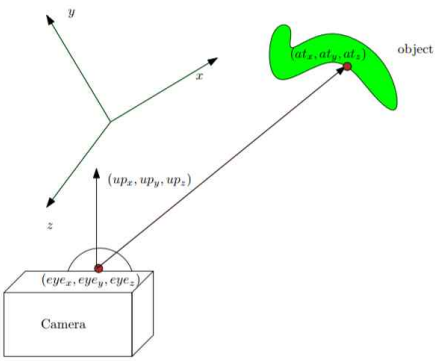
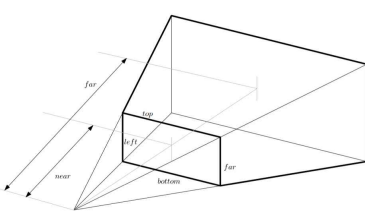


□ 직교 / 원근 투영 카메라 요약 그림

- * 디폴트 카메라 : 중심이 원점, 각 변의 길이가 2인 상자모양
- * 직교 투영 카메라 (glOrtho) : 디폴트 카메라의 위치와 길이를 변경
- * 원근 투영 카메라 (glFrustum, gluPerspective) : 원근 투영 설정

| 디폴트 카메라 | 직교 투영 카메라 | |
|---|--|---|
|  |  |  |
| | <code>glOrtho(float left, float right, float bottom, float top, float near, float far);</code> | |
| 원근 투영 카메라 | 직교 투영과 원근 투영 비교 | |
|  |  | |

- * 절두체 공간 : 3차원 그래픽스의 카메라 모델은 렌더링 대상이 되는 영역을 일정한 범위로 제한
- * 정규 장치 좌표계 : 화면에 출력을 하기 위해서는 관측 볼륨을 정규 장치 좌표계로 바꾼다.
- * 카메라 위치 변경 (gluLookAt) : 카메라를 원하는 곳으로 옮겨줌
// gluPerspective를 사용하여도 카메라는 여전히 원점(0, 0, 0)에 놓임
- * glFrustum(float left, float right, float bottom, float top, float near, float far);

| 절두체 공간 | 정규 장치 좌표계 | 카메라의 위치 변경 |
|---|---|---|
|  |  |  |
| glFrustum | glFrustum과 gluPerspective | |
|  | <pre>gluPerspective(fovy, aspect, near, far); glFrustum(left, right, bottom, top, near, far);</pre> | |
| | $top = near \cdot \tan \frac{\theta}{2}$ $bottom = -near \cdot \tan \frac{\theta}{2}$ $right = top \cdot aspect$ $left = bottom \cdot aspect$ | |

□ display() 함수 추가 코드

- glBegin(프리티미브) ~ glEnd() : 그림을 그리는 것
- glFushMatrix() : 이전 변환행렬을 저장 (push ~ pop 내부에 사용한 함수는 push~pop 내부에서만 적용)
- glPopMatrix() : Push로 저장된 행렬 값을 가져옴
- glRotate*(), glTranslate*(), glScale*() 등의 함수의 기능을 활성화하기 위해 위의 두 함수를 지정

display() 함수 추가 코드

```
// global variables
GLfloat xrot, yrot, zrot;
GLfloat red=1.0, green=1.0, blue=1.0, alpha=1.0;

void myDisplay() {
    char info[128];

    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    /* Translate & rotate the image */
    glPushMatrix();
    glTranslatef(1.0, 1.0, 0.0); // move rotation axis to triangle center
    glRotatef(xrot, 1.0f, 0.0f, 0.0f); // rotate the image
    glRotatef(yrot, 1.0f, 0.0f, 0.0f); // rotate the image
    glRotatef(zrot, 1.0f, 0.0f, 0.0f); // rotate the image
    glTranslatef(-1.0, -1.0, 0.0); // restore axis origin

    /* Draw triangle */
    glColor4f(red, green, blue, alpha); // color set as RGBA

    glBegin(GL_TRIANGLES);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(2.0f, 1.0f, 0.0f);
    glVertex3f(1.0f, 2.0f, 0.0f);
    glEnd();

    glPopMatrix(); // restore the coord. matrix

    sprintf(info, "x=%.1f, y=%.1f, z=%.1f, mag=%.2f", xrot, yrot, zrot, magfac);
    glutSetWindowTitle(info);

    glFlush();
}
```

* display() 함수 설명

- glutSetWindowTitle() : 'g' key를 누르고, PgUp key로 삼각형을 회전시키고, Window Title창에 x,y,z축에 대한 회전각이 나타남
- 방향키 사용으로 회전, +, - 로 이미지의 크기를 확대 및 축소
- glRotate*(angle, x, y, z) : 좌표계의 축에 따라 주어진 각만큼 이미지를 회전시킴
- glutGetModifiers() : callback 함수에서 SHIFT, CTRL, ALT를 동시에 눌러 이벤트를 Handle할 경우 GLUT_ACTIVE_SHIFT, GLUT_ACTIVE_CTRL, GLUT_ACTIVE_ALT mode를 연계해서 사용, Alt+r, Alt+b를 이용하여 색상 변화
- 3개의 키 사용 코드

Special key 3개의 키 사용

```
mod = glutGetModifiers();
if (mod &&(GLUT_ACTIVE_CTRL | GLUT_ACTIVE_ALT)) {
```


* Advanced Keyboard 기능

- 화살표 같은 key를 계속 눌러 자동 반복 실행을 할 때, delay현상의 문제점을 해결해주는 함수

glutSetRepeat (전역 기반으로 작용)

```
int glutSetRepeat( int repeatMode );
```

Parameters:

- GLUT_KEY_REPEAT_OFF - 자동반복 Mode 기능 비활성
- GLUT_KEY_REPEAT_ON - 자동반복 Mode 기능 활성화
- GLUT_KEY_REPEAT_DEFAULT - default 상태로 자동반복 Mode 를 Reset

- Application으로부터 하나가 아닌 모든 Window의 반복 기능에 영향을 준다. 따라서 이 함수를 자동 mode의 비활성화로 사용할 때는 Application을 끝내기 전 default 상태로 복원하는 것이 편리

glutIgnoreKeyRepeat

```
int glutIgnoreKeyRepeat( int repeatMode );
```

Parameters:

- repeatMode - 0 이면 자동반복 mode 활성화, 0 이 아니면 자동반복 mode 비활성

- key 반복이 일어날 때의 callback 받는 것을 비활성화 시켜, 다른 Application에 영향을 주지 않고 안전하게 key 누름을 무시해야 할 때 사용

키 입력

```
void glutKeyboardUpFunc( void (*func)(unsigned char key,int x,int y) );
```

```
void glutSpecialUpFunc( void (*func)(int key,int x, int y) );
```

Parameters:

- func - callback 함수 이름

- key 반복이 일어날 때 callback 받는 것을 멈출 것인데, 만약 key를 누르고 있는 동안에만 어떤 Action이 실행되기를 원한다면, 그 key가 누르기를 해지하는 때를 알아야함.
- GLUT는 key가 해지되었을 때를 위한 두 개의 register callback 기능을 제공함

□ OPENGL API □

□ **Keyboard Event** : 키보드 이벤트는 2개의 callback 함수가 있다.

