5장. 오픈지엘 기본틀

학습목표

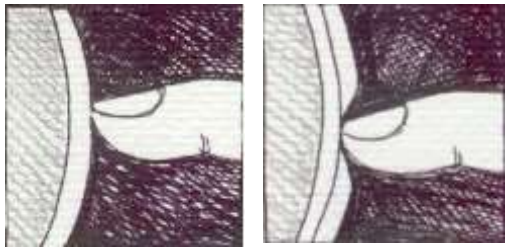
- 논리적 입력장치를 설정하는 이유와 종류를 이해한다.
- 세 가지 입력모드의 차이점을 이해한다.
- GLUT 콜백함수의 종류와 사용법을 이해한다.
- GL의 화면 좌표계와 GLUT의 화면 좌표계 사이의 차이점을 이해한다.
- 더블 버퍼링의 필요성에 대해 이해한다.
- 정점 배열, 디스플레이 리스트의 필요성과 사용법을 이해한다.

물리적 입력장치

- 마우스, 조이스틱, 트랙볼, 스페이스 볼
  - 상대입력과 절대입력

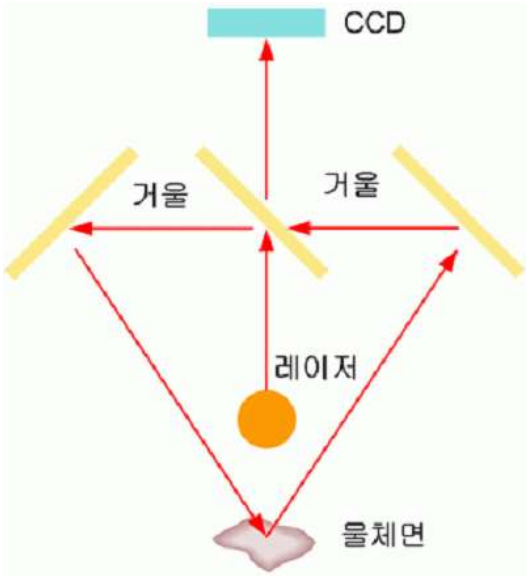
- 타블렛, 스타일러스 펜
  - 크로스 헤어 커서
  - 디지타이징

- 터치 패널
  - 광학 패널, 전기 패널



물리적 입력장치

- 3D 스캐너
  - 물체 표면의 X, Y,
  - 레이저
  - 촬상소자(CCD)



- 버튼 박스와 다이얼
  - 버튼 박스: 매크로 기능
  - 다이얼:
    - 물체에 대한 기하변환
    - 아날로그 방식

논리적 입력장치

- 입력을 논리적으로 취급
  - scanf("%d", &x); 키보드? 버튼박스?
  - 물리적 입력장치가 바뀌어도 프로그램은 동일

- 좌표 입력기(Locator)
  - 절대좌표 또는 상대좌표. 마우스, 키보드의 화살표 키, 트랙 볼
- 연속좌표 입력기(Stroke)
  - 일련의 연속 좌표. 마우스, 태블릿 커서.
- 문자열 입력기(String)
  - 문자열. 키보드.
- 스칼라 입력기(Valuator)
  - 회전각, 크기조절 비율 등 스칼라 값. 키보드, 마우스, 다이얼
- 메뉴선택 입력기(Choice)
  - 메뉴, 서브메뉴, 메뉴옵션 선택. 마우스, 키보드, 터치 패널, 음성
- 물체선택 입력기(Pick)
  - 물체를 선택. 마우스나 터치 패널

입력 모드

- 메저와 트리거
  - 메저(Measure): 응용프로그램에게 전달되는 입력값
  - 트리거(Trigger): 전달하라는 신호

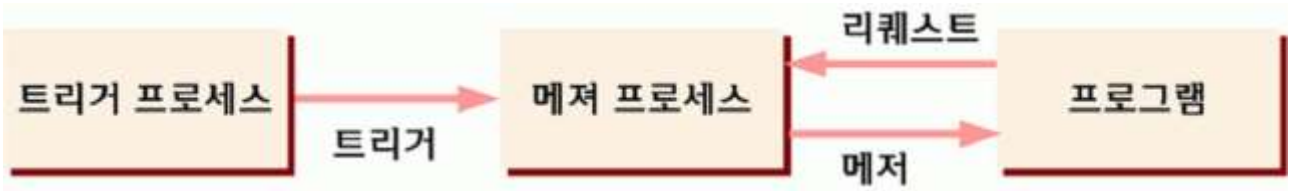
- Ex. 메저 / 트리거
  - DIR <ENTER>
  - 마우스 좌표와 클릭
  - 선택된 메뉴 아이디와 클릭

- 메저 프로세스
  - 운영체제 초기화 시에 실행
  - 항상 시스템 버퍼에 메저값이 저장되어 있음.

리퀘스트 모드

- 프로그램이 실행 중 메저를 요구
  - 트리거가 일어날 때까지 대기상태
  - Request_Locator(Device_ID, &Measure);
  - Device_ID 필드에 의해 물리적 입력장비 제어

- 예제
  - 프로그램은 키보드에서 값을 입력 받고 싶다.
  - 프로그램은 키보드 입력 함수(scanf) 호출
  - scanf는 키보드 값(값+enter)가 입력될 때 까지 대기
  - Enter(트리거)가 입력되면 현재 얻은 값을 프로그램으로 전달
  - 트리거가 입력될 때까지 어떤 값이 입력되었는지 알 수 없음

