Computer Graphics

Prof. Jibum Kim

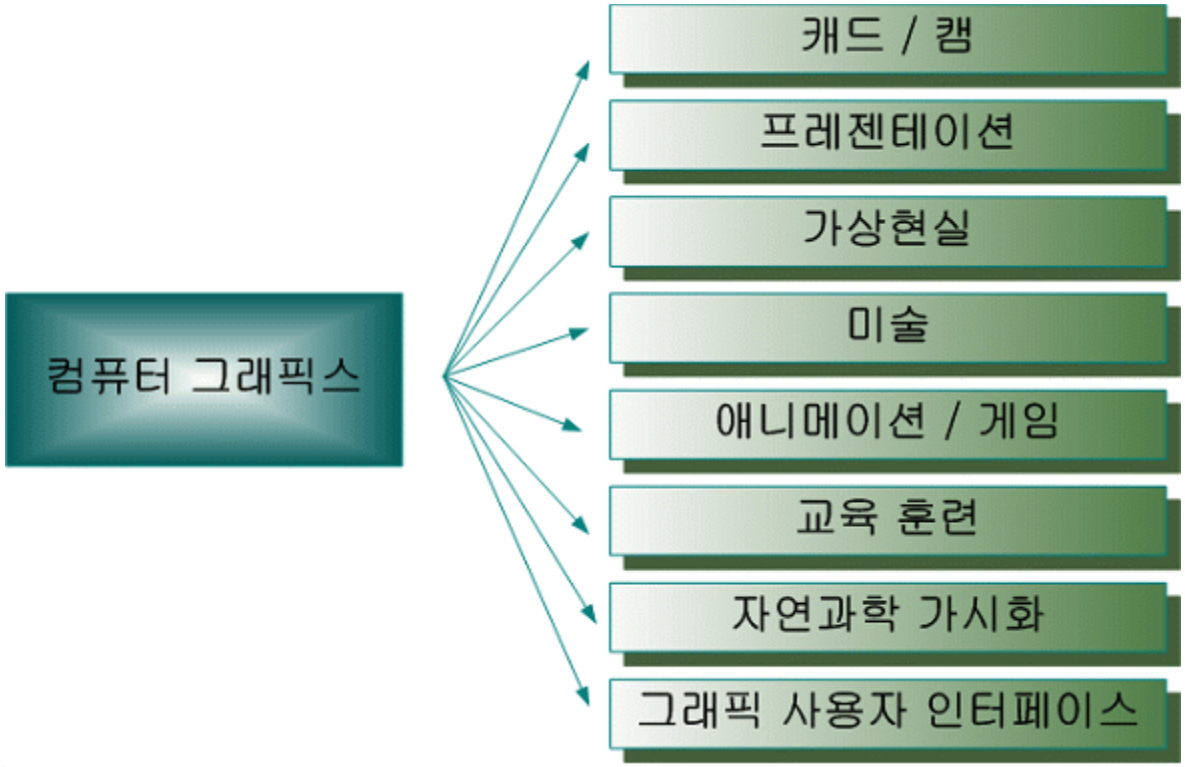
Department of Computer Science & Engineering