* glutKeyboardFunc(keyboard); // 일반 키보드처리 (main에 작성)

- void keyboard(unsigned char key, int x, int y){ switch(){ } glutPostRedisplay(); }

* glutKeyboardUpFunc(keyboard); // 키보드를 뗐을 때 이벤트 발생

- void keyboard(unsigned char key, int x, int y){ switch(){ } glutPostRedisplay(); }

* glutSpecialFunc(special); // 특수 키보드 처리 (main에 작성)

- void special(int key, int x, int y){ switch(){ } glutPostRedisplay(); }

* glutSpecialUpFunc(special); // 특수 키보드 처리 (main에 작성)

- void special(int key, int x, int y){ switch(){ } glutPostRedisplay(); }

* glutPostRedisplay(); : 윈도우를 그려야 할 때만 그려줌 (glutIdleFunc()를 안 써도 됨)

* keyboard()의 switch (key) { case : }에 사용할 입력 문자

- 'a' ~ 'z' : 일반 키 입력

- GLUT_ACTIVE_ALT, GLUT_ACTIVE_CTRL : 특수 키를 누른 상태

* special()의 switch (key) case : 에 사용할 입력 문자

- GLUT_KEY_F1 : GLUT_KEY_특수키 입력

□ **Mouse Envent** : 마우스 이벤트는 3개의 callback 함수가 있다.

- * **glutMouseFunc(mouse);** // 마우스 클릭 시 발생하는 이벤트
 - **void mouse(int button, int direction, int x, int y){ switch(){ } glutPostRedisplay(); }**
- * **glutMotionFunc(motion);** // 마우스 클릭 후 이동 시 발생하는 이벤트
 - **void motion(int x, int y){ 예제 참조 }**
- * **glutPassiveMotionFunc(mouse);** // 마우스 클릭을 하지 않고 이동 시 발생하는 이벤트
- * **glutMouseWheelFunc(mousewheel);** // 마우스 휠을 사용한 이벤트
 - **void mousewheel(int wheel, int direction, int x, int y){ 예제 참조 }**
- * **mouse()의 switch (key) { case : }**에 사용할 입력 문자
 - **GLUT_DOWN** : 마우스를 클릭 시

□ **Keyboard Envent 예제**

keyboard() 함수 - 일반 워드

```
void keyboard(unsigned char key, int x, int y)
{
    int mod;

    switch (key) {
        case 'r':
            red = 1.0f; green = 0.0f; blue = 0.0f;
            mod = glutGetModifiers();
            if (mod && GLUT_ACTIVE_ALT) { // 알트키를 누른상태로 키 입력
                red = 0.5f;
            }
            break;
        case 'g':
            red = 0.0f; green = 1.0f; blue = 0.0f;
            mod = glutGetModifiers();
            if (mod && GLUT_ACTIVE_ALT) {
                green = 0.5f;
            }
            break;
        case 'b':
            red = 0.0f; green = 0.0f; blue = 1.0f;
            mod = glutGetModifiers();
            if (mod && GLUT_ACTIVE_ALT) {
                blue = 0.5f;
            }
            break;
        case 27:
            exit(0);
            break;
        default:
            red = 1.0f; green = 1.0f; blue = 1.0f; alpha = 1.0f;
            break;
    }
    glutPostRedisplay();
}
```

special() 함수 - 특수 키

```
void special(int key, int x, int y)
{
    switch (key) {
        // select image view mode when reshape the window
        case GLUT_KEY_F1: // both object shape & size is not changed
            viewmode = 1; // Ortho view mode
            break;
        case GLUT_KEY_F2: // both object shape & size is chanded
            viewmode = 2; // Ortho view mode
            break;

        // spin key for image rotation
        case GLUT_KEY_UP:
            xrot -= 2.0f;
            if (xrot < -360.0f) xrot += 360.0f;
            break;
        case GLUT_KEY_DOWN:
            xrot += 2.0f;
            if (xrot > +360.0f) xrot -= 360.0f;
            break;
        case GLUT_KEY_PAGE_DOWN:
            zrot -= 2.0f;
            if (zrot < -360.0f) zrot += 360.0f;
            break;
        case GLUT_KEY_PAGE_UP:
            zrot += 2.0f;
            if (zrot > +360.0f) zrot -= 360.0f;
            break;
        case '+':
            magfac += 0.02;
            break;
        case '-':
            magfac -= 0.02;
            break;

        case GLUT_KEY_F10:
            glutFullScreen();
            break;
        case GLUT_KEY_F9:
            glutReshapeWindow(wwidth, wheight);
            glutPositionWindow(100, 100);
            break;
        default:
            break;
    }
    glutPostRedisplay();
}
```

□ Mouse Envent 예제

mouse() 함수 - 마우스 클릭

```
void mouse(int button, int direction, int x, int y)
{
    switch (direction){
        case GLUT_DOWN:
            // 마우스 버튼을 누른 위치를 기록
            cx = x;
            cy = y;
            // 표시하고 있는 물체의 회전 각을 기록
            ca = angle;
            break;
        default:
            break;
    }
}
```

motion() 함수 - 마우스 클릭 후 이동 시

```
void motion(int x, int y){
    double dx, dy, a;

    // 마우스 포인터 위치의 끝기 시작 위치에서의 변위
    dx = (x - cx) * sx;
    dy = (y - cy) * sy;

    // 마우스 포인터 위치의 끝기 시작 위치에서의 거리
    a = sqrt(dx * dx + dy * dy);

    if (a != 0.0){
        // 거리를 각도로 환산하여 드래그 시작시의 회전 각에 가산
        angle = fmod(ca + SCALE * a, 360.0);

        // 마우스 포인터의 변위에서 회전축 벡터를 요청
        ax = dy / a;
        ay = dx / a;
        az = 0.0;

        // 도형의 재 묘화 (윈도우를 그려야 할 때만 그려줌)
        glutPostRedisplay();
    }
}
```

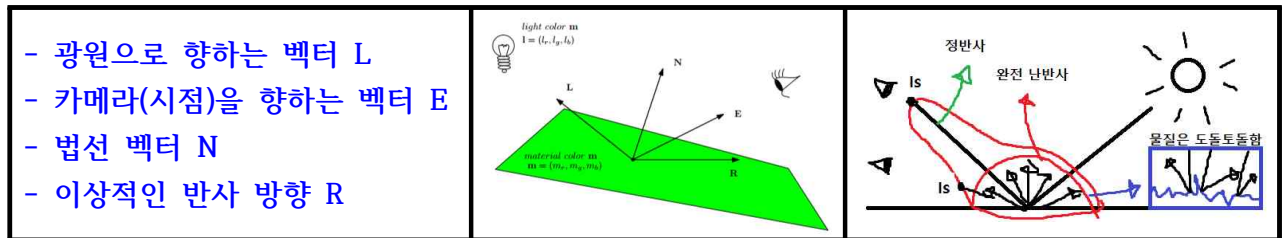
mousewheel() 함수 - 마우스 휠

```
void mousewheel( int wheel, int direction, int x, int y)
{
    switch (direction){
        case GLUT_ :
            //
            cx = x;

            break;
        default:
            break;
    }
}
```

□ 조명

○ 풍 모델(셰이딩) : 정반사와 난반사, 주변광 반사 특성을 표현할 수 있는 간단하고 빠른 모델

* 수정된 풍 모델 : 반사벡터 대신에 반 벡터 H (half vector) 를 도입

- R 을 계산해야 정반사를 계산, 각 거리를 봄, R 계산하는데 벡터 계산을 많이 해야 해서, N 과 E 의 중간벡터 (반 벡터 H)를 구해서 N 과 일치하면 E 와 R 이 줄어들면 일치하면 H 와 N 이 일치한다고 가정한다.