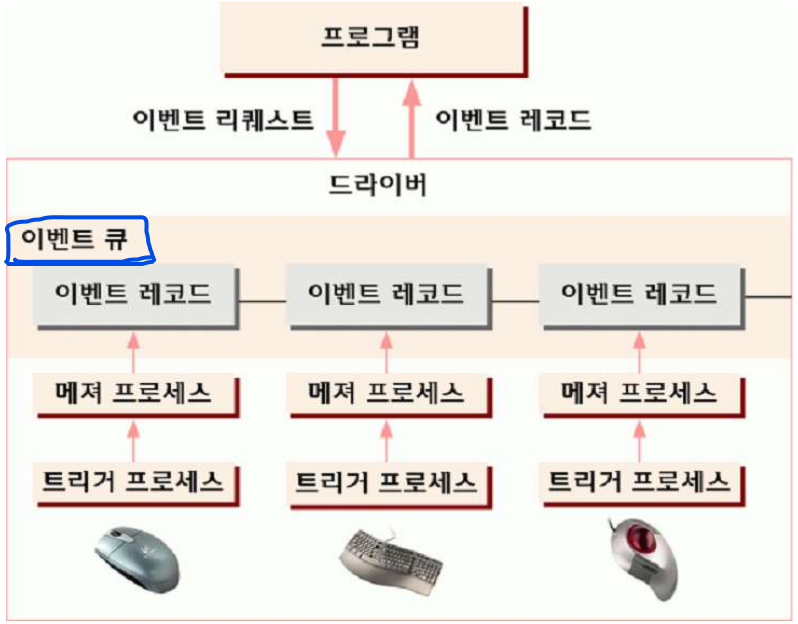


샘플 모드

- 직접모드
  - 사용자 트리거가 불필요
  - sample_Locator(Device_ID, &Measure);
  - 이미 필요한 메저가 준비된 상태
- 예제
  - 프로그램은 Ctrl 키가 현재 눌려있는지 확인하고 싶다.
  - 프로그램은 함수 **GetAsyncKeyState(VK_CONTROL)** 호출
  - 현재 Ctrl 키가 눌려있으면 양수, 아니면 0을 프로그램으로 즉시 전달
  - 각 키(a, b, c, Ctrl …)가 눌리는 순간이 궁금하면, 계속 여러 함수 호출해서 확인해야 함

이벤트 모드

- 이벤트 모드 오픈지엘에서는 이벤트 모드 사용
  - 사용자가 입력 선택 주도권 있음
  - cf. 리퀘스트/샘플 모드: 프로그램이 주도권
- 이벤트 레코드: 이벤트 타입, 장치 아이디, 메저
- 응용 프로그램은 주기적으로 이벤트 큐를 검사
  - 드라이버에게 이벤트 리퀘스트. 드라이버가 큐 프런트 레코드를 전달
  - 큐가 비어있으면 응용 프로그램은 다른 일을 수행
- 예제
  - 키보드가 눌리면, 키보드가 눌렸다는 정보와 키 값이 메모리(이벤트 레코드)에 저장
  - 응용 프로그램은 필요할 때 메모리에서 확인



콜백함수

- 응용 프로그램 구조

이벤트 모드를 프로그램으로 다루기 위해 콜백함수 사용

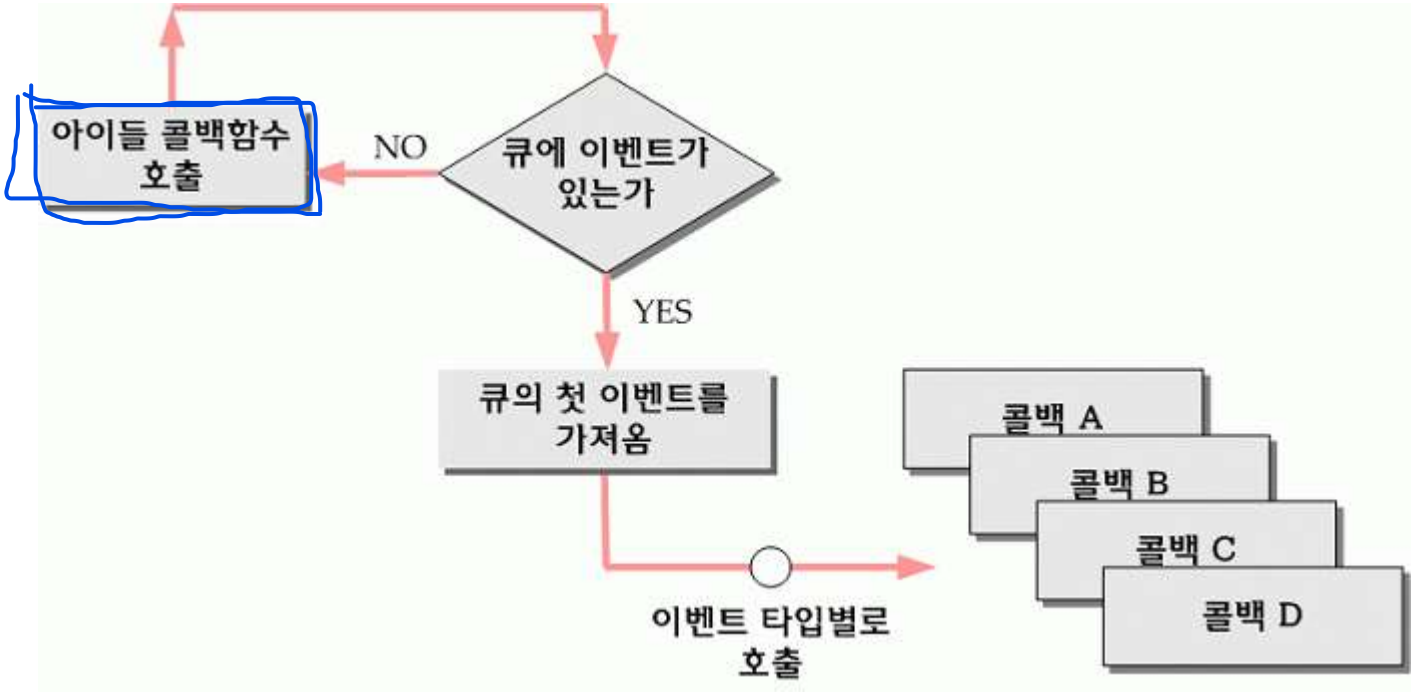
```
Initialize Input Devices;
do
{    if (There Is an Event on the Event Queue)
        switch (Event Type)
        {    case Keyboard Event:
                    Get  Event Record, Run Keyboard Callback
             case Mouse Event:
                  Get  Event Record, Run Mouse Callback
                    ...
        }
       else Do Background Process
}
while (User Does Not Request Escape);
```