Incheon National University

- **Lecture 1: 컴퓨터 그래픽스의 구성 요소, 그래픽스 시스템, 그래픽스 pipeline, first OpenGL 프로그램**

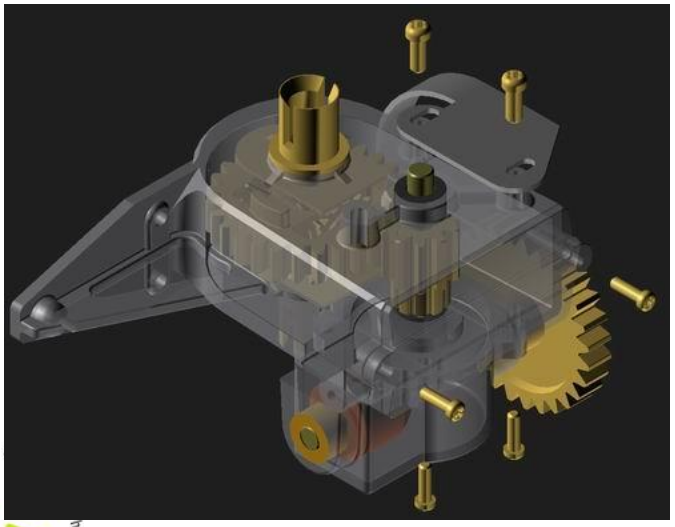
- 컴퓨터 그래픽스의 응용 분야

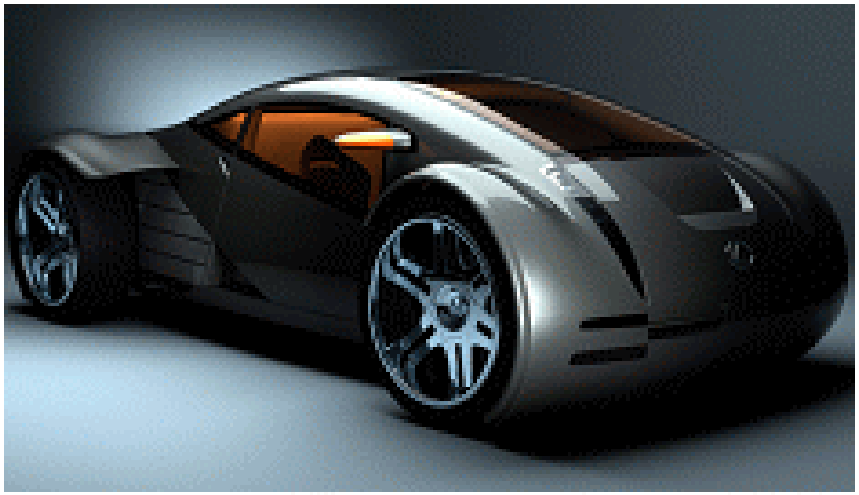
컴퓨터 그래픽스의 응용 분야



1. Computer-Aided Design (CAD)

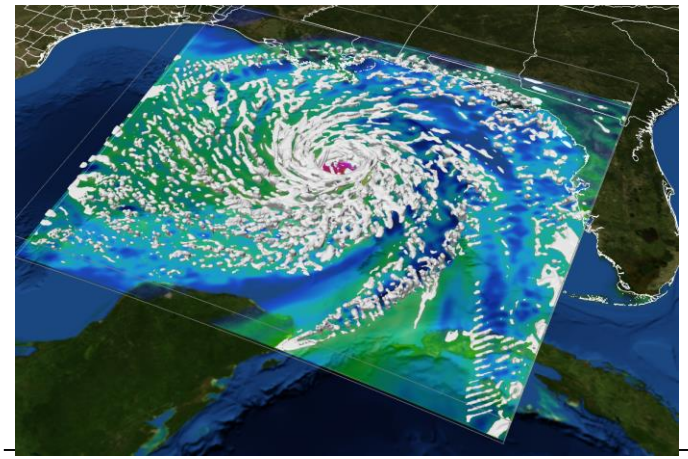
- Interactive한 컴퓨터 그래픽스 기술을 이용하여 설계에 필요한 인력,시간, 노력 등을 단축함으로써 설계 효율을 향상 (건축 설계, 제품 디자인, 산업디자인등)

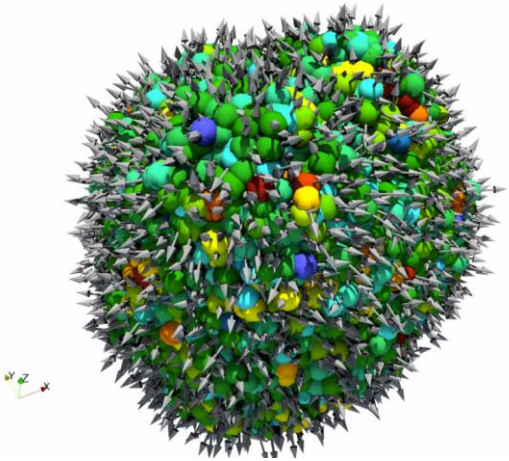




- 2. 애니메이션 및 게임
- 컴퓨터 그래픽스 기술은 2차원 또는 3차원 애니메이션 및 게임을 제작하는데 사용된다
- 실제로 촬영된 영상과 컴퓨터 그래픽스 기술을 조합하여 현실감을 높이기도 한다

