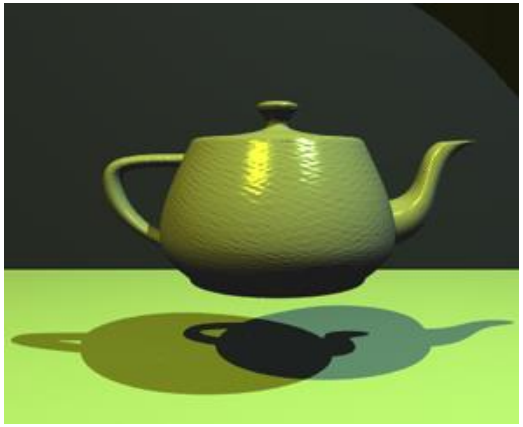
Computer Graphics

Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

What is computer graphics?

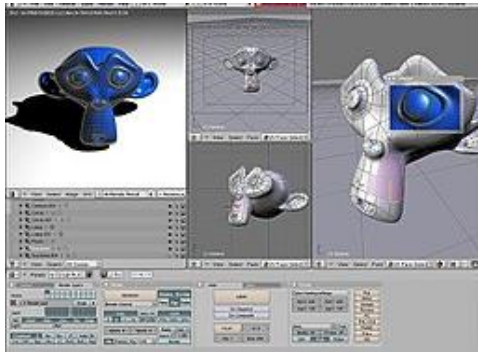


1. Use of computers to generate images

- “컴퓨터를 사용하여 그림을 생성하는 기술“
- “컴퓨터를 사용하여 물체를 그리는 기술”

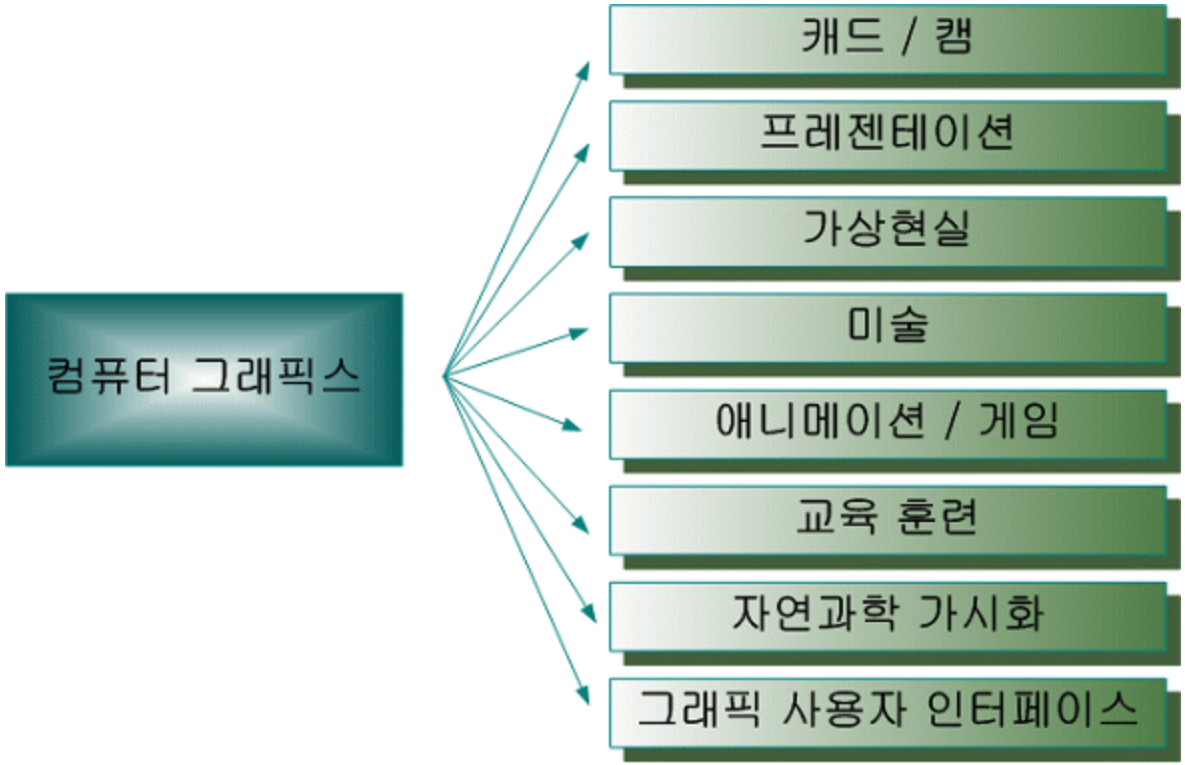
2. Pictures that are generated by a computer

3. Computer-generated image data created with help from specialized graphical hardware and software



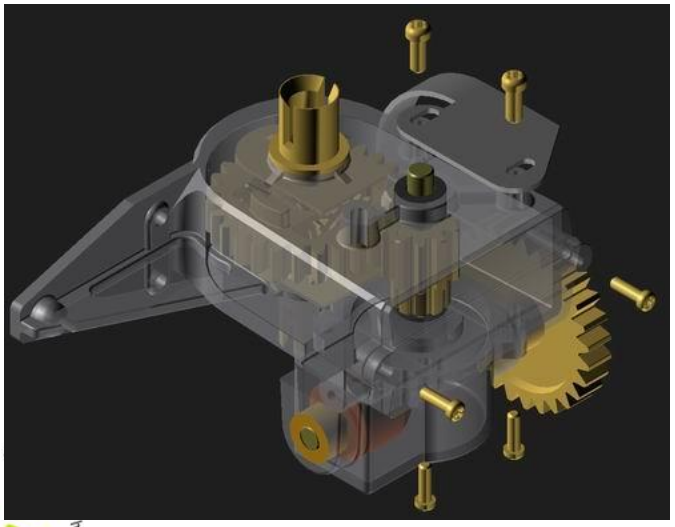
- 컴퓨터 그래픽스의 응용 분야

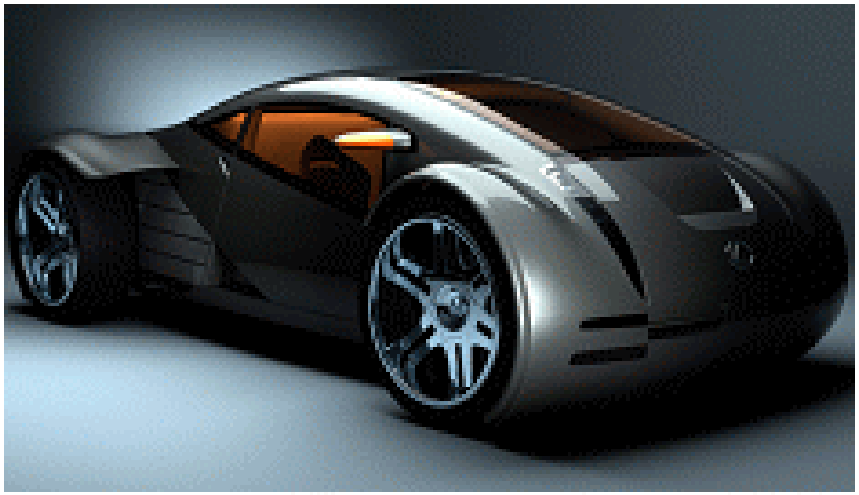
- 컴퓨터 그래픽스의 응용 분야



1. Computer-Aided Design (CAD)

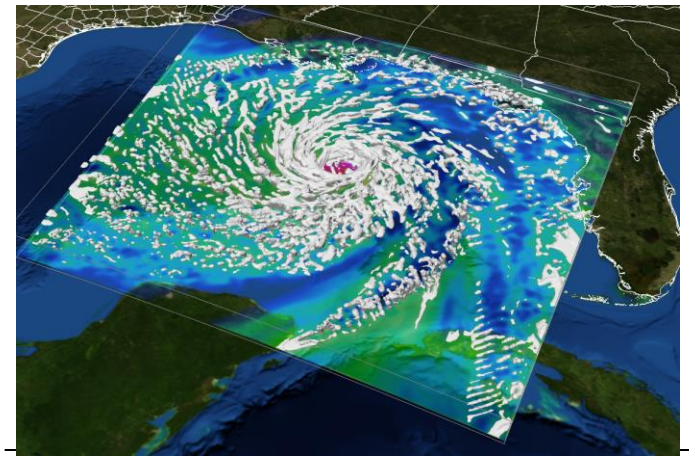
- **Interactive**한 컴퓨터 그래픽스 기술을 이용하여 설계에 필요한 인력,시간, 노력 등을 단축함으로써 설계 효율을 향상 (건축 설계, 제품 디자인, 산업디자인등)