지엘의 콜백

이벤트 타입	콜백함수 명
DISPLAY	MyDisplay( )
RESHAPE	MyReshape( )
KEYBOARD	MyKeyboard( )
MOUSE	MyMouse( )
IDLE	MyIdle( )

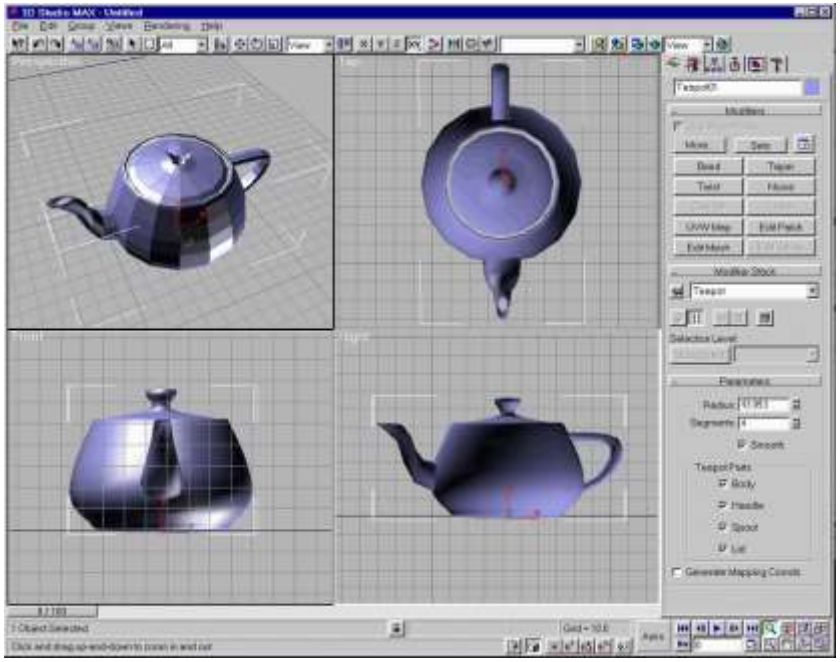
지엘의 콜백

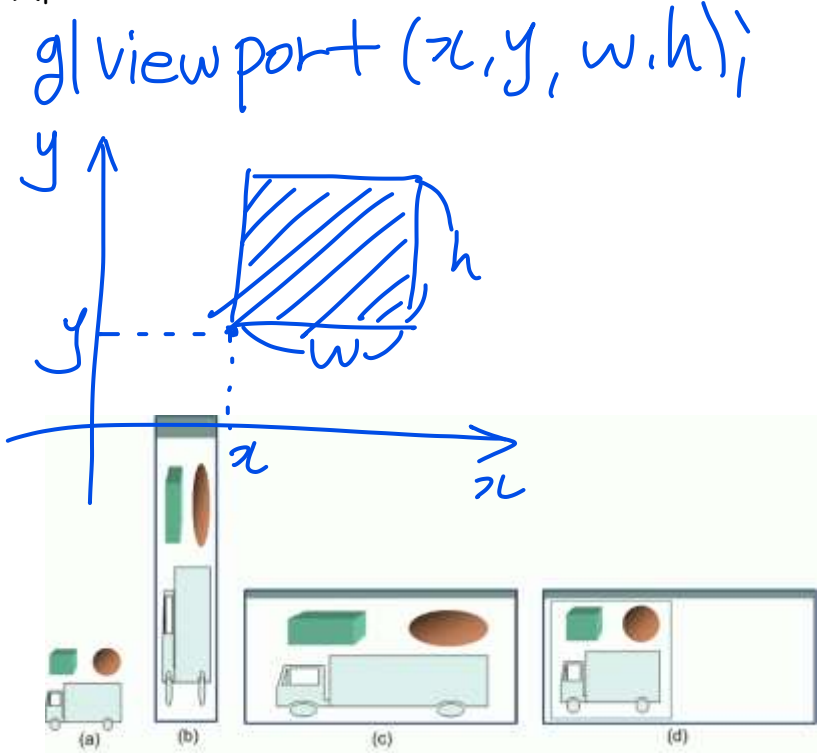
- 아이들 콜백
  - 큐에 이벤트가 없을 때 실행
  - 정의되어 있지 않으면 운영체제는 다른 일을 수행
  - 드라이버를 통해 주기적으로 이벤트 검사