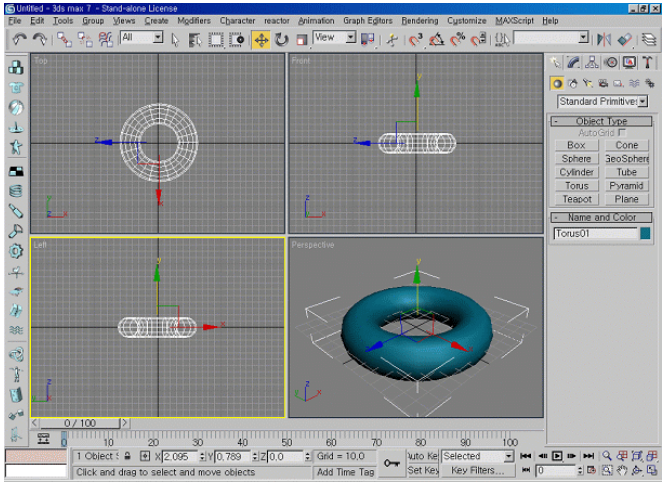
- **3. 과학분야 시각화 (scientific visualization)**
  - 과학 및 공학 분야에서 발생하는 데이터의 분석, 주로 대용량의 정보를 분석한다
  - 목적: 자연현상을 시각화 현상 내부의 패턴을 직관적으로 파악
  - 예: 허리케인 데이터 시각화, 의학 데이터 시각화 (암)
  - https://www.tacc.utexas.edu/scientific-visualization-gallery





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

윈도우 GUI



3D 스튜디오 MAX (3DS Max)

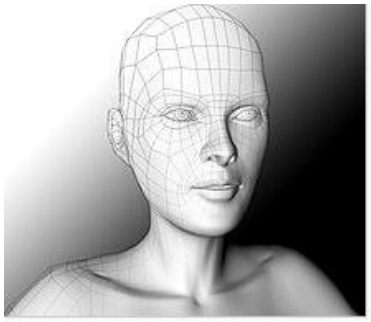
- 컴퓨터 그래픽스의 구성 요소

- 컴퓨터 그래픽스의 구성 요소
- 컴퓨터 그래픽스를 통해 어떠한 image를 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)