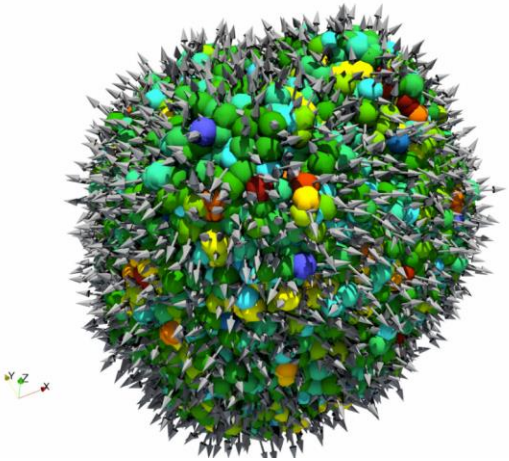




- **2.** 애니메이션 및 게임
- 컴퓨터 그래픽스 기술은 **2**차원 또는 **3**차원 애니메이션 및 게임을 제작하는데 사용된다
- 실제로 촬영된 영상과 컴퓨터 그래픽스 기술을 조합하여 현실감을 높이기도 한다

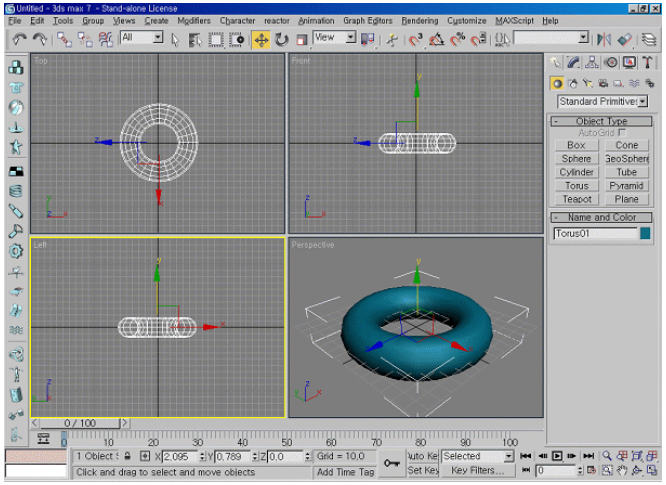
- **3. 과학분야 시각화 (scientific visualization)**
  - 과학 및 공학 분야에서 발생하는 데이터의 분석, 주로 대용량의 정보를 분석한다
  - 목적: 자연현상을 시각화 현상 내부의 패턴을 직관적으로 파악
  - 예: 허리케인 데이터 시각화, 의학 데이터 시각화 (암)
  - https://www.tacc.utexas.edu/scientific-visualization-gallery





- **4. GUI (Graphical user interface,** 그래픽 사용자 인터페이스)
- 사용자가 컴퓨터와 그래픽 아이콘, 메뉴 등을 통하여 **interact**하는 사용자 인터페이스

윈도우 GUI



3D 스튜디오 MAX (3DS Max)

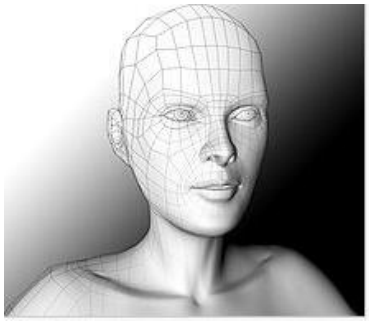
- 컴퓨터 그래픽스의 구성 요소

- 컴퓨터 그래픽스의 구성 요소
- 컴퓨터 그래픽스를 통해 어떠한 **image**를 생성하기 위해서는 두 단계의 과정이 있다

- **Modeling:** 무엇을 그릴 것인지에 관련된 것
- 그래픽으로 표현하고자 하는 장면, 혹은 물체를 정의 하는 작업
- 다각형 **vertex (**정점**)**의 위치, 연결등 정의
- **A vertex (plural vertices) in computer graphics is a data structure that describes certain attributes, like the position of a point in 2D or 3D space**

Modeling

(what to draw)

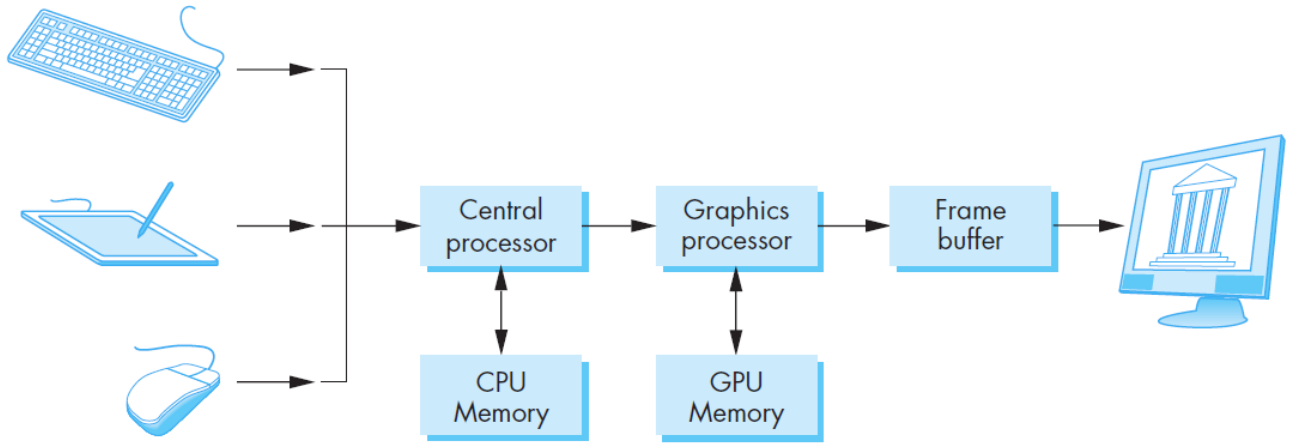


3D modeling of a human face

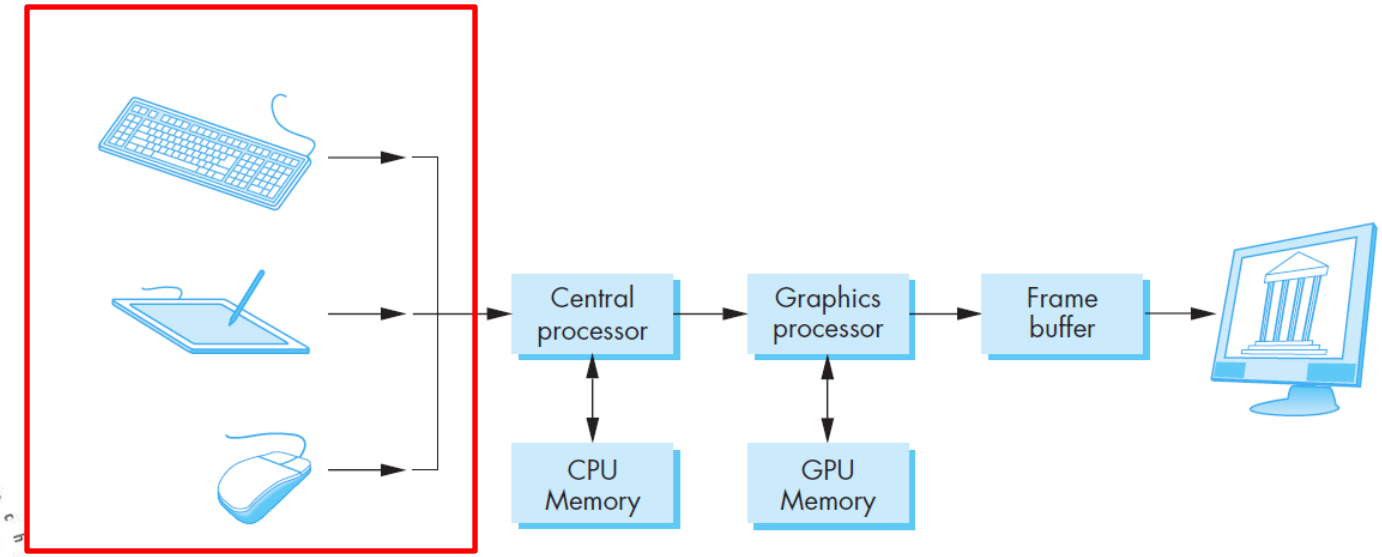
- **Rendering:** 모델링 된 물체를 그려내는 작업
- 조명 처리
- 카메라의 위치, 방향 설정
- **3**차원의 물체를 최종적으로 어떻게 **2**차원으로 사상 **(projection)**시키는지 결정
- 물체의 재질 **(texture)** 입히기