// $R \cdot E$ 가 일치하는지 보는 것과 $N \cdot H$ 가 일치하는지 보는 것이 유사할 것이라 보고 계산하기 더 쉬운 $N \cdot H$ 를 본다.* 광강도 : 난반사 I_d 와 정반사 I_s (빛의 세기)

- 눈에 감지되는 빛의 색상을 결정 (음영을 광강도에 의해 결정)
- 광강도의 값은 스칼라(scalar) 값으로 I 로 표현
- 실제 눈에 보이는 색 = 광원과 재질에 의해 결정되는 색상 c * 광강도 I // $k = I c$

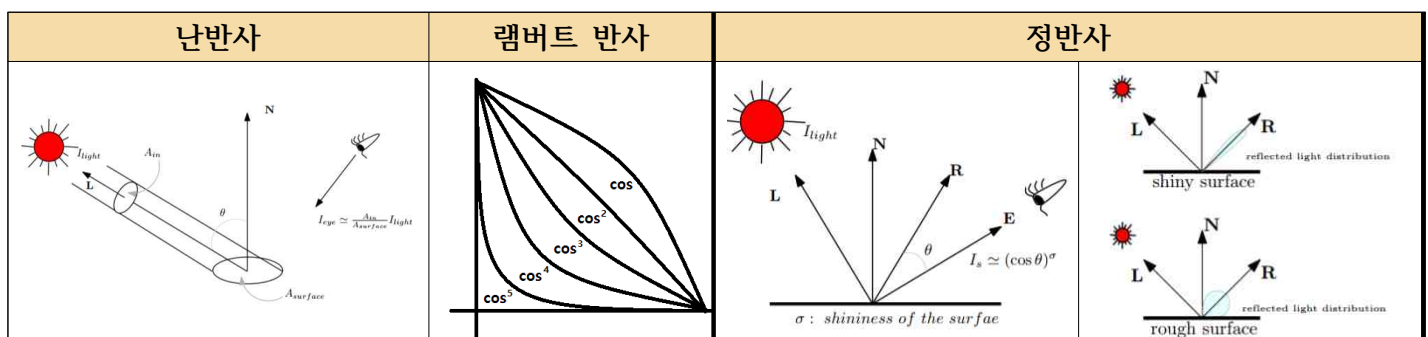
* 난반사 : 눈의 위치에 상관없는 빛

- N 과 L 을 내적해서 구함
- 눈이 어디에 있는지 동일한 색상 관찰
- 눈의 움직임에 따라 변하는 하이라이트는 표현 못함
- 색을 칠하려고 하는 한 지점에 대해 어디서 쳐다보든 동일한 밝기
- 밝기는 얼마나 많은 에너지가 해당 지점에 떨어지는지에 달려 있음
- 광원 벡터 L 과 법선 벡터 N 이 일치할 때 최대, 90도를 이룰 때 0이 됨
- 램버트 반사 모델 : 두 벡터 사이각의 코사인에 비례 // \cos 의 제곱에 따라 밝고 어두움을 설정
- 광강도를 계산해야하는 지점에서 빛을 향하는 방향벡터 L 과 표면 법선벡터 N 의 내적으로 I_d 를 구함
-> $I_d = \cos \theta = L \cdot N$

* 정반사 : 거울과 같이 입사각에 대칭되는 방향으로 반사

- 거울과 같은 반사가 아니라 반사 벡터 R 중심으로 퍼지는 반사를 표현
- 반사 벡터 R 근처에서 강하게 관찰되기 때문에 눈을 R 근처로 가져가야 강한 반사가 일어남
- 정반사의 광강도 I_s 는 R 과 E 의 사이각의 코사인에 연관
- 물체의 재질에 따라 R 방향으로 집중되는 정도가 달라짐 - 물체의 반질함(shininess) σ 에 의해 결정
-> $I_d = \cos \theta = (R \cdot E)\sigma$

* 주변광 : 모든 곳에 색을 다 칠해 주는 것



* 라이트를 사용하면 glColor3f를 듣지 않음 / Color를 사용하는 방법

- 첫 번째 방법 : draw할 때는 라이트를 꺼줌
- 두 번째 방법 : material에 색을 추가해줌

* 용어 설명

- * **Material** : 어떤 메시가 빛을 받는다고 할 때, 마테리얼에서 해당 광원으로부터 빛을 받으면 어떻게 발산할 것인지를 설정해준다. // 물체가 빛을 받으면 그 빛을 반사하고 반사한 빛이 우리 눈에 인지된다. 그리고 반사되는 정도는 물체 마다 다 다르다 (ex) 고무, 거울, 나무 등) 3D에서는 이런 정도를 Ambient, Diffuse, Specular 라는 속성으로 이 재질을 표현한다.
- * **Diffuse** : 표면이 반사하는 난반사광
- * **Ambient** : 표면이 반사하는 환경광
- * **Specular** : 표면이 반사하는 정반사광
- * **Emissive** : 자체 발광
- * **Power** : 발광 정도

* 다이렉트X의 개념 - 출처 : <http://blog.naver.com/znfgkro1/220185340157>

- * **D3DLIGHTTYPE type** : 광원 종류
- * **D3DCOLORVALUE Diffuse** : 표면이 반사하는 난반사광 색상
- * **D3DCOLORVALUE Ambient** : 표면이 반사하는 환경광 색상
- * **D3DCOLORVALUE Specualr** : 표면이 반사하는 정반사광 색상
- * **D3DVECTOR Position** : 광원의 위치 지정 벡터 (방향성광원은 상관없음- 거리에 따라 빛의 세기가 변하지 않음)
- * **D3DVECTOR Direction** : 빛이 향하는 방향 지정 (점광원은 필요 없음 - 주변으로 다 비치기 때문)
- * **float Range;** : 빛이 소멸되는 거리 (가로등 하나가 모든 곳을 다 비추진 않음)
- * **float Falloff;** : (스포츠에서만 사용) 밝은 원과 주변 어두운 원의 밝기 차이
- * **float Attenuation0;** : 상수 감소
- * **float Attenuation1;** : 선형 감소
- * **float Attenuation2;** : 이차 감소
- * **float Theta;** : 스포트 광원에서 가장 밝은 부분 비추는 각도
- * **float Phi;** : 스포트 광원에서 전체를 비추는 각도
- * **화면에 표현되는 색상 : 빛 * 물체의 재질 * 텍스처(존재할 시)**

* 조명 - 조명과 재질 적용할 데이터

- 조명의 위치는 동차좌표(4x4 행렬)로 표현 // 누적된 이동, 회전 변환을 하나의 행렬로 표현 가능
- 마지막 성분이 1이면 점광원이고, 0이면 방향광원(directional light source)
 - 점광원 : 한 점에서 시작해서 퍼져나감
 - 방향광원(집중광원) : 해당 지역만 밝게 비춰줌

조명 - 전역 변수

//재질의 정반사, 난반사, 주변광, 반질거림 특성으로 사용될 데이터

GLfloat matSpec[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat matDiff[] = { 1.0, 1.0, 0.0, 1.0 };

GLfloat matAmbi[] = { 0.5, 0.1, 0.1, 1.0 };

GLfloat matShin[] = { 127.0 };

// 광원의 정반사, 난반사, 주변광 특성으로 사용될 데이터

//GLfloat light[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat lit_specular[] = { 1.0, 1.0f, 1.0f, 1.0f }; // 빛의 정반사

GLfloat lit_diffuse[] = { 0.0, 1.0f, 1.0f, 1.0f }; // 빛의 색깔

GLfloat lit_ambient[] = { 0.5, 1.0f, 1.0f, 1.0f };

// 광원의 위치로 사용될 데이터

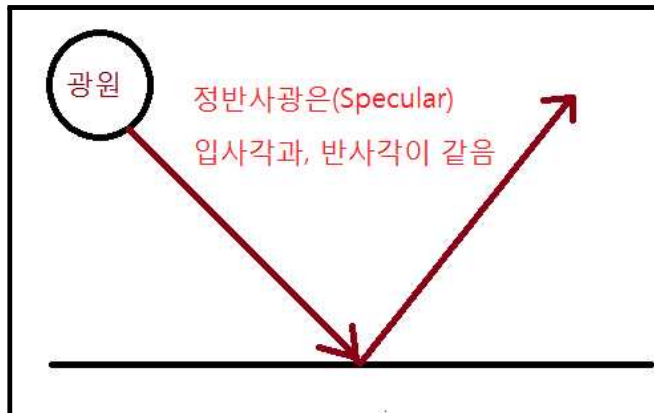
GLfloat lightPos[] = { 1.0, 1.0, 1.0, 1.0 }; // 빛(광원)의 위치