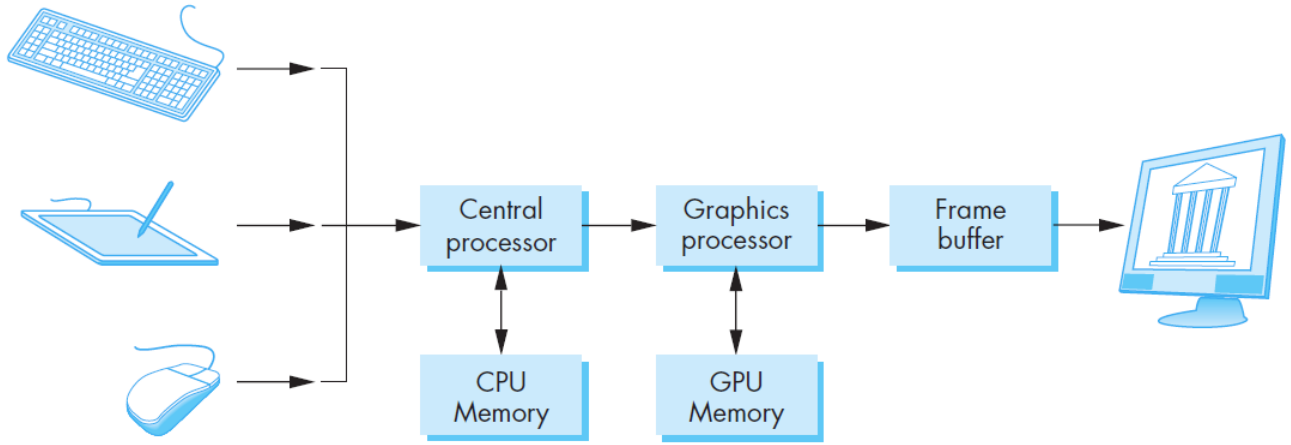


3D modeling of a human face

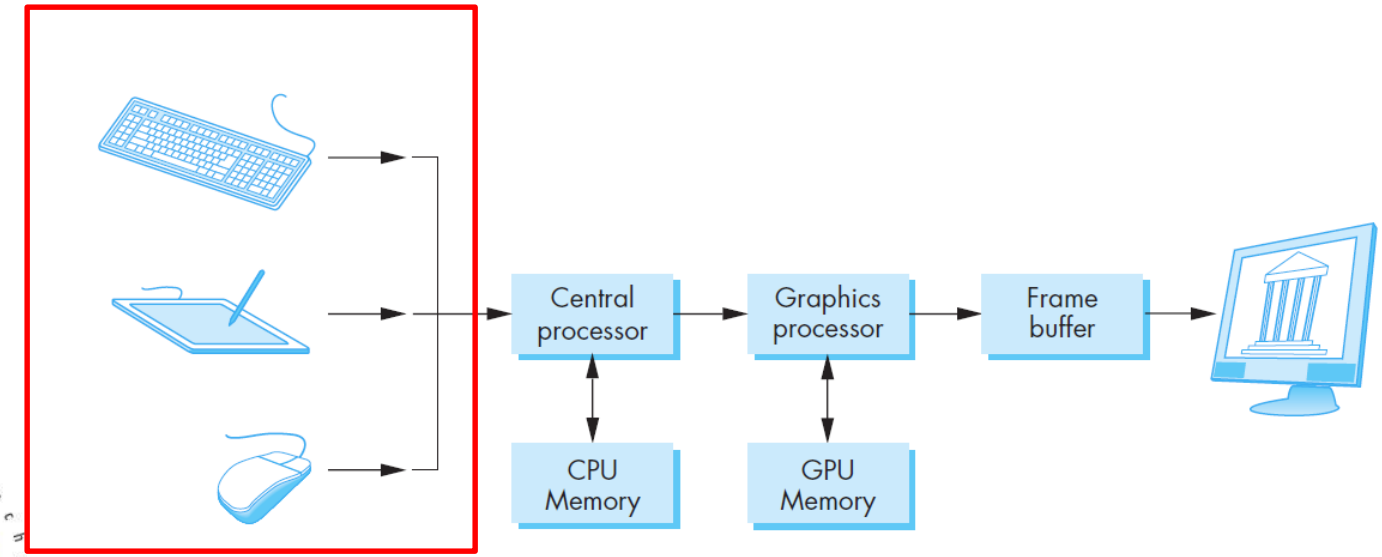
- **Rendering: 모델링 된 물체를 그려내는 작업**
- **조명 처리**
- **카메라의 위치, 방향 설정**
- **3차원의 물체를 최종적으로 어떻게 2차원으로 사상 (projection)시키는지 결정**
- **물체의 재질 (texture) 입히기**