- 그래픽스 시스템 **(Graphics system)**

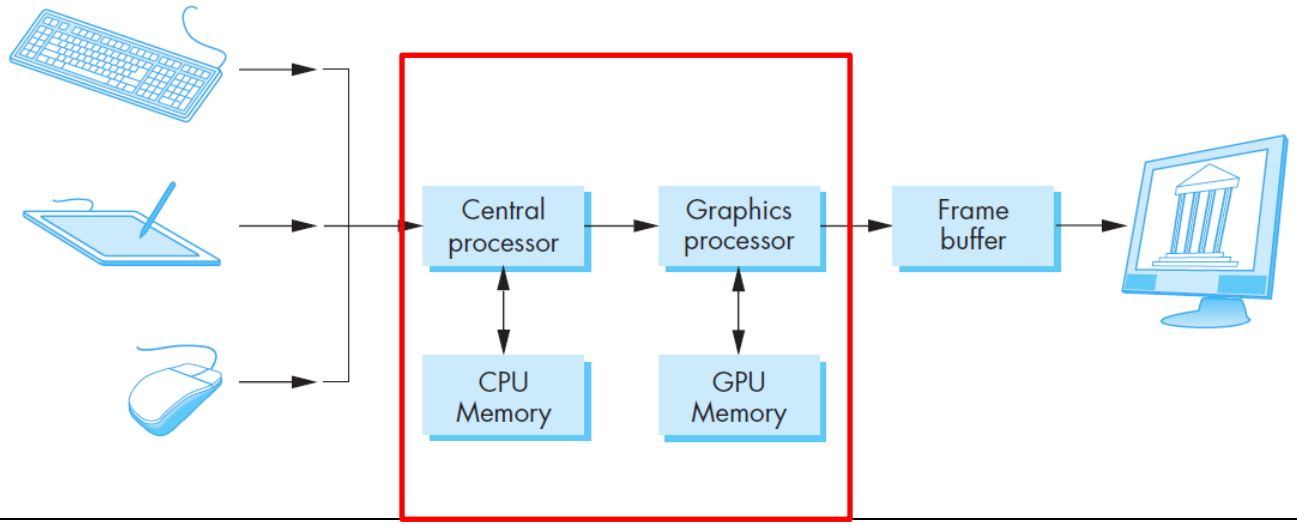
- 컴퓨터 그래픽스 시스템은 컴퓨터 시스템으로 일반적인 목적의 컴퓨터 시스템의 요소를 모두 갖추고 있다. 컴퓨터 그래픽스 시스템은 크게 보면
- 다음과 같은 구성 요소로 구성되어 있다
- **1. Input device 2. CPU (central processing unit)**
- **3. GPU (graphics processing unit)**
- **4. 메모리 5. Frame buffer 6. Output device**



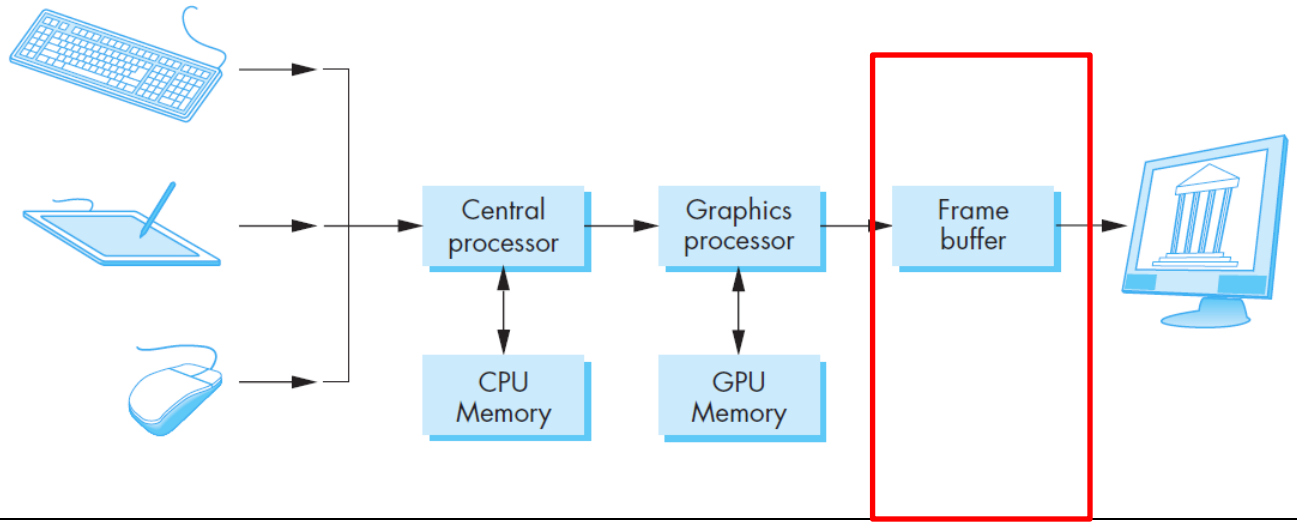
- **Input device (입력 장치)**
- 키보드, 마우스, 조이스틱, 테블릿등



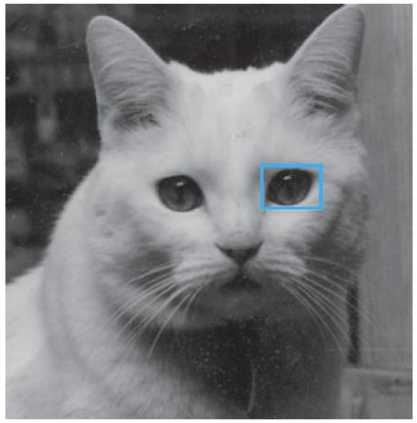
- **CPU and GPU**
- 간단한 시스템에서는 하나의 **CPU**만 존재 한다**.** 이 **CPU**는 연산과 그래픽 처리와 관련된 일을 수행한다
- 최신 그래픽스 시스템에서는 특별한 그래픽스 함수를 수행 **(**처리**)**하기 위하여 **GPU (graphics processing unit)**를 가지고 있으며 이는 메인 보드에 있거나 그래픽스 카드 형태로 있다

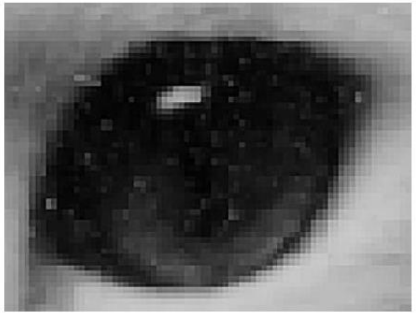


- **Frame buffer**
- 우리가 모니터와 같은 **output device**에서 최종적으로 보는 것은 **image**이다
- 이 **image**는 **pixel (픽셀, picture element)**로 이루어져 있다
- **Pixel**은 다른 말로 **raster**라고도 불린다



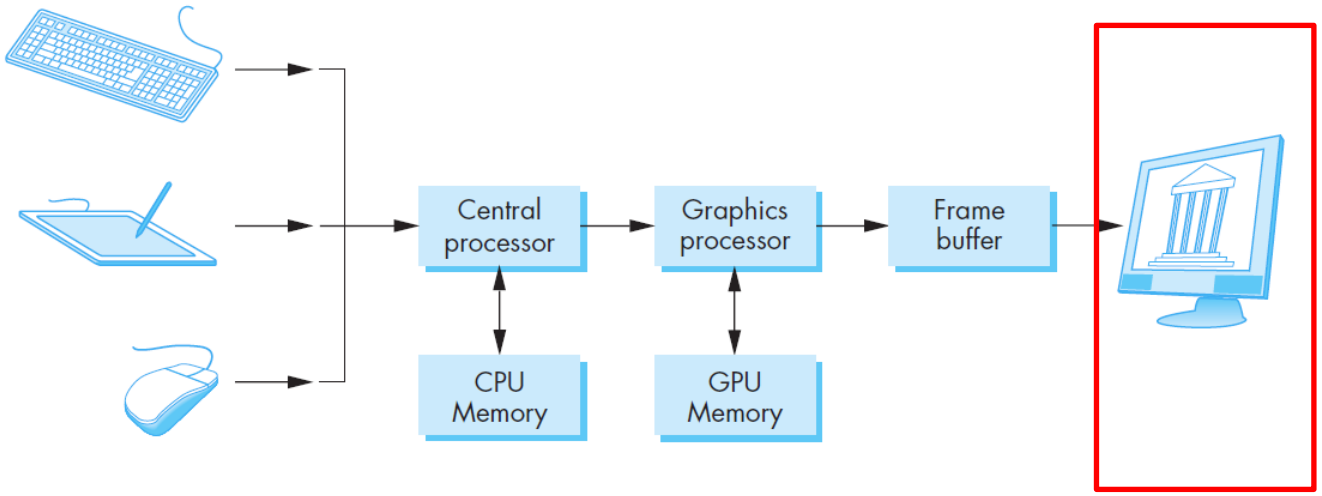
- 아래 그림과 같이 각각의 픽셀은 어떤 위치 **(**작은 영역을 갖는**)**에 대응한다
- 각 픽셀은 메모리에 **frame buffer**라는 곳에 저장 된다
- **Frame buffer**에 있는 **pixel** 수를 해상도 **(resolution)**이라고 한다
- 기본적으로 메모리의 **frame buffer**에는 각 **pixel**의 **color** 정보가 저장되지만 그 이외에 **depth (**깊이**)** 정보 등도 저장 된다
- 즉**, Frame buffer**는 여러 개의 **buffer**로 구성되어 이는데 그 중에 **color buffer**가 포함된다고 생각하면 된다