* **반사광** : 어떤 광원에 의해 어떤 오브젝트가 빛을 받았을 때, 그 빛을 반사시키는 빛

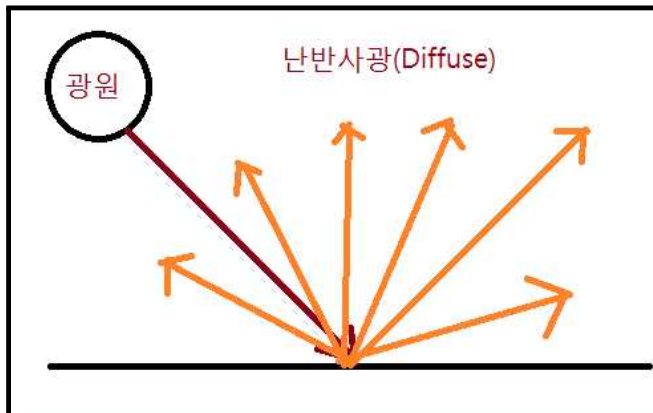
- 광원에 의해 입사광이 있으면, 반사광이 있다.

* **반사광의 종류**

- **정반사광(Specular)** : 어떤 광원(Spot, Dir, Point)에 의해 어떤 오브젝트가 빛을 받았을 때, 그 빛을 한 방향으로 반사시킴(하이라이트 표현)



- **난반사광(Diffuse)** : 어떤 광원(Spot, Dir, Point)에 의해 어떤 오브젝트가 빛을 받았을 때, 그 빛을 여러 방향으로 고르게 반사되는 빛



* **실제 조명과 재질에 적용**

* **glLight*** : 조명의 특성을 설정하는 함수

* **glMaterial*** : 재질의 특성을 설정하는 함수

조명 - LightSet() 함수와 LightPosition() 함수

// 조명과 재질의 특성을 준비된 데이터로 설정하는 함수

```
void LightSet() {  
    glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec); // 마테리얼 정반사  
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff); // 마테리얼 난반사  
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbi); // 마테리얼의 주변광  
    glMaterialfv(GL_FRONT, GL_SHININESS, matShin); // 마테리얼에 빛을 추가해줌(최대 128)  
  
    glLightfv(GL_LIGHT0, GL_SPECULAR, lit_specular); // 빛의 정반사  
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lit_diffuse); // 빛의 난반사  
    glLightfv(GL_LIGHT0, GL_AMBIENT, lit_ambient); // 빛의 주변광  
  
    glEnable(GL_LIGHTING); // 라이트 on  
    glEnable(GL_LIGHT0); // 0번 라이트를 켜 / 기본 8개까지 켤 수 있음  
}
```

// 조명의 위치를 설정하는 함수

```
void LightPosition() {  
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);  
}
```

* 렌더링

* `LightSet()`: : 조명의 특성과 재질의 특성을 설정

* `LightPositioning()`: : 점광원의 위치를 설정

렌더링 설정

```
// 메인에 추가
void GLInit() {
    glClearColor(0.0, 0.0, 0.0, 0.0);

    // 조명의 특성과 재질의 특성을 설정
    LightSet();
    glEnable(GL_DEPTH_TEST);
}

// 디스플레이
void display() {
    [[GL_MODELVIEW 모드 설정]]
    gluLookAt (0, 1, 2, 0, 0, 0, 0, 1, 0);

    // 점광원의 위치를 설정
    LightPositioning();
    gluSolidTeapot(0.5);
    gluSwapBuffers();
}
```



(a) 주변광 포함



(b) 주변광 제외

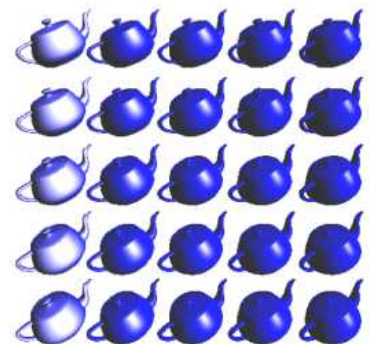
* 반질거림 조정

* `glMaterialf(GL_FRONT, GL_SHININESS, 4.0f + 123.0f * (i / 4.0));`

- 재질의 반질거림을 변경하여 설정

마테리얼 설정

```
for (int i=0; i<5; i++){
    // 재질의 반질거림을 변경하여 설정하고 주전자를 그림
    glMaterialf(GL_FRONT, GL_SHININESS, 4.0f+123.0f*(i/4.0));
    for (int j=0; j<5; j++){
        glPushMatrix();
        glTranslated(float (i)+0.5, float (j)+0.5, 0.5);
        glRotated(45, 1, 1,1);
        glutSolidTeapot(0.4);
        glPopMatrix();
    }
}
```



○ 반사각 구하는 방법

$$- 2d = 2LN$$

$$- R = -L + (2d)N$$

$$= -L + 2(L*N)*2$$

- * 점광원 구현 // 한 점에 대한 모든 곳을 비춤
- * `glLightfv(GL_LIGHT0, GL_POSITION, lit_position);`
 - `GL_LIGHT0` : 빛의 근원지
 - `GL_POSITION` : 광원(빛)의 위치 지정

점광원 설정

```
[[ 광원과 재질의 색상 설정 ]]
// 광원의 위치를 설정
// 광원의 좌표 가운데 w 성분이 1로 설정되어 있다
// 디스플레이 콜백에서 광원의 위치를 바꾸지만, w 좌표는 변경하지않는다.

void SetLight(){
    glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff);
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbi);
    glMaterialfv(GL_FRONT, GL_SHININESS, matShin);

    glLightfv(GL_LIGHT0, GL_SPECULAR, lit_specular);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lit_diffuse);
    glLightfv(GL_LIGHT0, GL_AMBIENT, lit_ambient);

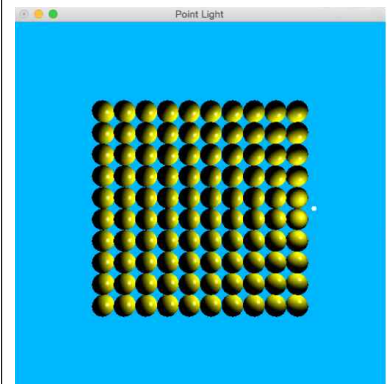
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
}

void init(){
    [[ 윈도우, 버퍼 초기화, 카메라 설정 등의 초기 작업 ]]
    SetLight();
}

void display(){
    [[ 버퍼 지우기, 모델뷰 행렬 모드 설정, 카메라 위치 설정 ]]
    static float t = 0.0;
    t+=0.01;

    // 광원의 위치를 설정함. w좌표는 1로 고정하고 회전하도록 함
    lit_position[0] = 5.0 * sin(t);
    lit_position[2] = 5.0 * cos(t);
    glLightfv(GL_LIGHT0, GL_POSITION, lit_position);

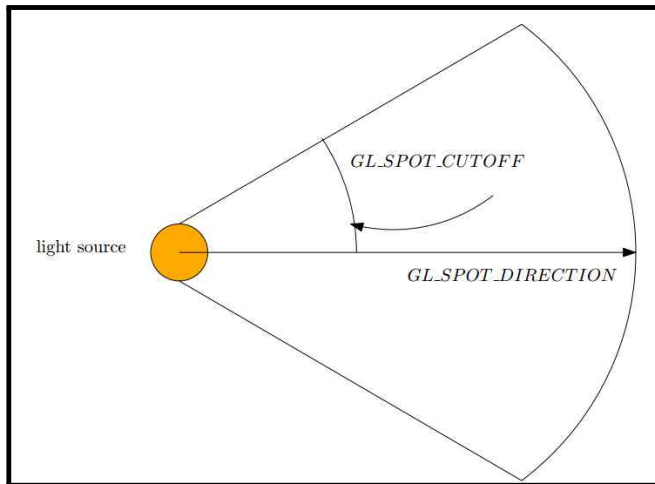
    for (int i=0; i<10; i++){
        for (int j=0; j<10; j++){
            glPushMatrix();
            glTranslated(i-4.5, j-4.5, 0);
            glutSolidSphere(0.5, 30, 30);
            glPopMatrix();
        }
    }
    [[ 광원의 위치 등을 표시하여 확인할 수 있도록 함 ]]
    glutSwapBuffers();
}
```