- **그래픽스 시스템 (Graphics system)**

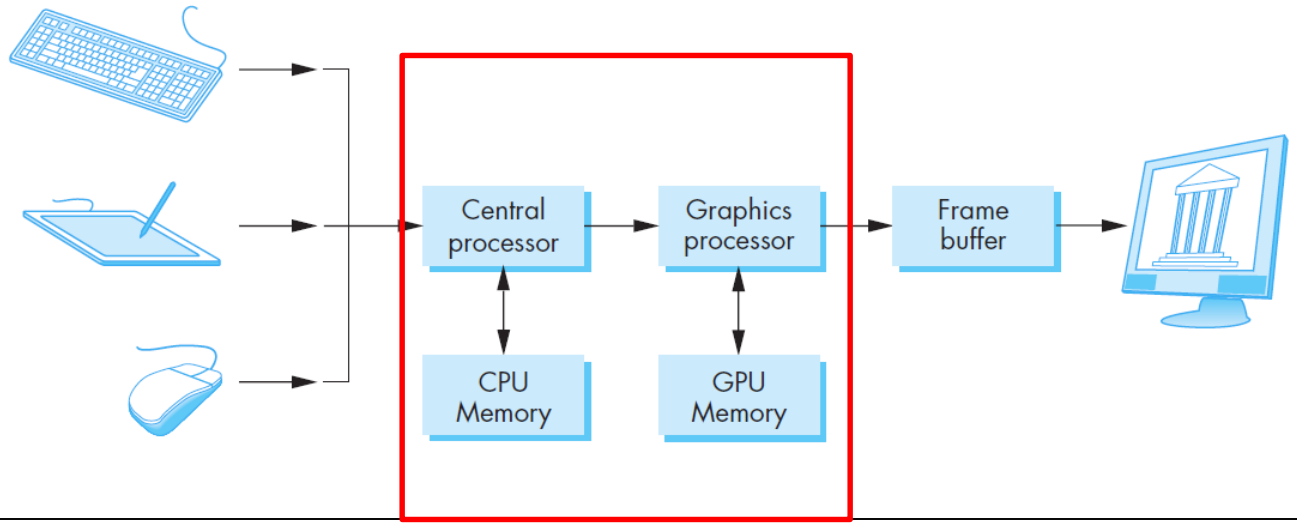
- 컴퓨터 그래픽스 시스템은 컴퓨터 시스템으로 일반적인 목적의 컴퓨터 시스템의 요소를 모두 갖추고 있다. 컴퓨터 그래픽스 시스템은 크게 보면
- 다음과 같은 구성 요소로 구성되어 있다
- 1. Input device 2. CPU (central processing unit)
- 3. GPU (graphics processing unit)
- 4. 메모리 5. Frame buffer 6. Output device



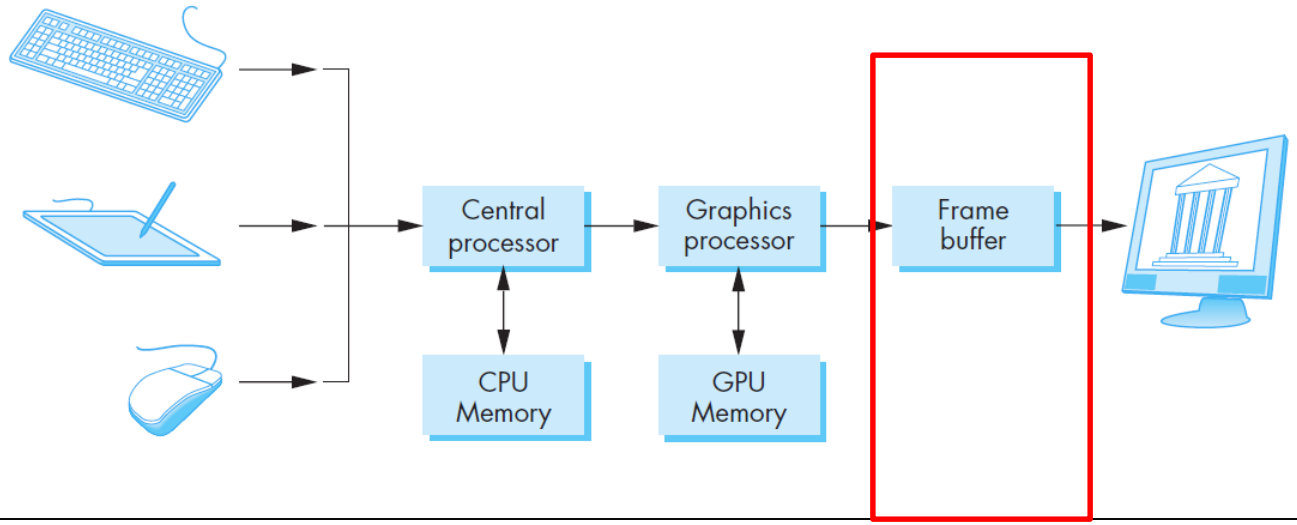
- **Input device (입력 장치)**
- **키보드, 마우스, 조이스틱, 테블릿등**