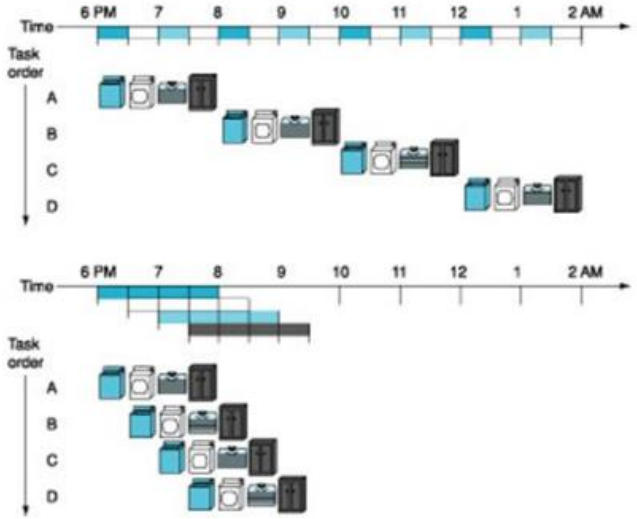
Sun TGX Framebuffer

- **Output device (출력 장치)**
- 주로 모니터를 의미함
- **LCD, CRT, OLED**등이 있음



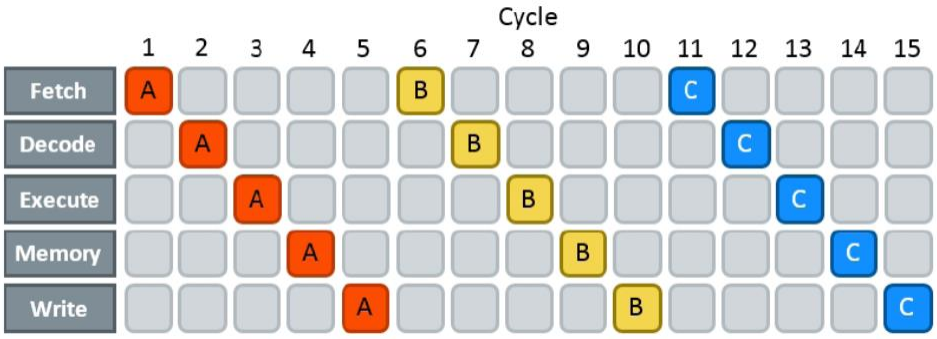
- **Graphics Pipeline**

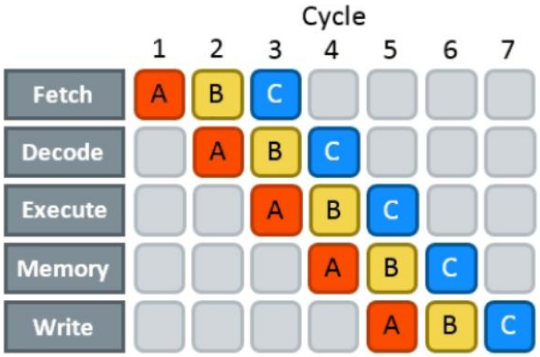
- **Pipelining**이란? 컴퓨터 구조에서 나온 개념
- 예: **Laundry example.** 아주 많은 빨래가 있어서 하나의 세탁기에 다 들어가지 않은 경우
- **4**가지 작업 (단계) 필요 **1. Washing 2. Drying 3. Folding 4. Storing**
- **Pipeline approach takes much less time (**병렬 구조를 이용**)**



The laundry analogy for pipelining

- **Instruction pipelining**
- **Pipelining is an implementation technique in which multiple instructions are overlapped in execution**
- **Pipeline approach takes much less time (병렬 구조를 이용)**
- 예: 하나의 **instruction**이 **5**개의 **step**을 거침. **3**개의 **instruction**을 수행 시
- **(왼쪽) without pipelining, 15 cycles (오른쪽) with pipelining, 7 cycles**



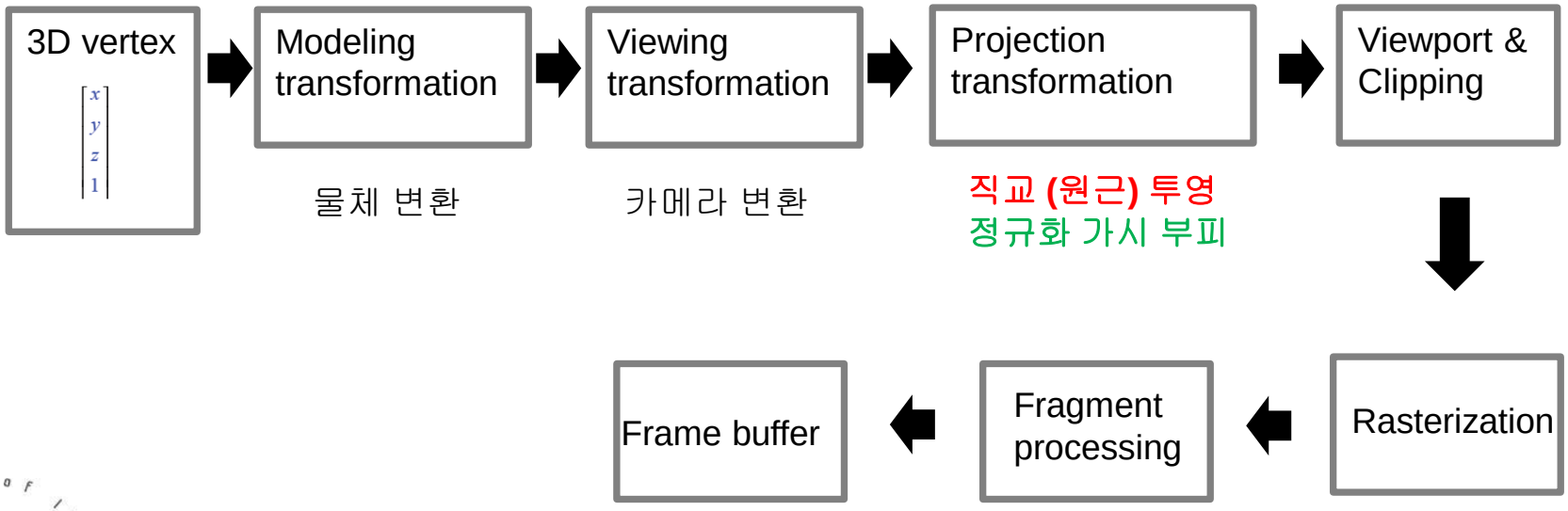


- **Graphics pipeline:** 컴퓨터 그래픽스 시스템에서도 기본적인 **vertex**로부터 최종적으로 모니터 **(**스크린**)**에 **pixel**로 보여지는데 까지 여러 개의 단계로 나뉘어져 있다

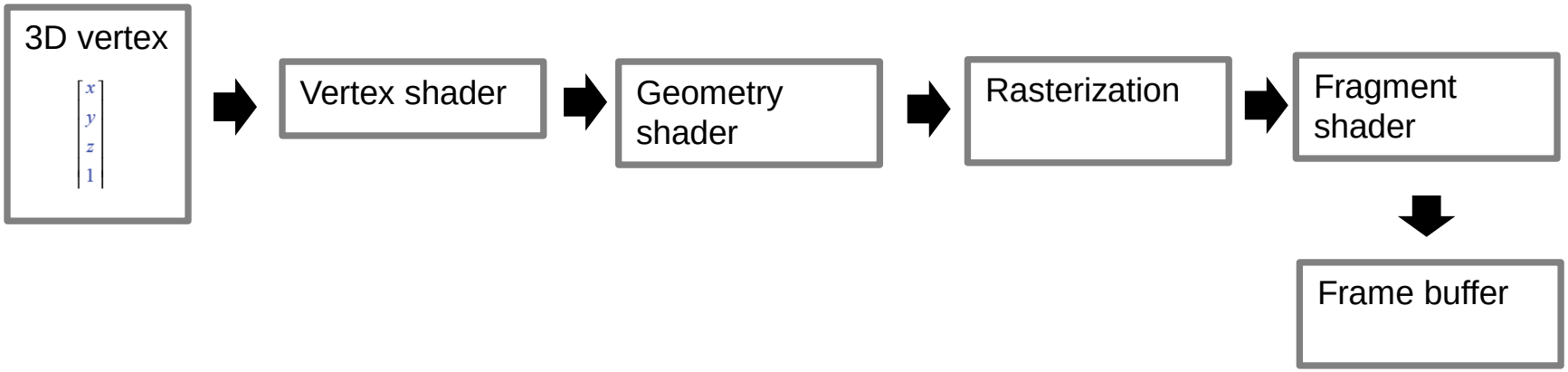
- 이 단계들은 순서대로 **(**순차적으로**)** 수행되며 한 단계의 결과는 바로 다음 단계로 넘어가며 다음 **vertex (polygon,** 다각형**)**가 바로 연달아 수행되도록 **pipeline** 형태로 **parallel**하게 병렬적으로 수행된다**. Why?**

- 일반적인 **Graphics pipeline**의 예
- **1. Vertex processing (vertex** 처리**)**
- **2. Clipping and primitive assembly**
- 예**: vertex 3**개로 이루어진 **triangle**
- **3. Rasterization (**픽셀로 이루어진 모니터**, raster display**에 **mapping**하기 위한 픽셀 선택**, color)**
- **4. Fragment processing**
- **Frame buffer**에 기록하기 바로 전 단계

- **OpenGL 버전, GPU 사용 여부에 따라 Graphics pipeline에 차이는 있음**
- **Fixed function pipeline (OpenGL 2.x 버전)**