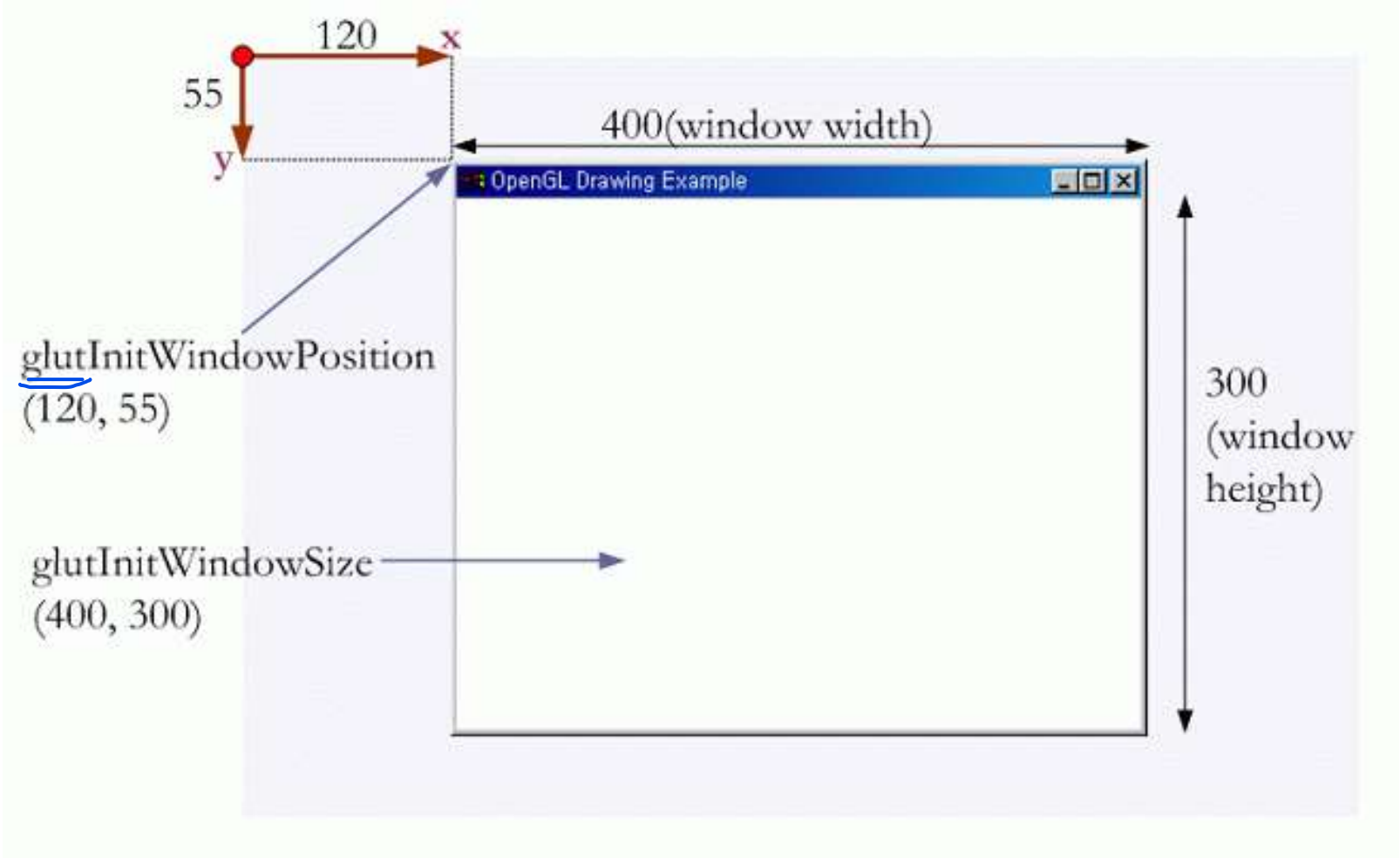
윈도우와 뷰포트

- 윈도우를 분할
  - 그리기가 뷰포트 내부로 제한됨
- 왜곡
  - 뷰포트 미 설정시 기본값으로 윈도우 = 뷰포트
  - 윈도우 크기조절에 따라 뷰포트 내부 그림도 자동으로 크기조절
  - 별도 뷰포트 설정에 의해 왜곡 방지