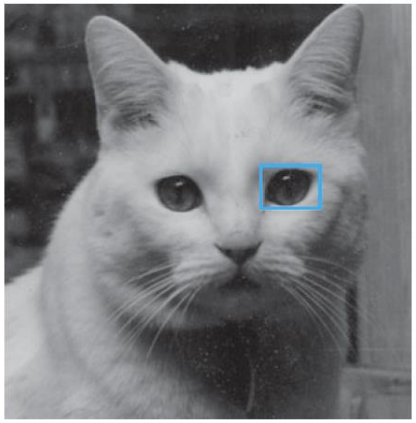
- **CPU and GPU**
- **간단한 시스템에서는 하나의 CPU만 존재 한다. 이 CPU는 연산과 그래픽 처리와 관련된 일을 수행한다**
- **최신 그래픽스 시스템에서는 특별한 그래픽스 함수를 수행 (처리)하기 위하여 GPU (graphics processing unit)를 가지고 있으며 이는 메인 보드에 있거나 그래픽스 카드 형태로 있다**

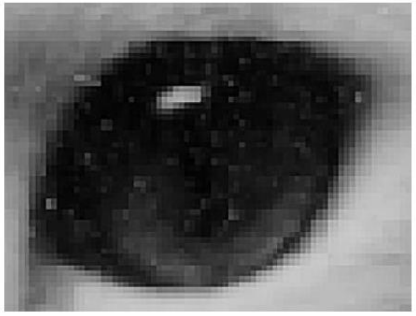


- **Frame buffer**
- 우리가 모니터와 같은 output device에서 최종적으로 보는 것은 image이다
- 이 image는 pixel (픽셀, picture element)로 이루어져 있다
- **Pixel**은 다른 말로 **raster**라고도 불린다

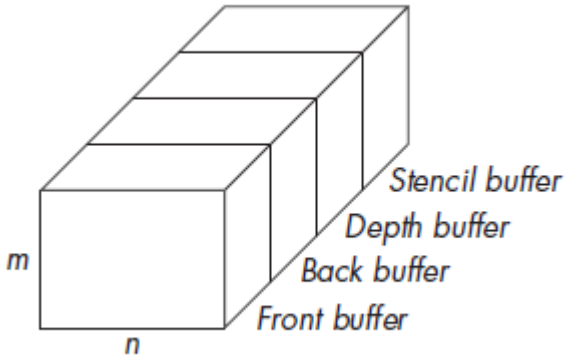


- 아래 그림과 같이 각각의 픽셀은 어떤 위치 (작은 영역을 갖는)에 대응한다
- 각 픽셀은 메모리에 frame buffer (buffer)라는 곳에 저장 된다
- Frame buffer에 있는 pixel 수를 해상도 (resolution)이라고 한다
- 기본적으로 메모리의 frame buffer에는 각 pixel의 color 정보가 저장된다 (그 이외에 depth와 같은 다른 정보도 저장된다)
- 즉, Frame buffer는 여러 개의 buffer로 구성되어 이는데 그 중에 color buffer가 포함된다고 생각하면 된다