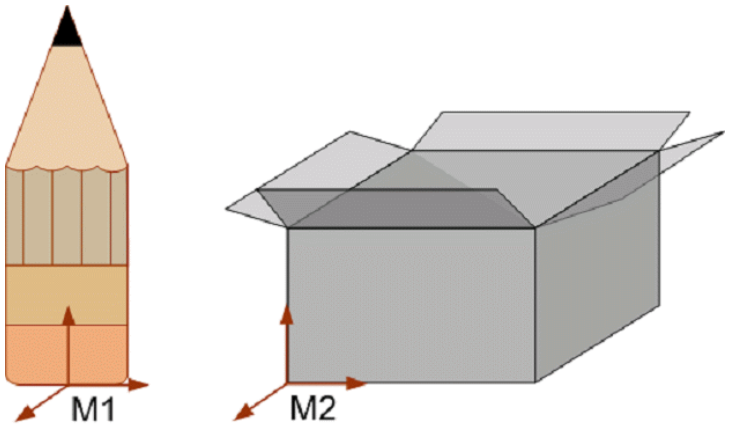
- **Shader** 기반의 최신 **OpenGL** 버전
- **GPU**를 사용하여 성능 향상 및 **rendering**

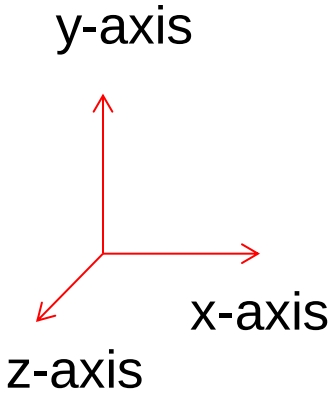


- 좌표계 (coordinate system)

- **Graphics pipeline** 각 **pipeline** 단계에서 여러 개의 서로 다른 좌표계 **(coordinate system)**가 사용된다
- 모델 좌표계, 전역 좌표계, 시점 좌표계, 절단 좌표계, 정규 좌표계, 화면 좌표계
- **Graphics pipeline** 내에서 순차적으로 작업이 수행됨에 따라서 여러 개의 다른 좌표계로 바뀌게 된다
- 많은 경우 **Black-box**와 같이 내부적으로 자동적으로 좌표계 변환이 이루어 져서 사용자가 변환할 필요는 없음

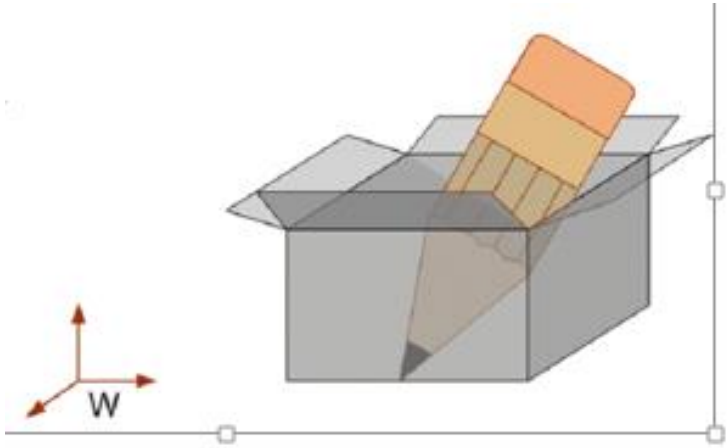
- **Local coordinate system (LCS, model coordinate system,** 모델 좌표계, 지역 좌표계**)**
- 모델 자표계**:** 각 물체별로 물체를 모델링하기 편하게 설정된 좌표계
- **Local coordinate system**은 물체마다 좌표계의 원점 **(origin: (0,0,0))**및 축 방향이 **(x, y, z axis)** 다르다



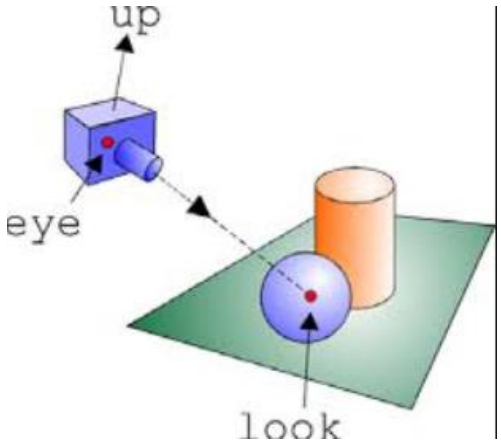


- 이렇게 물체 별로 좌표계를 각각 둔다면 각각의 물체를 모델링 하기에는 편하지만 물체가 여러 개 존재 한다면 불편할 수 있다.
- **What about use one unified coordinate system for all objects?**
- 즉, 여러 물체를 한꺼번에 아우를수 있는 좌표계가 전역 좌표계 **(world coordinate system)**이다

- **World coordinate system (**전역 좌표계**)**
- **Local coordinate system**의 불편을 해소하고 사용자의 편의를 위해 모든 물체를 한꺼번에 일률적으로 표현할 수 있는 가상의 **coordinate system**
- 사용자가 사용하는 좌표계로 **3D** 좌표계
- 임의로 원점 **(0,0,0)**을 설정, 단위 **(unit) (**예**: cm, m)**도 사용자가 설정



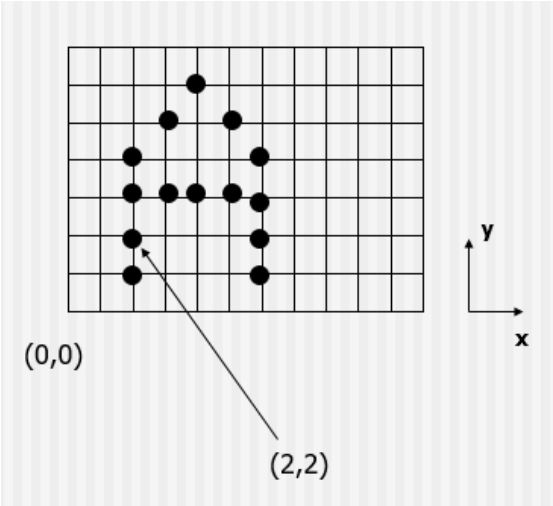
- 시점 좌표계 **(View coordinate system)**
- 물체를 보는 사람 (카메라)의 위치와 보는 방향에 따라서 물체 모습이 다르게 보인다
- 카메라 기준의 좌표계



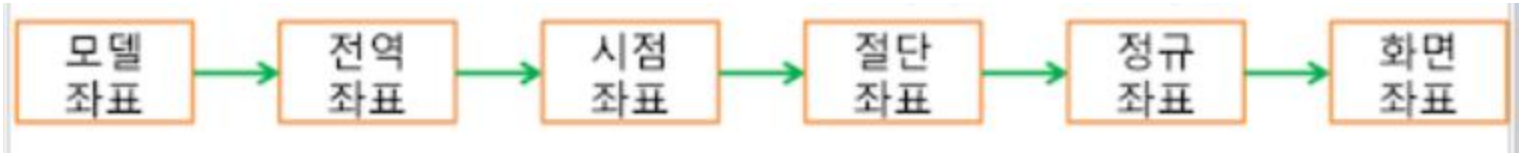
- 절단 좌표계: 물체를 보는 사람 (카메라) 입장에서 보이지 않는 물체를 잘라내기 편하게 하기 위해 만든 좌표계

- 정규 좌표계 (NDC, normalized device coordinates): 화면 좌표계로 변환 전에 정규화위해 사용하는 좌표계

- **Screen (window) coordinate system: 화면 좌표계**
- 최종적으로 컴퓨터 스크린 **(**모니터**)**에서 보이는데 필요한 좌표계로 픽셀 단위로 표현하며 **2D**임
- 예**:** 왼쪽 좌측이 원점 **(0, 0),** 각 픽셀의 좌하단 기준



- **OpenGL**에서의 좌표계 변환 순서
- 실제 이 좌표계 변환은 **OpenGL** 내부에서 행렬 **(4행4열 행렬, 행렬 변환)**을 통해서 이루어진다



- **OpenGL**

- **OpenGL: OpenGL : Open graphics library**
- **OpenGL**이란 실리콘 그래픽스 **(SGI)**사가 개발한 **3**차원 컴퓨터 그래픽스 **API (1992)**
- **OpenGL** 그래픽스 함수는 프로그래밍 언어에 독립적인 기능으로 지정되어 있어서 **C/C++/Java**등 다수 언어와 사용 가능
- **OpenGL**은 **cross-platform**해서 **OS**에 독립적인 **API** 이다
- **OpenGL**은 프로그래밍 언어가 아니다 **(**수백 개의 함수 및 명령어로 이루어짐**)**

- **OpenGL**의 발전 과정
- 현재 **OpenGL 4.6** 버전 **release**
- https://www.opengl.org/
- 최근 버전에는 **OpenGL shading language(GLSL)** 추가
- 본 수업에서는 이전 버전의 **OpenGL**인 **pre-shader OpenGL** 기준의 문법으로 실습할 예정임 **(OpenGL fixed-function pipeline** 기준**)**
- **Why use old version of OpenGL?**
- 본 수업의 목적은 **OpenGL**을 배우는 것이 목적이 아니라 그래픽스 이론 및 알고리즘을 배우는 것이 목적으로 처음 접하는 학생들에게는 이전 버전의 **OpenGL**이 더 이해하기가 쉽다