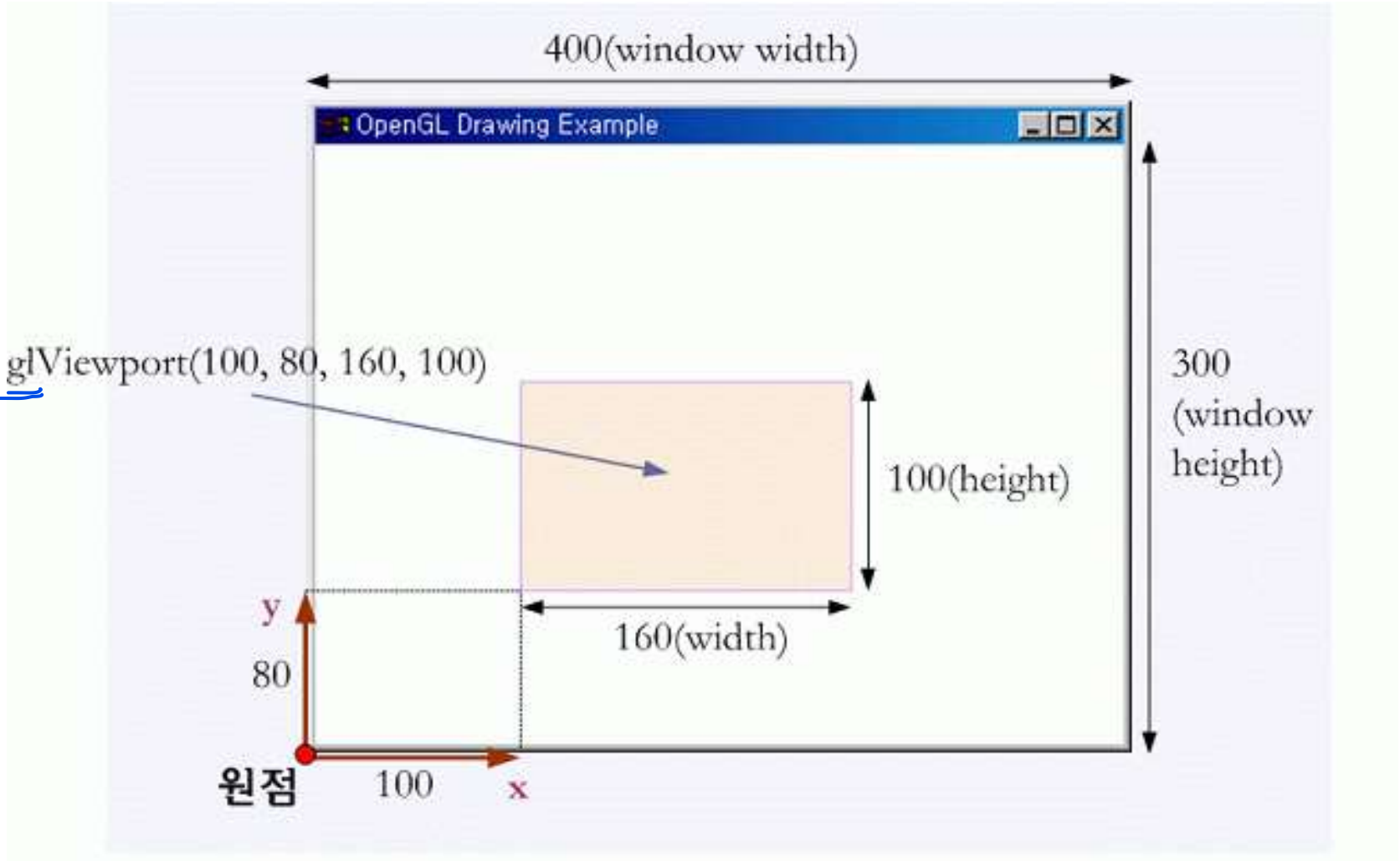




GLUT 윈도우 제어

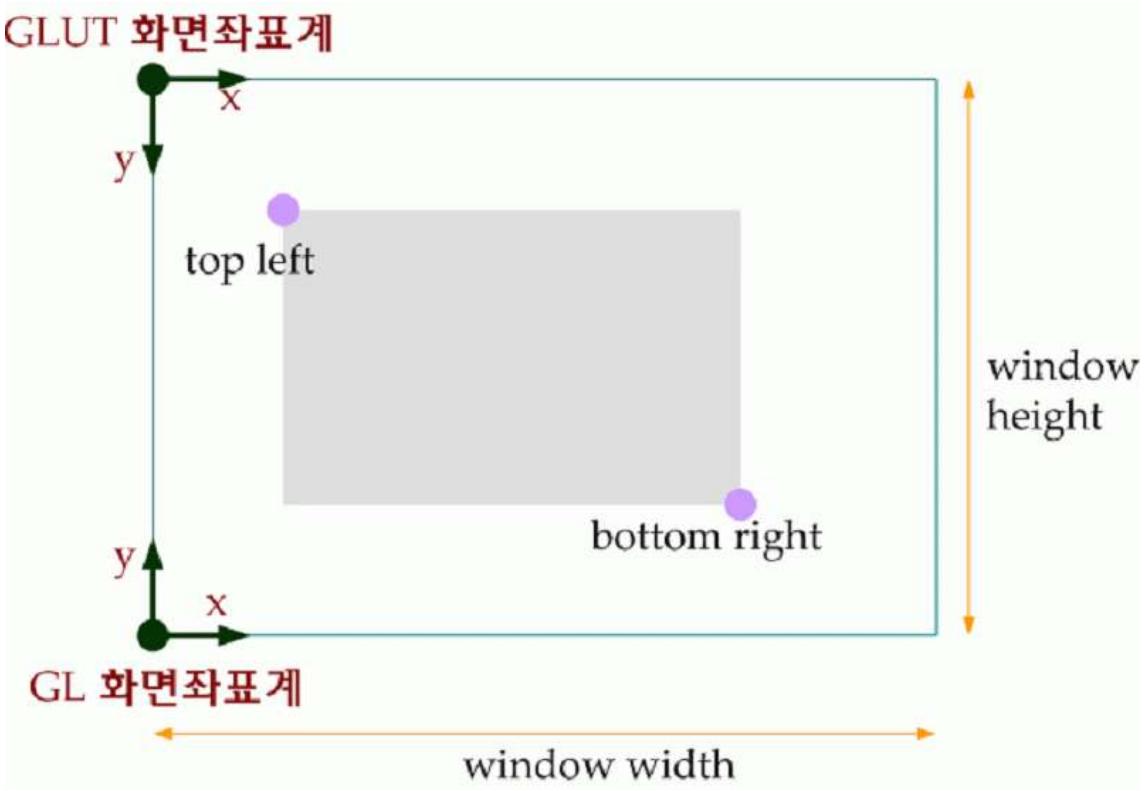

120
x
55
y
400(window width)
OpenGL Drawing Example
glutInitWindowPosition
(120, 55)
glutInitWindowSize
(400, 300)
300
(window
height)

GL의 뷰포트 설정


400(window width)
OpenGL Drawing Example
glViewport(100, 80, 160, 100)
100(height)
300
(window
height)
y
80
160(width)
원점
100
x

GL의 화면좌표, GLUT 화면좌표

- 사무용 프로그램(windows)의 경우 아래방향으로 y좌표가 증가 (문서의 줄 번호)
- 수학 프로그램의 경우, 수학 좌표계에 대응하여 위쪽 방향으로 y좌표가 증가



GLUT 모델링

- 사용자 편의를 위해 이미 모델링 된 몇 가지 물체 제공

- void glutSolidCube(GLdouble size);
- void glutWireCube(GLdouble size); // 0.5

- void glutSolidSphere(GLdouble sradius, GLint slices, GLint stacks);
- void glutWireSphere(GLdouble sradius, GLint slices, GLint stacks); // 0.5, 10, 10