* 집중광원 구현 // 위치를 지정하고 주변을 밝힘

* 집중광원에 필요한 요소

- light source : 빛의 근원지
- GL_SPOT_CUTOFF : 빛의 각도
- GL_SPOT_DIRECTION : 빛의 최대 사거리



* glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 20.0f);

- GL_LIGHT0 : 빛의 근원지
- GL_SPOT_CUTOFF : 스포트라이트(빛)의 절단 각도
- 20.0f : 빛의 최대 사거리

* glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, 20.0f);

- GL_SPOT_DIRECTION : 스포트라이트(빛)의 거리 (스�포트라이트 방향지정 벡터)

* glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 20.0f);

- GL_SPOT_EXPONENT : 스포트라이트(빛)의 지수 (초점)

집중광원 설정

// 집중 광원의 방향으로 사용될 데이터를 준비

```
GLfloat spotDir[] = { 0.0f, 0.0f, -1.0f }
```

```
void SetLight(){
```

```
    glEnable(GL_LIGHTING);
```

```
    glEnable(GL_LIGHT0);
```

```
    glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
```

```
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiff);
```

```
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbi);
```

```
    glMaterialfv(GL_FRONT, GL_SHININESS, matShin);
```

```
    glLightfv(GL_LIGHT0, GL_SPECULAR, lit_specular);
```

```
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lit_diffuse);
```

```
    glLightfv(GL_LIGHT0, GL_AMBIENT, lit_ambient);
```

// 집중광원에 필요한 데이터를 설정

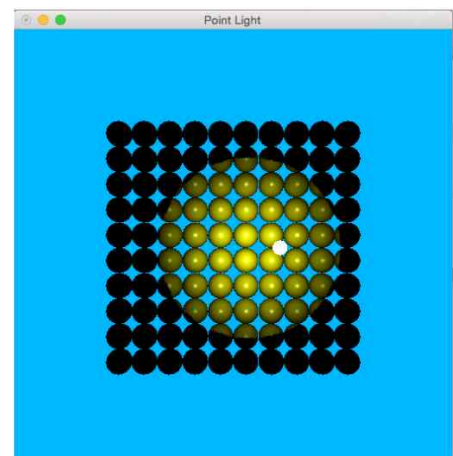
```
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 20.0f);
```

```
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDir);
```

// Exponent 집중광의 테두리를 부드럽게 한다.

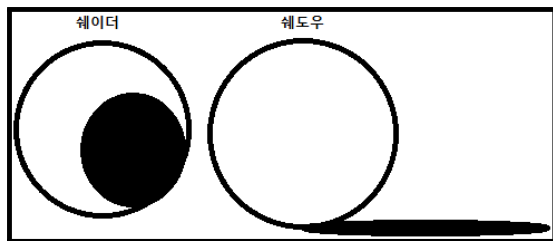
```
    glLightf (GL_LIGHT0, GL_SPOT_EXPONENT, 20.0f);
```

```
}
```



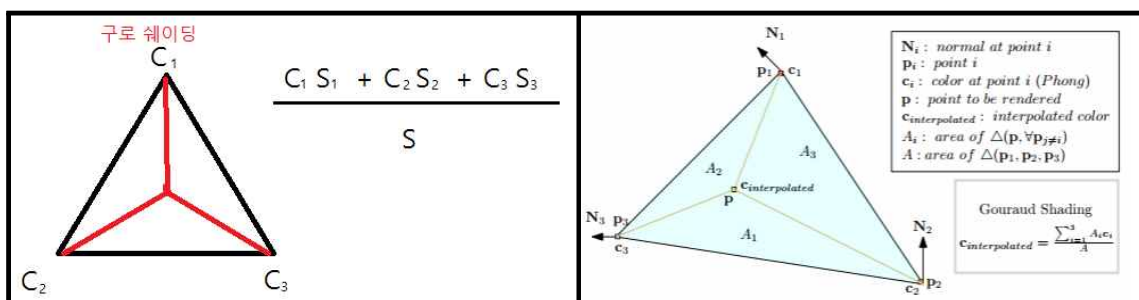
○ 셰이더

* 셰이더 : 물체 내의 그림자



* 구로 셰이딩

- 각 정점에 법선 벡터 정의
- 법선 벡터와 조명의 관계를 이용하여 정점별 Phong 셰이딩
- 정점의 색을 이용하여 내부의 픽셀은 선형보간(linear interpolation)을 통해 얻음
- 세 직선이 만나는 점에 따라 단면 내의 빛 위치와 음영이 바뀜



* 구로 셰이딩 예제

구로 셰이딩 예제

```
glBegin(GL_TRIANGLES);
glNormal3f(0, 1, 0);
glVertex3f(0, 0, 0);
glNormal3f(2/sqrt(3), 1/sqrt(3), 0);
glVertex3f(2, 0, 0);
glNormal3f(-2/sqrt(3), 1/sqrt(3), 0);
glVertex3f(1, 0, -1);
glEnd();
```

구로 셰이딩 예제2 - 두 개의 인접한 면 그리기

```
glBegin(GL_TRIANGLES);
glNormal3f(-1/sqrt(2), 1/sqrt(2), 0);
glVertex3f(0, 1, 0);
glVertex3f(-1, 0, 0);
glVertex3f(0, 1, 1);
glNormal3f(1/sqrt(2), 1/sqrt(2), 0);
glVertex3f(0, 1, 0);
glVertex3f(0, 1, 1);
glVertex3f(1, 0, 0);
glEnd();
```

* 법선 벡터

```
// 외적구하기 / u, v에 3개 짜리 배열을 사용
void Cross(float *u, float *v, float *cross){
    cross[0] = u[1] * v[2] - u[2] * v[1];
    cross[1] = u[2] * v[0] - u[0] * v[2];
    cross[2] = u[0] * v[1] - u[1] * v[0];

    float len = sqrt(cross[0] * cross[0] + cross[1] * cross[1] + cross[2] * cross[2]);
    if (len > 0){
        cross[0] /= len;
        cross[1] /= len;
        cross[2] /= len;
    }
}
```

// RGBA mode 색 지정 //

```
void glColor3f( GLfloat r, GLfloat g, GLfloat b );
void glColor3f( GLfloat r, GLfloat g, GLfloat b, GLfloat a );
void glColor3fv( const GLfloat* v );
void glColor4fv( const GLfloat* v );
```

// 셰이딩 (Shading) //

단색 셰이딩(flat shading) : glShadeModel(GL_FLAT); // 한 면을 같은 색으로 셰이딩
Gouraud shading(smooth shading) : glShadeModel(GL_SMOOTH); // 입체감으로 셰이딩

// OpenGL 의 조명 //

난반사광(분산광) (diffuse light) : 특정방향으로 빛이 들어오고 모든 방향으로 균일하게 반사
정반사광 (specular light) : 특정방향으로 빛이 들어오고 특정 방향으로 반사
주변광 (ambient light) : 특정방향을 가지지 않는 빛.
방사광 (emissive light) : 객체 자체가 광원처럼 빛을 낸다. 객체에 영향을 미치지 않는다.