- **OpenGL libraries**
- **OpenGL core library (GL library):** 렌더링 기능을 제공하는 함수 라이브러리. 함수 이름이 **'gl'**로 시작
- **OpenGL utility library (GLU library): OpenGL** 코어 라이브러리를 추가로 지원하는 함수들의 모음
- **OpenGL utility toolkit (GLUT library):** 사용자 입력을 받아 들이거나 화면 윈도우를 제어하기 위한 함수
- **OpenGL Extension Wrangler (GLEW)**
- 이중 **GLUT**은 윈도우 운영체제와 **OpenGL** 프로그램 사이의 인터페이스 역할을 한다
- **GLUT library**가 오래되었기 때문에 최근 **freeglut**이라는 **GLUT**이 업데이트 된 라이브러리도 있음

- **Event-driven 프로그래밍과 callback 함수**

- 대부분의 **window** 기반의 프로그램은 **event driven** 방식이다
- 즉, 프로그램이 마우스 클릭이나 키보드 누름 등과 같은 **event**에 반응한다는 의미이다
- 시스템은 **event queue**를 운영하여 각각의 **event**를 대응하며 기본적으로 **First-come-First-serve (FCFS)** 방식으로 이 **event queue**를 운영한다
- 프로그래머는 프로그램 작성시 **event**가 발생시 수행되는 **callback** 함수의 모음으로 프로그램을 구성하며 어떠한 **event**가 생겼을 때 그에 해당되는 **callback 함수가 호출**된다.
- 기본적으로 **event** 발생 전에는 아무것도 수행하지 않으며 시스템은 **event** 발생 전까지 대기한다. 이를 **event loop**라고 한다

- 프로그래머는 각각의 **event**에 해당하는 **callback 함수를 등록 (register)**해야 한다
- 예: **glutMouseFunc(myMouse);**
- **GLUT** 라이브러리를 사용
- **mouse event**가 생기면 **callback** 함수 **“myMouse”**를 호출해라
- 단, **“myMouse”**는 프로그래머가 선택한 **callback** 함수 이름임

- **OpenGL** 프로그램은 윈도우 기능과 입출력 제어에 있어서 **GLUT** 라이브러리를 사용할 수 있다
- 프로그래머가 필요한 **callback** 함수를 등록 **(register)**하고 해당 **callback** 함수에 원하는 내용을 넣기만 하면, 이에 대한 호출은 **GLUT**에서 알아서 처리한다
- 이를 위해 **GLUT**은 이벤트 타입 별로 불러야 할 **callback** 함수를 아래와 같이 **callback table** 형태로 저장

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

- **Common events in OpenGL**

이벤트 타입	콜백함수	**Main** 함수
DISPLAY	void MyDisplay()	glutDisplayFunc(MyDisplay)
MOUSE	void MyMouse()	glutMouseFunc(MyMouse)
KEYBOARD	void MyKeyboard()	glutKeyboardFunc(MyKeyboard)
Reshape	void MyReshape()	glutReshapeFunc(MyReshape)

- **Display event:** 렌더링을 위한 이벤트로 대부분의 렌더링 명령은 대부분 디스플레이 **callback** 함수 내부에 정의 된다
- **Mouse (keyboard) event:** 마우스 (키보드) 버튼을 누르거나 떼었을 때 발생
- **Reshape event:** 사용자가 윈도우 크기를 바꿀 때 발생

- **First OpenGL code**

- 파일→새로만들기 →프로젝트 →Visual C++의 빈 프로젝트 → 확인
- 소스파일 → 마우스 오른쪽 클릭 → 추가 (새항목) → 파일이름.cpp

```
#include <GL/glut.h>
void myInit(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
glColor3f(0.0f, 0.0f, 0.0f);           // set the drawing color to black
glPointSize(4.0);            //set the point size to 4 by 4 pixels
glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
glLoadIdentity();// to be explained
gluOrtho2D(0.0, 640.0, 0.0, 480.0);// to be explained
}
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
glBegin(GL_POINTS);
glVertex2i(100, 50);        // draw some points (don't know how many)
glVertex2i(100, 130);
glVertex2i(150, 130);
glEnd();
glFlush();                // send all output to display
}
void main(int argc, char **argv)
{
glutInitWindowSize(640, 480);     // set the window size
glutInitWindowPosition(100, 150); // set the window position on the screen
glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
glutDisplayFunc(myDisplay);     // register the redraw function
myInit();
glutMainLoop();      // go into a perpetual loop
```

- **main 함수**

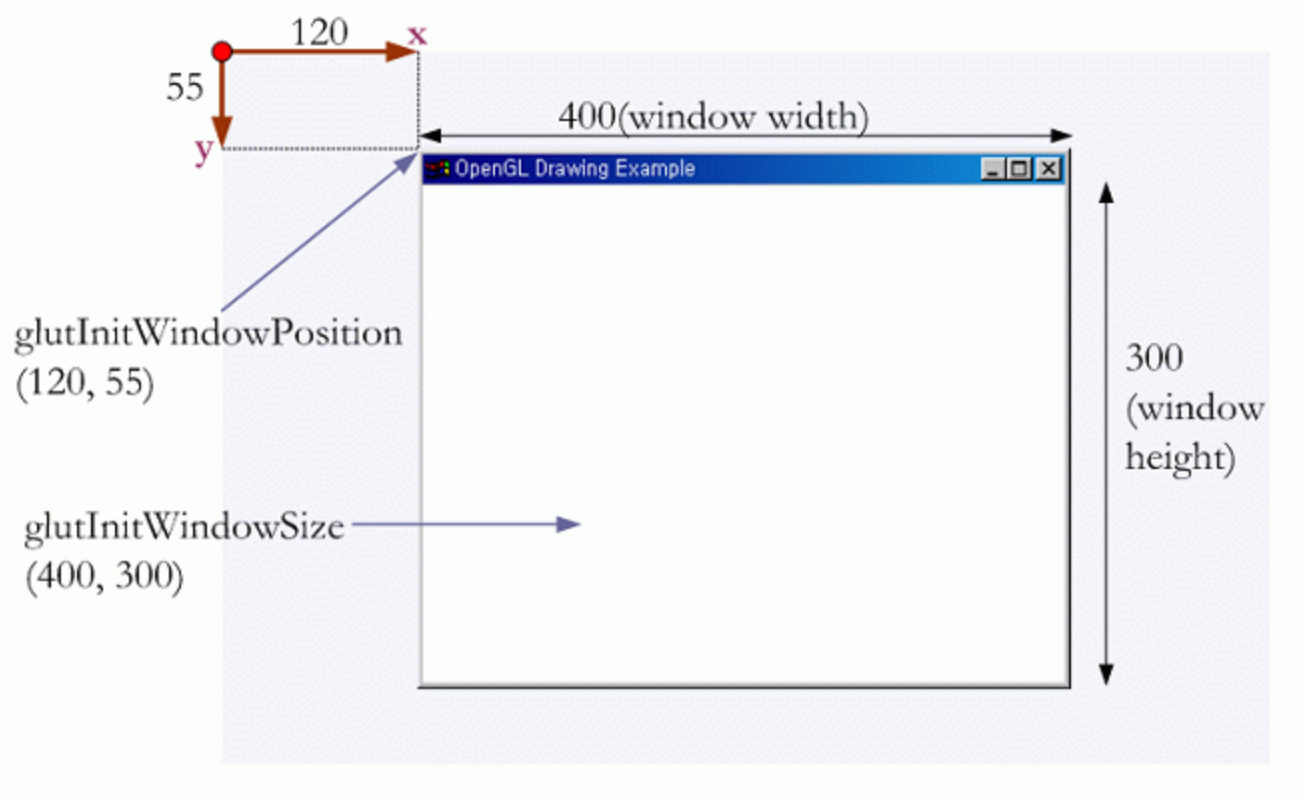
1. glutInitWindowSize(640,480);

결과가 화면에 출력되는 **window (screen window)**가 가로 **640 pixel** , 세로 **480 pixel** 로 되게 설정 한다

2. glutInitWindowPosition(100, 150);

결과가 화면에 출력되는 **window**의 왼쪽 위 시작점의 위치가 스크린에서 왼쪽에서 **100pixel,** 위에서 **150pixel**이다

- **glutInitWindowSize(400,300);**
- **glutInitWindowPosition(120, 55);**



- **3. glutCreateWindow("my first attempt");**
- 새로운 윈도우를 생성하고 **“my first attempt”**라는 문자열이 윈도우 상단에 나타나도록 한다

- **4. glutDisplayFunc(myDisplay);**
- **myDisplay** 라는 함수를 **display** 이벤트에 대한 콜백 함수 **(callback function)**으로 등록 **(register)**해라
- **This is called registering the callback function**

main 함수의 나머지 **line**들

- **5. myInit();**
- 또다른 초기화 과정

- **6. glutMainLoop();**
- **OpenGL** 프로그램 작성시 **main** 함수의 마지막에 오는 함수로 이벤트 처리 루프