GLUT 모델링

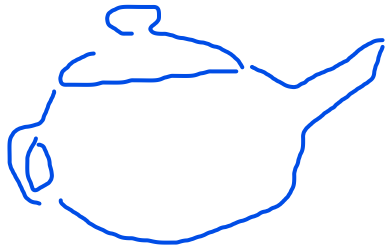
- glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
- glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings); // 0.05, 0.5, 5, 15
  - outerRadius: Torus의 전체 반지름
  - innerRadius: Torus의 굵기

- glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
- glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks); // 0.5, 0.7, 20, 10

GLUT 모델링

- void glutWireIcosahedron();
  - 반지름 1인 정20면체

- void glutWireTeapot(GLdouble size);



glutReshapeFunc [코드 5-5]

- glutReshapeFunc
  - 창의 크기가 변경되면 호출
  - 창의 크기와 연동하여 출력 영상을 변경하는 경우 필요

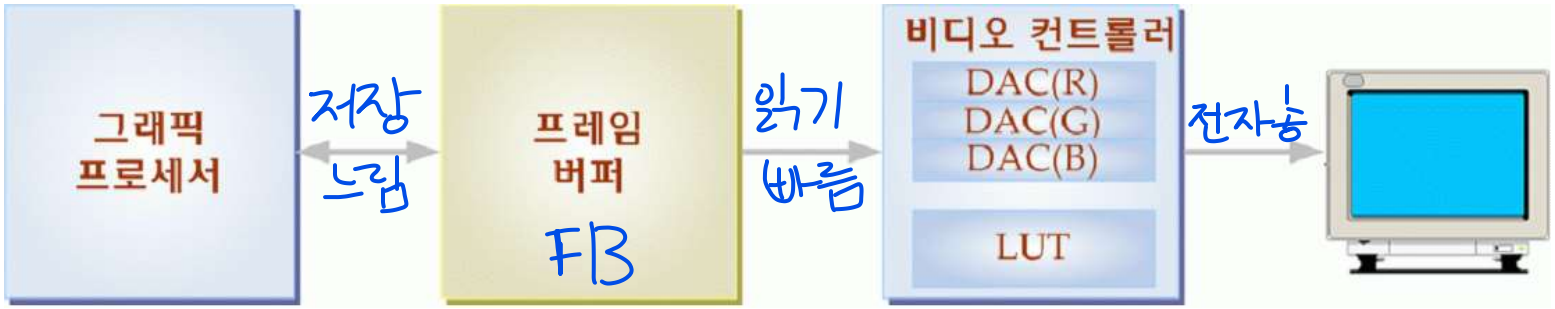
- glOrtho(left, right, bottom, top, near, far);
  - 공간 속의 일부를 관찰(촬영)하는 함수

glutKeyboardFunc [코드 5-5]

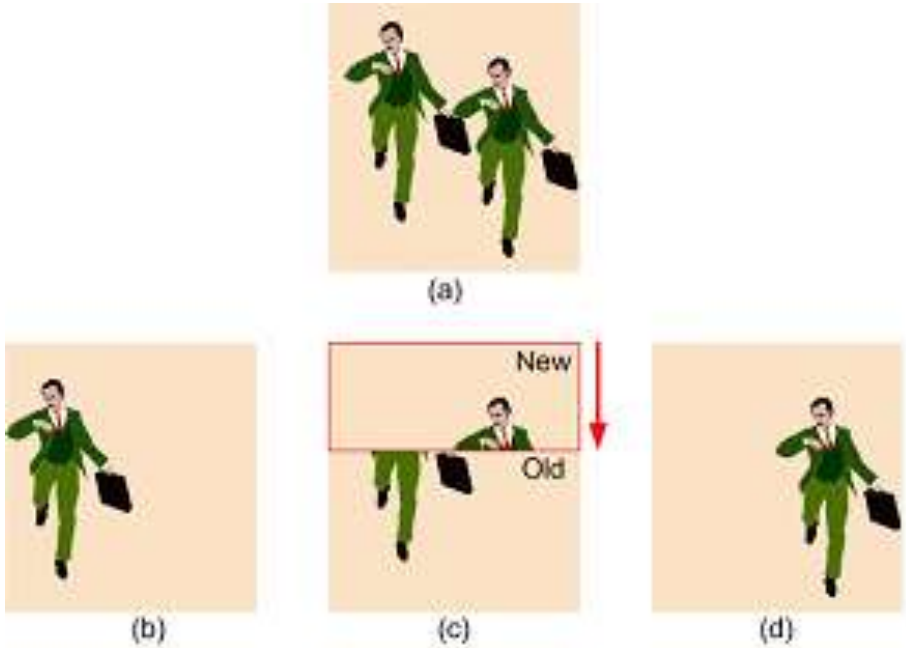
- glutKeyboardFunc
  - 키보드가 눌린 경우
  - void MyKeyboard(unsigned char KeyPressed, int x, int y)
- glutSpecialFunc
  - 특수키가 눌린 경우
  - void MySpecialFunc (int key, int x, int y)
  - GLUT_KEY_F1, GLUT_KEY_LEFT, GLUT_KEY_INSERT 등
- Ctrl+c 키가 눌린 것을 확인
  - if(glutGetModifiers()==GLUT_ACTIVE_CTRL && key=='c')

프레임 버퍼

- 2중 포트 구조
  - Read Port, Write Port
  - 버퍼를 읽어서 화면에 그리는 것은 매우 빠름 (거의 동시, 1/100초)
  - (전통적으로) 버퍼에 기록하는 것은 상대적으로 느림
    - 많은 데이터의 좌표 연산 필요
  - 애니메이션에서 문제가 됨 더블 버퍼링 필요