// OpenGL 의 조명 사용법 //

1. 모든 객체의 모든 정점들에 대해 법선들을 계산. 그 법선들은 광원에 대한 객체의 상대적인 방향을 결정.
2. 모든 광원들을 생성, 선택, 배치
3. 조명 모델을 선택. 조명모델은 주변광 및 조명계산을 위한 시점의 위치 결정
4. 장면안의 객체들에 대한 재질 속성 정의

```
glEnable( GL_LIGHTING ); // 조명 활성화
// 조명값들
float ambientLight[] = { 0.3f, 0.5f, 0.8f, 1.0f }; // 주변광
float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f }; // 분산광
float lightPosition[] = { 0.0f, 0.0f, 0.0f, 1.0f }; // 광원위치
```

재질 값도 동일하게 정의

// 광원 설정 (glLightfv() 함수) //

```
glLightfv( GLenum light( 광원 ), GLenum pname( 속성이름 ), TYPE *param( 속성 ) );
```

// pname 인자들 //

| 사용 코드 | 값 | 의미 |
|---------------------------|--------------------------|---------------------|
| glMaterialfv glLightfv | GL_AMBIENT | 주변광의 세기 |
| | GL_DIFFUSE | 난반사광(분산광)의 세기 |
| | GL_SPECULAR | 정반사광의 세기 |
| glLightfv | GL_POSITION | 광원위치지정 |
| | GL_SPOT_DIRECTION | 스포트라이트 방향지정 벡터 |
| glLightf | GL_SPOT_EXPONENT | 스포트라이트 지수 |
| | GL_SPOT_CUTOFF | 스포트라이트 절단 각도 |
| | GL_CONSTANT_ATTENUATION | 불변 감쇠 값 (빛이 줄어드는 값) |
| | GL_LINEAR_ATTENUATION | 선형 감쇠 값 (빛이 줄어드는 값) |
| | GL_QUADRATIC_ATTENUATION | 2차 감쇠 값 |

정리 - 2

```
glEnable ( GL_LIGHTING );
glEnable( GL_LIGHT숫자 );          // 조명 ON
float ambientLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };          // 흰색 주변광
glLightfv( GL_LIGHT숫자, GL_AMBIENT, ambientLight );      // 주변광을 지정
Diffuse, Specular 도 동일하게 처리
[[ 광원위치 ]]

// w 값이 0 이면 특정방향만 가진 지향광( directional light ),태양광 1 이면 위치와 방향을 함께 가짐
float lightPosition[] = { 0.0f, 0.0f, 1.0f, 0.0f };

glLightfv( GL_LIGHT0, GL_POSITION, lightPosition );

// w 값이 1 이면 위치와 방향을 함께 가지는 위치광( position light ) / 전구 램프등
float lightPosition[] = { 0.0f, 0.0f, 0.0f, 1.0f };
glLightfv( GL_LIGHT0, GL_POSITION, lightPosition );

-----

// 조명 설정 함수 //
void Initialize()
{
    glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );    // 검은색으로 화면 지움
    glShadeModel( GL_SMOOTH );                // 매끄러운 셰이딩 사용
    glEnable( GL_DEPTH_TEST );                // 가려진 면 제거
    glEnable( GL_CULL_FACE );                 // 후면 제거
    glFrontFace( GL_CCW );                    // 다각형을 반시계방향으로 감는다.

    glEnable( GL_LIGHTING );                  // 조명 활성화

    // LIGHT0 에 대한 재질을 설정
    glMaterialfv( GL_FRONT, GL_AMBIENT, matAmbient );
    glMaterialfv( GL_FRONT, GL_DIFFUSE, matDiff );

    // LIGHT0 설정
    glLightfv( GL_LIGHT0, GL_AMBIENT, ambientLight );      // 주변광 성분 설정
    glLightfv( GL_LIGHT0, GL_DIFFUSE, diffuseLight );      // 분산광 성분 설정
    glLightfv( GL_LIGHT0, GL_POSITION, lightPosition );    // 광원 위치 설정

    // LIGHT0을 켜다.
    glEnable( GL_LIGHT0 );

    // glColor3f를 색으로 받아들이기 위함
    glEnable( GL_COLOR_MATERIAL );
}

-----

// 스포트라이트 //
스포트 라이트 : 절단각( spotlight cutoff angle ), 방향, 초점 필요
절단각 : GL_SPOT_CUTOFF 로 설정
방 향 : GL_SPOT_DIRECTION 로 설정( 기본 값은 (0, 0, -1) 음의 z축 방향)
초 점 : GL_SPOT_EXPONENT 로 설정
예) float spotlightDirection[] = { 0.0f, 0.0f, -1.0f };          // 스포트라이트 방향
    glLightf( GL_LIGHT1, GL_SPOT_CUTOFF, 40.0f );                // 80도 원뿔
    glLightf( GL_LIGHT1, GL_SPOT_EXPONENT, 80.0f );              // 초점 설정
    glLightfv( GL_LIGHT0, GL_SPOT_DIRECTION, spotlightPosition ); // 방향 설정

-----

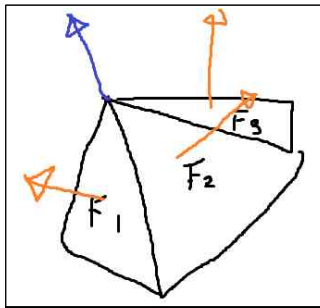
반투명 효과 :
// 혼합을 켜다.
glEnable( GL_BLEND );
// 깊이 버퍼를 읽기 전용으로 설정
glDepthMask( GL_FALSE );
// 반투명 효과를 위한 혼합 함수들을 설정
glBlendFunc( GL_SRC_ALPHA, GL_ONE );
// 반투명 구를 그린다.
DrawSphere();
// 깊이 버퍼를 보통 모드( 쓰기 가능 )로 되돌림
glDepthMask( GL_TRUE );
// 혼합 끄다
glDisable( GL_BLEND );
```

○ 메시 (mesh)

* 메시

- 임의의 면을 그릴 때는 법선 벡터가 존재하지 않음
- Phong 셰이딩 계산이 요구되는 법선 벡터를 오픈지엘에 넘겨주어야 함. (파일 입출력)

* 법선 벡터 구하는 방법



면적이 크면 클수록 더 큰 벡터가 되니,
다 더해버리면 면적을 고려한 상태로 더해짐.
작은 면적은 작은 벡터로 만들어짐.
큰 면적 두 벡터의 외적을 정규화 하지 않고 그대로 씀
(마지막에 한 번만 정규화하면 됨)

* 메시 예제

- 점도 저장해야 부드러운 면 그리기가 가능(법선벡터) // 새로운 면에서 다시 불러일 때 glNormal이 갱신

| 배열 | 점 | 면 |
|----|-----------|-------|
| 0 | -1 1 -1 | 0 1 3 |
| 1 | -1 1 1 | 0 3 2 |
| 2 | -0.5 0 -1 | 2 3 5 |
| 3 | -0.5 0 1 | 2 5 4 |
| 4 | 0.5 0 -1 | 4 5 7 |
| 5 | 0.5 0 1 | 4 7 6 |
| 6 | 1 1 -1 | |
| 7 | 1 1 1 | |