- **myDisplay () 함수 (콜백 함수)**

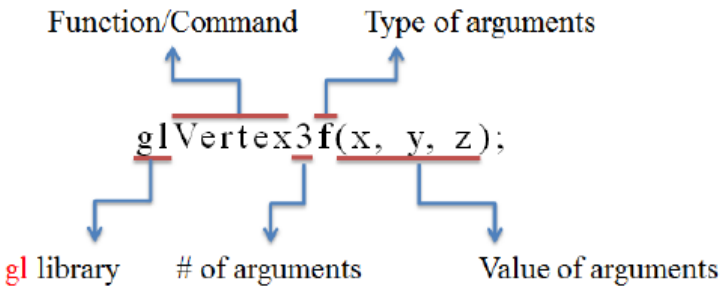
```
#include <GL/glut.h>
void myInit(void)
{
  glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
   glColor3f(0.0f, 0.0f, 0.0f);          // set the drawing color to black
   glPointSize(4.0);                                          //set the point size to 4 by 4 pixels
   glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
   glLoadIdentity();// to be explained
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);// to be explained
}
void myDisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
        glBegin(GL_POINTS);
                glVertex2i(100, 50);        // draw some points (don't know how many)
                glVertex2i(100, 130);
                glVertex2i(150, 130);
        glEnd();
        glFlush();                                    // send all output to display
}
void main(int argc, char **argv)
{
        glutInitWindowSize(640,480);     // set the window size
        glutInitWindowPosition(100, 150); // set the window position on the screen
        glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
        glutDisplayFunc(myDisplay);     // register the redraw function
        myInit();
        glutMainLoop();                          // go into a perpetual loop
```

- **1. glClear(GL_COLOR_BUFFER_BIT);**
- **window**안의 각 **pixel**의 **color** 버퍼 비우기 **(reset)**

- **2.** **glBegin(GL_POINTS);** GL_POINTS: Geometric Primitives
- **glVertex2i(100, 50);**
- **glVertex2i(100, 130);**
- **glVertex2i(150, 130);**
- **glEnd();**

- 컴퓨터 그래픽스에 **vertex (정점)**란?
- 복수: **vertices**
- 어떠한 점의 위치, 색상등과 같은 정보를 **2D**나 **3D**로 묘사하고 있는 자료 구조
- https://en.wikipedia.org/wiki/Vertex_(computer_graphics)

OpenGL 함수의 기본적인 구조



접미사	데이터 타입	C/C++ 타입명	GL 타입명
f	32-bit floating point	float	GLfloat
i	32-bit integer	int	GLint
d	64bit floating point	double	GLdouble
…	..	…	…

- **Flush ? Meaning**
- **glFlush();**
- **Graphic** 카드 드라이버에서는 **GL** 명령어 하나하나를 받는 즉시 실행하지 않고 일정 분량의 명령어를 쌓아두었다가 한번에 실행하는 명령
- **Use when all functions related with rendering are defined**

- **myInit() 함수**

```
#include <GL/glut.h>
void myInit(void)
{
  glClearColor(1.0, 1.0, 1.0, 0.0);      // set the bg color to a bright white
   glColor3f(0.0f, 0.0f, 0.0f);          // set the drawing color to black
   glPointSize(4.0);                                          //set the point size to 4 by 4 pixels
   glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
   glLoadIdentity();// to be explained
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);// to be explained
}
void myDisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
        glBegin(GL_POINTS);
                glVertex2i(100, 50);        // draw some points (don't know how many)
                glVertex2i(100, 130);
                glVertex2i(150, 130);
        glEnd();
        glFlush();                                    // send all output to display
}
void main(int argc, char **argv)
{
        glutInitWindowSize(640,480);     // set the window size
        glutInitWindowPosition(100, 150); // set the window position on the screen
        glutCreateWindow("my first attempt"); // open the screen window(with its exciting title)
        glutDisplayFunc(myDisplay);     // register the redraw function
        myInit();
        glutMainLoop();                          // go into a perpetual loop
```

1. glClearColor(1.0, 1.0, 1.0, 0.0);

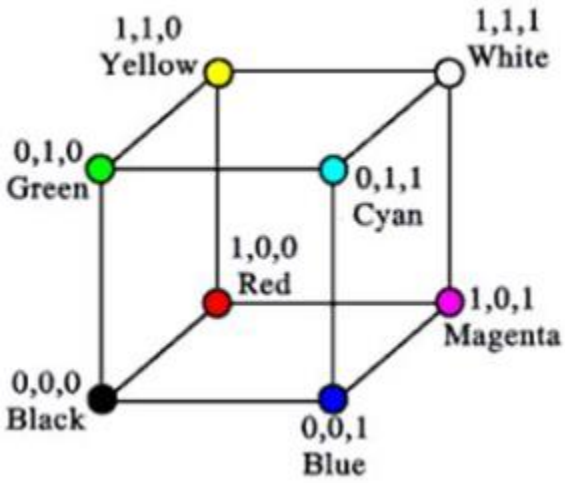
배경색 정하기

glClearColor(red, green, blue, alpha)

red, green, blue 값은 **0.0** 에서 **1.0** 사이의 값을 설정

alpha값: 투명도 (불투명도) , 현재는 그냥 **0.0**으로 둔다

- **Black: (0.0, 0.0, 0.0, 0.0);**
- **Red: (1.0, 0.0, 0.0, 0.0);**
- **Green: (0.0, 1.0, 0.0, 0.0);**
- **Yellow : (1.0, 1.0, 0.0, 0.0);**
- **White: (1.0, 1.0, 1.0, 0.0);**



- **2. glColor3f(0.0f, 0.0f, 0.0f);**
- **glrColor3f(red, green, blue)**
- **red, green, blue** 값은 **0.0** 에서 **1.0** 사이의 값을 설정

- **3. glPointSize(4.0);**
- 점 크기를 **4x4 pixel**로 정한다

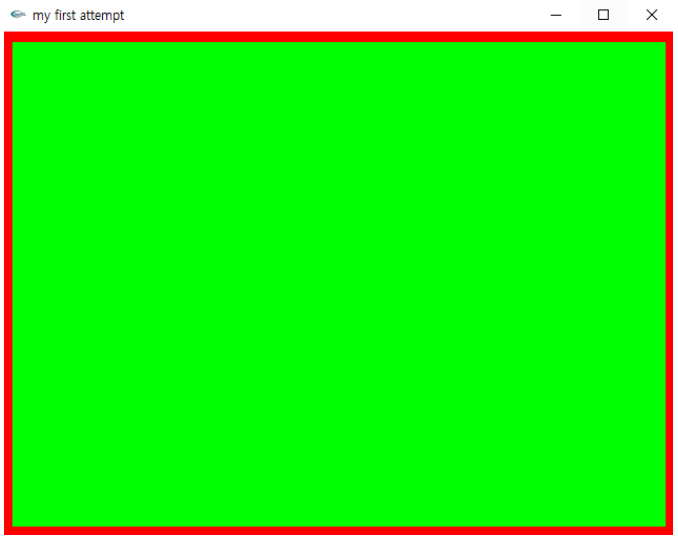
- **4. glMatrixMode(GL_PROJECTION);**
- **glLoadIdentity();**
- **gluOrtho2D(0.0, 640.0, 0.0, 480.0);**

- **=>** 가시공간 **(**가시부피**)** 설정 및 투영 설정

- 가시 공간 설정 **(viewing box 설정)**

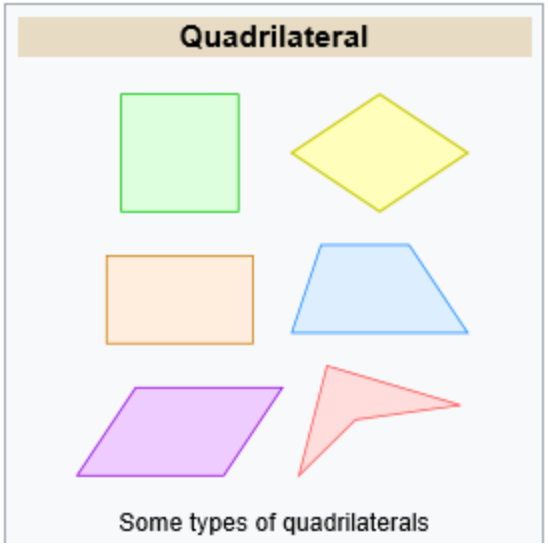
- 가시 공간 **(viewing box,** 가시부피)설정을 이해하기 위하여 다음 예제를 실행하여 보자
- 예에서 물체 위치는 전역 좌표계를 사용하여 설정 한다
- 앞의 코드와 달라진 부분 (빨간색 부분)을 확인해보자

```
#include <GL/glut.h>
 void myInit(void)
 {
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f(0.0f, 1.0f, 0.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
 }
 void myDisplay(void)
 {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);
    glVertex3f(10.0,10.0,0.0);
    glVertex3f(10.0,470,0.0);
    glVertex3f(630.0, 470.0, 0.0);
    glVertex3f(630.0, 10.0, 0.0);
    glEnd();
   glFlush();
 }
 void main(int argc, char **argv)
 {
            glutInitWindowSize(640,480);
            glutInitWindowPosition(100, 150);
            glutCreateWindow("my first attempt");
            glutDisplayFunc(myDisplay);
            myInit();
            glutMainLoop();
```