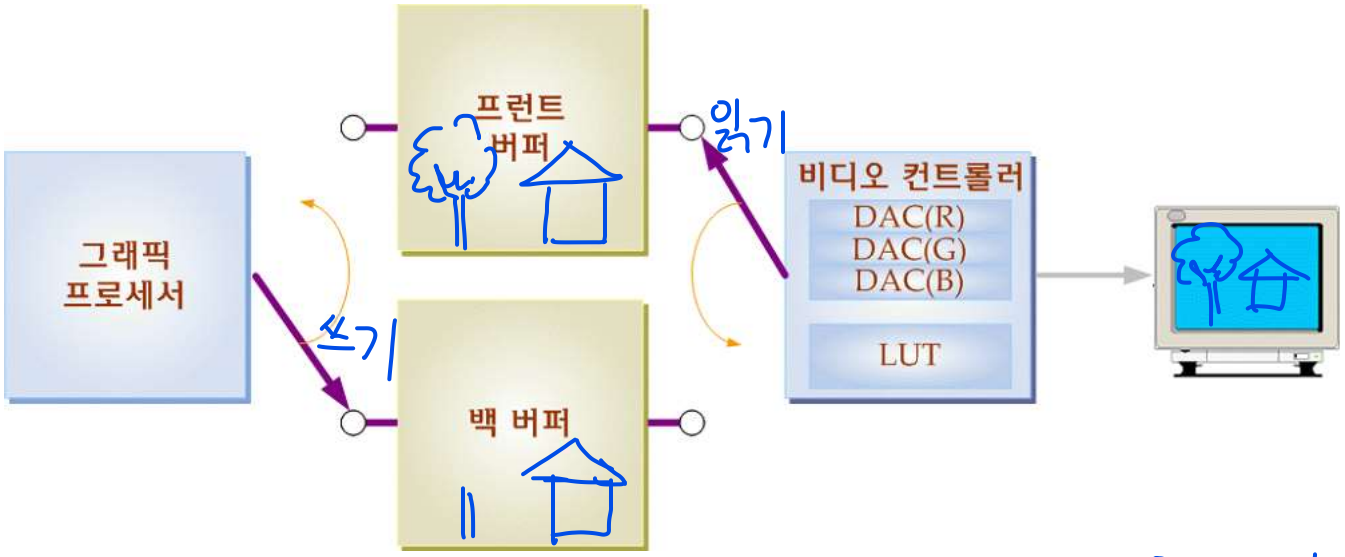


애니메이션 시의 문제



더블 버퍼링

- 프런트 버퍼와 백 버퍼



그리는 중, 다 그릴 때까지 같은 그림 출력,
다 그리면, 노란 화살표를 따라감
프런트/백의 위치가 바뀐

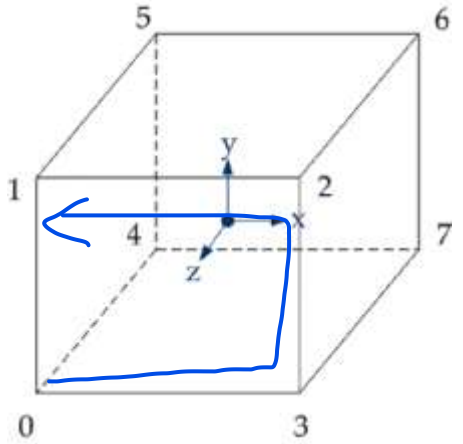
- 참고
  - 트리플 버퍼링
  - G-sync(nVidia), freeSync(AMD)

육면체 그리기

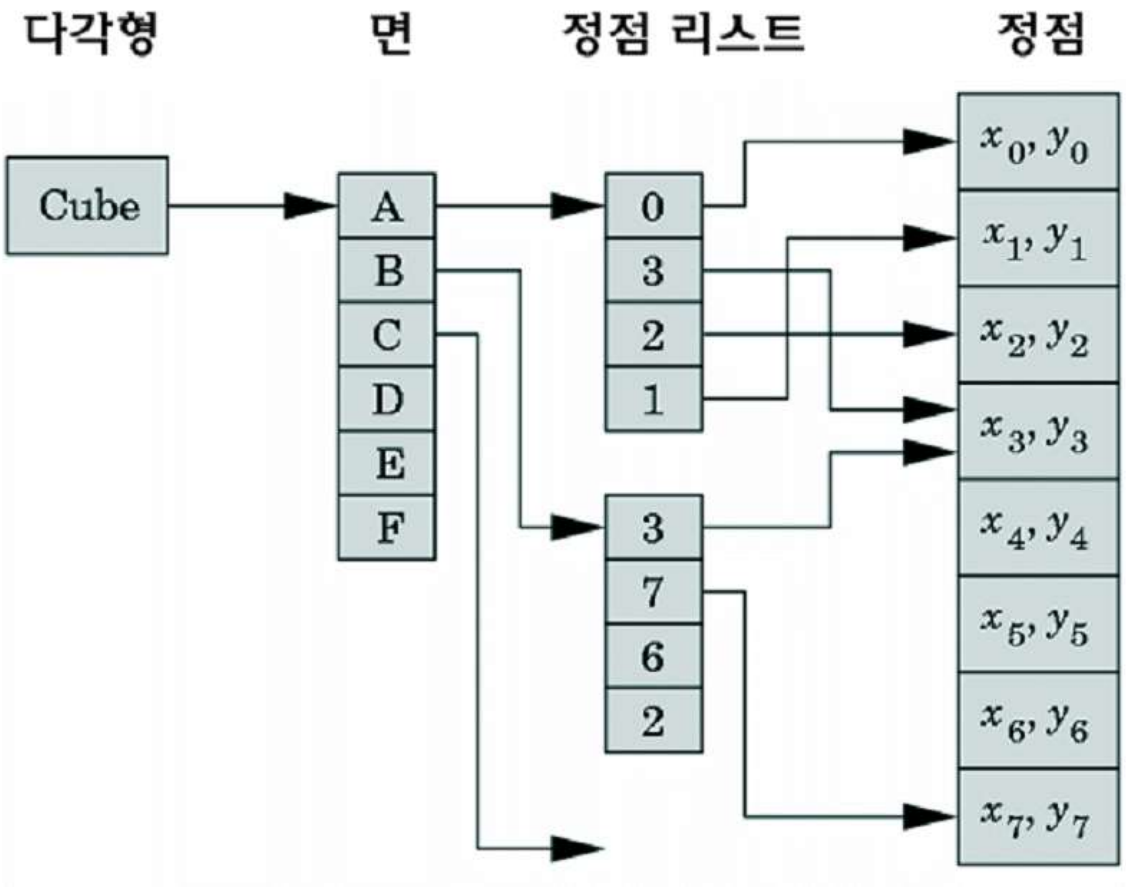
- GLfloat MyVertices[8][3] = {{-0.25, -0.25, 0.25}, {-0.25, 0.25, 0.25}, {0.25, 0.25, 0.25}, {0.25, -0.25, 0.25}, {-0.25, -0.25, -0.25}, {-0.25, 0.25, -0.25}, {0.25, 0.25, -0.25}, {0.25, -0.25, -0.25}};
- GLfloat MyColors[8][3]={{0.2, 0.2, 0.2}, {1.0, 0.0, 0.0}, {1.0, 1.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}, {1.0, 0.0, 1.0}, {1.0, 1.0, 1.0}, {0.0, 1.0, 1.0}};
- 정점 0, 3, 2, 1으로 구성된 면 (반시계 방향으로 명시)
  - glBegin(GL_POLYGON);
    glColor3fv(MyColors[0]); glVertex3fv(MyVertices[0]);
    glColor3fv(MyColors[3]); glVertex3fv(MyVertices[3]);
    glColor3fv(MyColors[2]); glVertex3fv(MyVertices[2]);
    glColor3fv(MyColors[1]); glVertex3fv(MyVertices[1]);
    glEnd();