메시 예제 - femail

```
// mesh
#define fscanf fscanf_s

struct vertex {
    float x, y, z;
};
int nVertex = 0;
vertex *verts;

void showMesh(vertex *v, int n) {
    glDisable(GL_LIGHTING);
    glBegin(GL_POINTS);
    for (int i = 0; i < n; i++) {
        glVertex3f(v[i].x, v[i].y, v[i].z);
    }
    glEnd();
}

void readMesh(char *fName) {
    // fname의 파일을 열어줌, "r" read 모드로 옴
    FILE *f;
    fopen_s(&f, fName, "r");
    // f파일에서 %d(정수)를 읽고, &nVertex의 크기를 만들
    fscanf(f, "%d", &nVertex);
    verts = new vertex[nVertex];
    for (int i = 0; i < nVertex; i++) {
        fscanf(f, "%f", &verts[i].x);
        fscanf(f, "%f", &verts[i].y);
        fscanf(f, "%f", &verts[i].z);
    }
}

void display() {
    // world
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    static float angle = 0;
    gluLookAt(200.0*cos(angle), 3, 200.0*sin(angle),
0, 0, 0, 0, 1, 0); angle += 0.01;

    SetLightPosition();

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLineWidth(1);
    //draw axes
    drawAxes(1.0);
    glEnable(GL_LIGHTING);
    // draw mesh
    showMesh(verts, nVertex);
    glDisable(GL_LIGHTING);
    glutSwapBuffers();
}

void init(void) {
    glClearColor(0, 0, 0, 1);
    glEnable(GL_DEPTH_TEST);

    // light enable
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    SetLighting();

    // read mesh
    readMesh("femail.sms.txt");
}
```

○ 함수 합축

* void init();

void init(); 함수 // 메인에서 초기화 시켜주는 함수들을 모아놓음

```
// 초기화를 함
void init(){
    glClearColor(0, 0, 0, 1);
    glEnable(GL_DEPTH_TEST);

    // light enable
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    SetLighting();
}
```

○ 헤더파일 분할

* OpenGLHeaders.h : include를 모아놓음

OpenGLHeaders.h

```
#ifndef OPENG_L_HEADERS_20161103
#define OPENG_L_HEADERS_20161103

#include <Windows.h>
#include <gl/GL.h>
#include <gl/glut.h>
#include <math.h>

#endif
```

* CameraMgr.h : Camera 함수를 모아놓음

CameraMgr.h

```
#ifndef CAMERAMGR_20161103
#define CAMERAMGR_20161103

void SetCamera();
void UpdateAspectRatio(float asp);

#endif
```

* LightMgr.h : Light 함수를 모아놓음

LightMgr.h

```
#ifndef LIGHT_20161103
#define LIGHT_20161103
#include "OpenGLHeaders.h"

void SetLighting(void);
void SetLightPosition(void);

#endif
```

* Object이름Mgr.h : 사용할 오브젝트의 동작들을 지정

ObjectMgr.h 예제

```
#ifndef _BALL_20161103
#define _BALL_20161103

void BallMove();
void BallDraw();
void BallSet(float x, float y, float z, float vx, float vy, float vz);

#endif
```

□ 텍스처

○ 텍스처 종류

* 컬러 매핑

- 색을 이용하는 방법

* 범프 매핑(Bump Mapping)

- 텍스처를 사용하여 법선 벡터를 조작하여 높 낮이를 구현하는 방법

* 범프 맵

- 범프 맵핑에 사용되는 텍스처로 주로 흑백으로 된 이미지의 밝기로 간단한 높낮이를 구현

* 노말 맵 (법선맵)

- 범프맵의 개량형으로 범프맵과는 다르게 RGB 값을 노말 값으로 사용하여 좀 더 높은 질감을 표현

* 라이트 매핑(Light Mapping)

- 조명 모델에 의해 물체면의 밝기를 계산하는 대신 조명의 결과 영상을 직접 물체 면에 입히는 것

* 주변(환경) 매핑(Environmental Mapping)

- 거울에 주위가 반사되는 것처럼 물체 외부환경이 해당 물체에 반사되어 보이도록 하는 방법

* 밍 매핑(MIP Mapping)


- 텍셀과 픽셀의 축소관계의 경우 텍셀의 평균을 구하여 해당 픽셀을 칠함으로써 안티 에일리어싱 효과를 기할 수 있는데, 텍스처 영역에서 이러한 계산을 미리 수행해서 저장함으로써 처리 속도를 높임

○ 텍스처 시 필요한 중요한 4가지

- `Vertex3f(x, y, z)` : 정점 좌표
- `glColor3f(r, g, b)` : 칼라
- `glNormal3f(n[v0].x, n[v0].y, n[v0].z)` : 노말 벡터
- `glTexCoord2f(s, t)` : 텍스처가 정점의 어느 좌표에 해당하는 지 알려줌
- * `glMatrixMode(GL_TEXTURE)` : 텍스처 수정 매트릭스 모드(텍스처를 수정할 때 사용)
- * 텍스처는 `glColor3f`에 영향을 받는다.

텍스처 정리 - 1

// 텍스처 //

- * CreateTexture() : 텍스처 이미지 생성
- * SetupTexture() : GPU로 이미지를 보내는 작업
 - glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, TEXTSIZE, TEXTSIZE, 0, GL_RGB, GL_UNSIGNED_BYTE, &myTex[0][0][0]); : GPU로 이미지를 보냄
 - glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
: S쪽으로 CLAMP (마지막 컬러 값이 이어짐)
 - glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
: T쪽으로 REPEAT (그림 반복)
 - glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
: 필터 최소 값
 - glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
: 필터 최대 값
 - * GL_LINEAR(선형보간) : x 값과 y 값을 섞어서 씬 
 - * GL_NEAREST : 제일 가까운 픽셀을 선택
 - glEnable(GL_TEXTURE_2D)
- * glMatrixMode(GL_TEXTURE); : 매트릭스 모드를 텍스처로 설정 (텍스처 수정)
 - glTranslatef(float x, float y, float z);
 - glRotate3f(float angle, int x, int y, int z);
 - glScale3f(float x_scale, float y_scale, float z_scale);

// 텍스처 인자들 //

| 사용 코드 | 값 | 의미 |
|-----------------|------------|-----------------------|
| glTexParameterf | GL_CLAMP | 마지막 컬러 값이 이어짐 |
| | GL_REPEAT | 그림 반복 |
| | GL_LINEAR | 선형보간, x 값과 y 값을 섞어서 씬 |
| | GL_NEAREST | 제일 가까운 픽셀을 선택해 줌 |

텍스처 정리 - 2

// 텍스처 //

* glGenTextures(GLsizei n, GLuint* textures); : CPU에는 텍스처 하나만 두고, GPU에 텍스처를 여러 개(GLsizei n)를 뒀서, 이름(GLuint* textures)을 불러 어떤 텍스처를 부를 것인지 정함

// 텍스처 매핑 방법 (강교수님 수업) //

1. imageData <- loadImage(STRIImage) : CPU Memory

2. glTexImage2D : GPU image(Texture)

3. Draw with TexCoord

- glVertex3f (좌표)
- light : glNormal3f (법선)
- glColor3f (색상)
- TexCoord (매핑) // s, t

* 전방 매핑 : 가야할 곳으로 바로 보냄 (빈 공간이 생기기 때문에 쓰지 않음)

* 후방 매핑 : 매핑할 좌표를 대칭하여 들고옴

- GL_NEAREST : 가까운 곳의 좌표의 색을 들고옴

- GL_LINEAR : 두 좌표의 사이 값(색)을 자동으로 들고옴

// 여러 개의 텍스처 불러오기 //

* 필요한 변수

- GLuint tex[2]; // 핸들러, 보통 int로 관리, [] 안에 사용할 이미지 개수

- unsigned char *myTex; // CPU에 임시 저장할 이미지

* glGenTextures(int n, GLuint *textures);