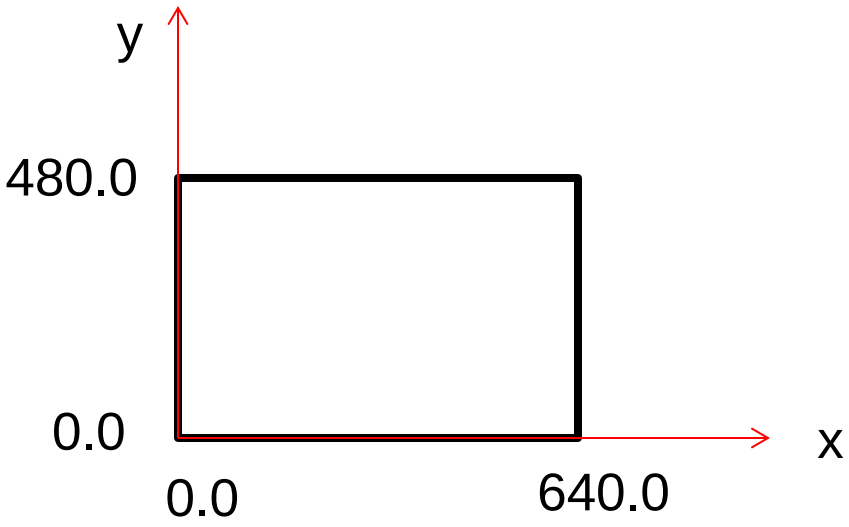


```
glBegin(GL_QUADS);
    glVertex3f(10.0,10.0,0.0);
    glVertex3f(10.0,470,0.0);
    glVertex3f(630.0, 470.0, 0.0);
    glVertex3f(630.0, 10.0, 0.0);
glEnd();
```

- **Quadrilateral (QUAD) ?**
- **A polygon with four edges and four vertices**
- **https://en.wikipedia.org/wiki/Quadrilateral**



- **Viewing box:** 가상의 가시 공간 설정 **(2D, 3D 모두 가능)**
- **gluOrtho2D(0.0, 640.0, 0.0, 480.0); // 2D**로 직사각형 가시 공간 설정
- **gluOrtho2D(left, right, bottom, top); //** 직사각형 형태로 설정
- 가시 공간에서 **(**왼쪽끝**,** 오른쪽끝**,** 아래쪽끝**,** 위쪽 끝**)** 설정
- 이 **viewing box** 안에 어떠한 물체가 위치해야 **window** 창에 보인다



- **Q)** 앞에서 **window**의 크기를
- **glutInitWindowSize(640,480);**
- 를 이용해 **640 pixel x 480 pixel**로 설정하였다
- 왜 앞에서 가시 공간을 설정시 세계 좌표계로
- **gluOrtho2D(0.0, 640.0, 0.0, 480.0);**
- 가로 **640,** 세로 **480** 으로 똑같이 맞췄을까**?**

- **Q)** 만일 이렇게 똑같이 맞추지 않고 가시 공간의 크기를 다르게 한다면 어떤일이 생길까**?**

- 현재는 **2**차원 직사각형 박스 형태로 가시공간을 설정하지만
- 후에는 **3**차원으로 **CUBE** 형태로 가시 공간을 설정할 수도 있다**.** 이를 가시 부피 **(**혹은 **viewing box)**라고 한다
- 가시 공간과 가시 부피는 모두 가상의 공간이다

- 코드에 보면
- **gluOrtho2D**를 사용하기 전에 보면 코드에 다음 두 줄이 있다.
- **glMatrixMode(GL_PROJECTION);**
- **glLoadIdentity();**
- 일단 이 부분은 후에 투영을 사용하는 행렬을 초기화 한다라고 생각하고 후에 자세하게 배운다

- **1.** 앞의 코드에서 **gluOrtho2D** 부분을
- **gluOrtho2D(10.0, 630.0, 10.0, 470.0);**
- 으로 바꾸었을때의 결과를 예상해 보자**.** 왜 그렇게 보이는가**?**

- **2.** 이번엔 **1**에서 사용한 **gluOrtho2D**를 그대로 사용하고

아래와 같이 바꾸어 보고 결과를 예상해 보자**.** 왜 그렇게 보이는가**?**

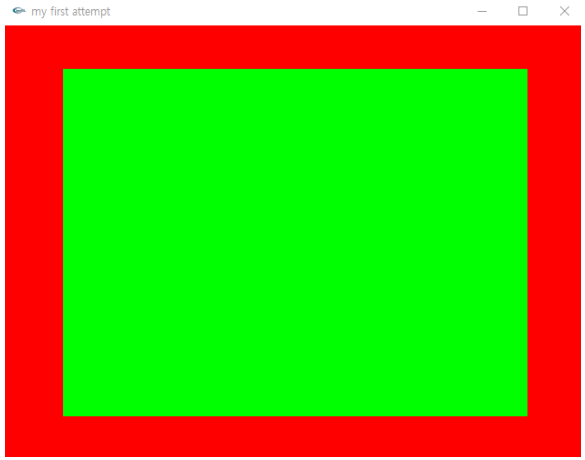
- **glBegin(GL_QUADS);**
- **glVertex3f(20.0,20.0,0.0);**
- **glVertex3f(20.0,30,0.0);**
- **glVertex3f(30.0, 30.0, 0.0);**
- **glVertex3f(30.0, 20.0, 0.0);**
- **glEnd();**

- **Q) OpenGL**에서의 기본적인 가시 공간은 무엇일까**?**

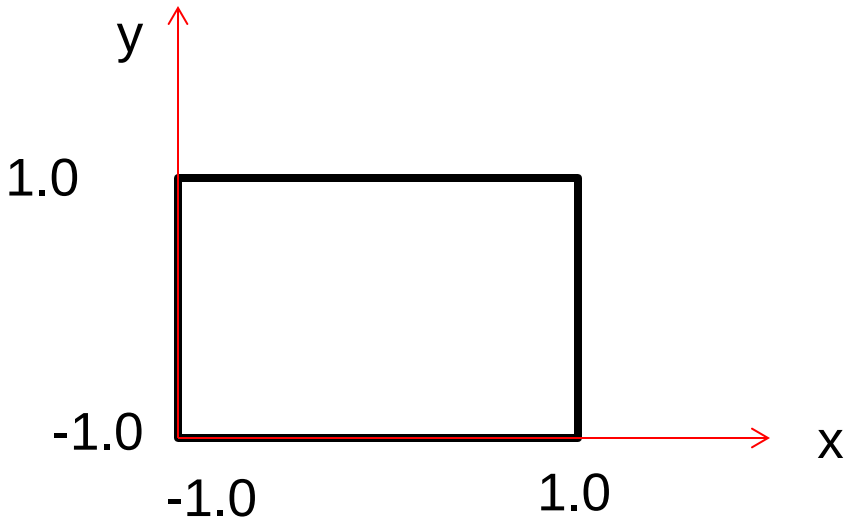
```
#include <GL/glut.h>

 void myInit(void)
 {
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f(0.0f, 1.0f, 0.0f);

 }
 void myDisplay(void)
 {
  glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);
    glVertex3f(20.0,20.0,0.0);
     glVertex3f(20.0,30,0.0);
     glVertex3f(30.0, 30.0, 0.0);
     glVertex3f(30.0, 20.0, 0.0);
     glEnd();
     glFlush();
 }

 void main(int argc, char **argv)
 {
            glutInitWindowSize(640,480);
            glutInitWindowPosition(100, 150);
            glutCreateWindow("my first attempt");
            glutDisplayFunc(myDisplay);
            myInit();
            glutMainLoop();
```

- 아무것도 안보인다. 즉, 위에서 그린 물체는 가시 공간 밖에 있다는 것이다
- 아래와 같이 바꾸어 실행해 보자. 왜 그렇게 보일까?

```
glBegin(GL_QUADS);
glVertex3f(-0.8, -0.8, 0.0);
glVertex3f(-0.8, 0.8, 0.0);
glVertex3f(0.8, 0.8, 0.0);
glVertex3f(0.8, -0.8, 0.0);
glEnd()
```



- **OpenGL** 에서의 기본 가시 공간은 **x: -1~1, y:-1,1** 이다
- 즉, 물체가 아래와 같은 직사각형 모양의 가시 공간 안에 있어야지만 물체가 보인다



```
void myInit(void)
{
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f(0.0f, 1.0f, 0.0f);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();  // projection matrix 초기화
        gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}
```

- 앞의 **graphics pipeline**에서 여러 좌표계가 사용되며 각각의 좌표계로의 변환이 이루어진다는 것을 배웠다
- **OpenGL**에서는 좌표계 변환에서 **행렬 (4x4)**을 사용한다
- 이전 버전의 **OpenGL (pre-shader)**에서는 **projection**시 사용되는 **projection 행렬**과 변환에 사용되는 **modelview 행렬**이 있다
- 이 부분은 이 두 행렬들을 어떻게 초기화 하고 어떻게 설정할지 정하는 부분으로 후에 설명될 예정이다