점 배열 위치(인덱스) 배열

float → int



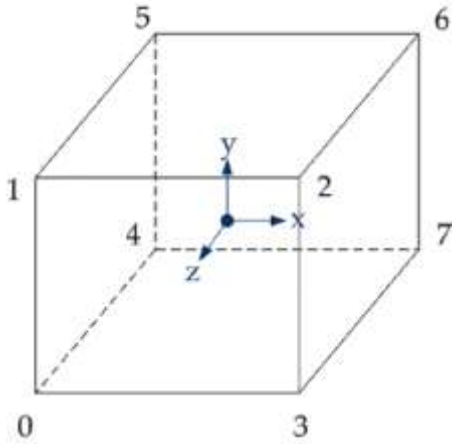
계층구조적 표현


다각형
면
정점 리스트
정점
Cube
A
B
C
D
E
F
0
3
2
1
3
7
6
2
x_0, y_0
x_1, y_1
x_2, y_2
x_3, y_3
x_4, y_4
x_5, y_5
x_6, y_6
x_7, y_7

정점 배열

```
GLfloat MyVertices[8][3] = {{-0.25,-0.25,0.25}, {-0.25,0.25,0.25},
    {0.25,0.25,0.25}, {0.25,-0.25,0.25}, {-0.25,-0.25,-0.25}, {-
    0.25,0.25,-0.25}, {0.25,0.25,-0.25}, {0.25,-0.25,-0.25}};
GLfloat MyColors[8][3]={{0.2,0.2,0.2}, {1.0,0.0,0.0}, {1.0, 1.0, 0.0},
    {0.0,1.0,0.0}, {0.0,0.0,1.0}, {1.0,0.0,1.0}, {1.0,1.0,1.0},
    {0.0,1.0,1.0}};
GLubyte MyVertexList[24]={0,3,2,1, 2,3,7,6, 0,4,7,3, 1,2,6,5, 4,5,6,7,
    0,1,5,4};
```

unsigned char




```
void MyDisplay( ){
    …
    glEnableClientState(GL_COLOR_ARRAY);
    glEnableClientState(GL_VERTEX_ARRAY);
    glColorPointer(3, GL_FLOAT, 0, MyColors);
    glVertexPointer(3, GL_FLOAT, 0, MyVertices);
    …
    for(GLint i = 0; i < 6; i++)
        glDrawElements(GL_POLYGON, 4, GL_UNSIGNED_BYTE,
                       &MyVertexList[4*i]);
    …
}
```

지엘의 실행모드

- 직접 모드(Immediate Mode)
  - 화면 렌더링과 동시에 물체 정보를 모두 파기
  - 다시 그리려면 전체 코드를 다시 실행

- 보류모드(Retained Mode)
  - 이미 정의된 물체를 컴파일 된 형태로 재사용
  - 디스플레이 리스트 기능을 이용하여 구현됨

- 디스플레이 리스트
  - 기본요소(Primitives), 상태변수(State Variable), 영상(Image)
  - 이동, 회전, 조명 작업과 관련된 모든 명령
  - 반복적으로 실행되어야 할 요소를 디스플레이 리스트 내부에 포함
  - 프로그램 속도 향상에 필수적임

Display List [5-14]

```
void MyCreateList() {
    MyListID = glGenLists(1);
    glNewList(MyListID, GL_COMPILE);
    glBegin(GL_POLYGON);
        glColor3f(0.5, 0.5, 0.5);
        glVertex3f(-0.5, -0.5, 0.0);        glVertex3f(0.5, -0.5, 0.0);
        glVertex3f(0.5, 0.5, 0.0);        glVertex3f(-0.5, 0.5, 0.0);
    glEnd( );
    glEndList(); // 명령 list를 한번 만들어 놓으면
}
void MyDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glViewport(0, 0, 300, 300);
    glCallList(MyListID); // 한번의 함수 호출로, 여러 명령 실행 가능
    glFlush();
}
```