: 몇 개의 텍스처(int n)를 GPU로 보낼 것인지 정하고, 사용할 텍스처 배열을 넣어준다.(GLuint *t)

* glBindTexture(GLenum target GL_TEXTURE_2D, GLuint texture);

: GPU에 저장한 텍스처의 타겟(2D인지 3D인지)을 정하고, 몇 번의 텍스처를 쓸 것인지를 정함

* void PrepareTextures() {

1. glGenTexture(2, tex); 로 두 개의 이미지를 쓴다고 정의

2. glBindTexture(GL_TEXTURE_2D, tex[i]); 로 tex[i]를 GL_TEXTURE_2D에 Bind 시킴

3. mytex = stbi_load(img, &texWidth, &texHeight, &bitPerPixel, 0); 으로 image를 읽어 들임

4. glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, texWidth, texHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, myTex);
: GPU로 텍스처를 보내고, tex[0]라는 이름으로 관리됨

5. delete[] mytex로 그 때 그 때 CPU의 텍스처를 지워줌 (CPU에 있는 그림은 쓰지 않음)

9. 원하는 그림을 그리고 싶을 때는 glBindTexture(GL_TEXTURE_2D, tex[i])를 새로 부름

}

// 한 물체에 여러 개의 텍스처 붙이기 //

* 필요한 변수

- GLuint tex[2]; // 핸들러, 보통 int로 관리, [] 안에 사용할 이미지 개수

- unsigned char *myTex; // CPU에 임시 저장할 이미지

* glMultiTexCoord2f(GL_TEXTURE0, 0, 0);

: 몇 번째 텍스처를 쓸 것인지 정하고, Coord 좌표를 적어 넣음 // glVertex3f 각각에 전부 입력

* glActiveTexture(GL_TEXTURE0);

: TEXTURE0번을 활성화

* void PrepareTextures() {

1.

}

// 텍스처 //

* 텍스처 매핑 방법 (2D 텍스처)

1. 먼저 사용할 텍스처의 종류를 glEnable 함수로 활성화해 준다.(주로 2D: GL_TEXTURE_2D)
 2. 텍스처로 사용할 그림을 불러오거나 배열로 임의의 텍스처 행렬을 생성한다.
 - 간단한 텍스처 행렬은 배열명[넓이][높이][3] 으로, 3은 RGB의 값이다.
 3. 배열에 저장된 데이터를 텍스처로 사용하기 위해 2D를 예로 glTexImage2 함수를 호출
 4. glTexCoord2 함수를 보간에 의해 자동적으로 맵핑을 하기위해 glTexGen 함수를 호출
- * 추가로, 에일리어싱을 방지하기 위해 원근에 따른 텍스처 보정을 가하기위해 glHint 함수를 사용 (하드웨어나 드라이버에 따라 다르게 해석 되므로 무시될 수 있음.)
5. glTexParameter 함수로 텍스처 파라미터를 설정하기 위해 두 종류로 구분한다.
 - 주어진 텍스처를 어떻게 확장할 것인가
 - 주어진 픽셀의 텍스처를 어떻게 계산할 것인가
 6. glTexEnv함수를 사용하여 색을 결정한다.

(텍스처를 물체면에 그대로 입힐 수 있지만, 물체 색과 조합할 수 있다.)

 - 완전히 텍스처 색으로만 색 결정
 - 물체면의 색과 텍스처의 색을 혼합

- * void glTexImage2D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid* texture);
- target : GL_TEXTURE_2D 같은 텍스처의 타입을 의미
 - level : mip 맵의 레벨을 의미
 - width, height : 텍셀 단위의 텍스처의 폭과 높이
 - border : 텍스처를 둘러싸는 윤곽선을 별도로 칠할 수 있는데, 이 때 윤곽 선의 두께를 몇 텍셀로 할 것인지를 의미

// internalformat //

| 상수 | 텍셀 구성요소 |
|--------------------|--------------|
| GL_RGBA | R, G, B, A |
| GL_RGB | R, G, B |
| GL_ALPHA | A |
| GL_LUMINANCE | intensity |
| GL_LUMINANCE_ALPHA | intensity, A |

// format, type 상수 //

| 파라미터 | 상수 | 의미 |
|--------|-------------------|-------------------|
| FORMAT | GL_COLOR_INDEX | 칼라 인덱스 |
| | GL_RGB | R, G, B |
| | GL_RGBA | R, G, B, A |
| | GL_RED | Red |
| | GL_GREEN | Green |
| | GL_BLUE | Blue |
| | GL_ALPHA | Alpha |
| Type | GL_UNSIGNED_BYTE | Unsigned 8bit 정수 |
| | GL_BYTE | Signed 8bit 정수 |
| | GL_UNSIGNED_SHORT | Unsigned 16bit 정수 |
| | GL_SHORT | Signed 16bit 정수 |
| | GL_INT | Signed 32bit 정수 |
| | GL_FLOAT | Signed 8bit 정수 |

텍스처 정리 - 4

// 텍스처 //

// 자동 매핑 //

* void glGenTextures(GLenum coord, GLenum pname, GLint param)

- 파라미터들을 설정한 후, 만약 GL_S를 사용하면 glEnable로 GL_TEXTURE_GEN_S를 활성화
- SPHERE_MAP : 환경 맵핑할 때 사용
- OBJECT_MAP : 물체에 고정된 텍스처를 입힘
- EYE_LINEAR : 물체가 아닌 좌표계에 텍스처를 입힐 때 사용

| coord | pname | param |
|------------------------------|---------------------|------------------|
| GL_S GL_T GL_R GL_Q | GL_TEXTURE_GEN_MODE | GL_OBJECT_LINEAR |
| | | GL_EYE_LINEAR |
| | | GL_GL_SPHERE_MAP |
| | GL_OBJECT_PLANE | Plane Array |
| | GL_EYE_PLANE | Plane Array |

* void glTexEnvf(GLenum target, GLenum pname, TYPE param)

- TYPE : 앞에 I인지, f인지에 따라 GLint 또는 GLfloat가 될 수도 있다.
- target : 꼭 GL_TEXTURE_ENV로 해야 한다.
- Ca : 텍스처의 RGB Aa는 텍스처의 알파 값
- Cb : 물체의 RGB
- Ab : 물체면의 알파 값
- glGenTextures 함수로 사용할 개수와 객체명을 저장할 배열을 넘겨준다.
- glBindTexture 함수를 호출하고 파라미터로 사용할 텍스처의 종류와 Gen으로 받은 객체명을 넘겨준다.
- 사용할 텍스처의 설정함수들을 호출해주며, 사용할 때 glBindTexture 함수를 사용한다.

| pname | param |
|----------------------|-------------|
| GL_TEXTURE_ENV_MODE | GL_DECAL |
| | GL_REPLACE |
| | GL_MODULATE |
| | GL_BLEND |
| GL_TEXTURE_ENV_COLOR | RGBA 배열 |

| param | GL_RGB | GL_RGBA |
|-------------|-----------------------------|-------------------------------|
| GL_REPLACE | C = Cb, A = Aa | C = Ca, A = Aa |
| GL_MODULATE | C = CaCb, A = Aa | C = CaCb, A = AaAb |
| GL_DECAL | C = Cb, A = Aa | C = Ca(1-Aa) + CbAb, A = Aa |
| GL_BLEND | C = Ca(1-Cb) + CcCb, A = Aa | C = Ca(1-Cb) + CcCb, A = AaAb |