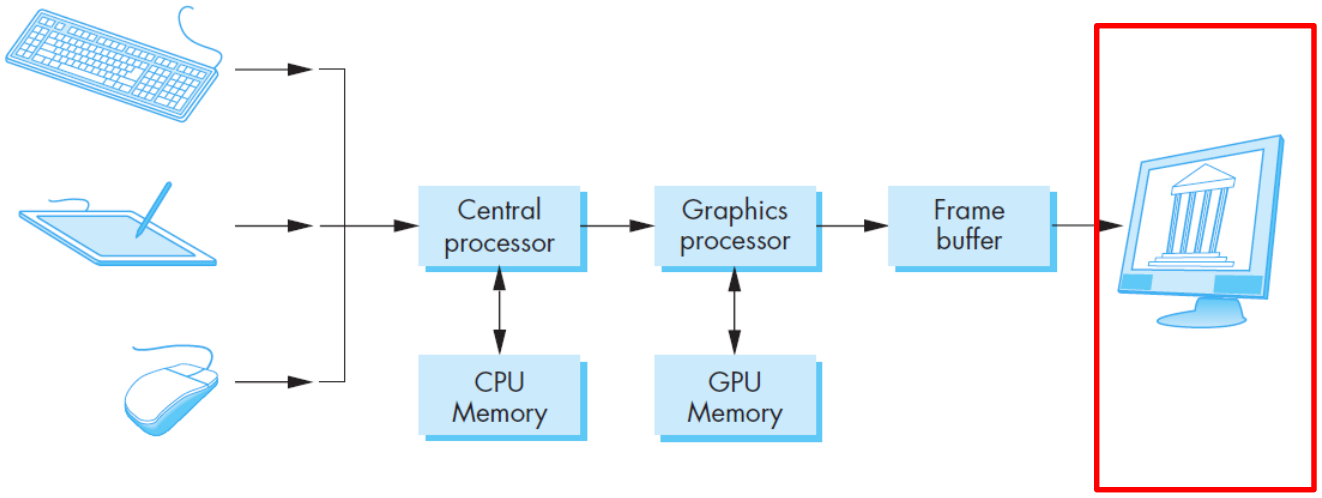


- **Frame buffer:**
- **It means the set of buffers that the graphics system uses for rendering including the color buffers, depth buffer, and other buffers the hardware may provide**
- **We can define a 2D buffer as a block of memory**
- **아래 왼쪽: OpenGL framebuffer**



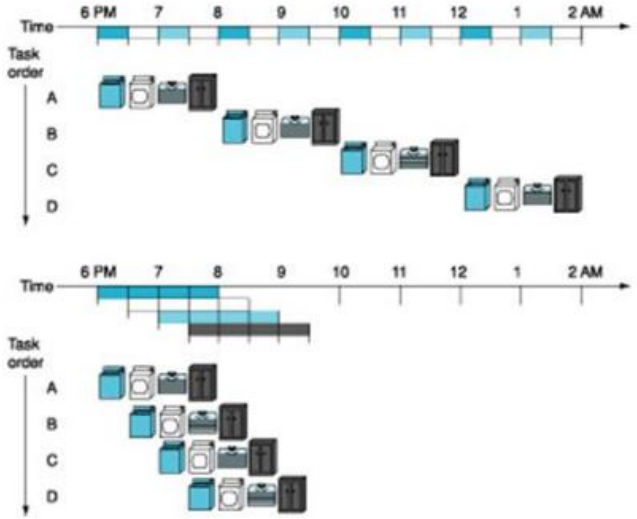

Sun TGX Framebuffer

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



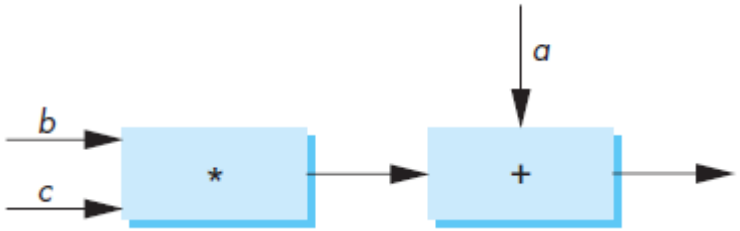
- **Pipelining**

- Pipelining이란? 컴퓨터 구조에서 나온 개념
- 예: Laundry example. 아주 많은 빨래가 있어서 하나의 세탁기에 다 들어가지 않은 경우
- 4가지 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

- **Concept of pipelining for a simple arithmetic calculation**
- **a+(b*c) : one multiplication and one addition**
- **Suppose we have to carry out the computation with many values of a, b, c**
- **The multiplier can pass on the results of its calculation to the adder and can start its next multiplication while the adder carries out the second step of the calculation on the first set of data**



- It takes the same amount of time to calculate the results for any one set of data, when we are working on two sets of data at one time, our total time for calculation is shortened
- We can construct pipelines for more complex arithmetic calculations that

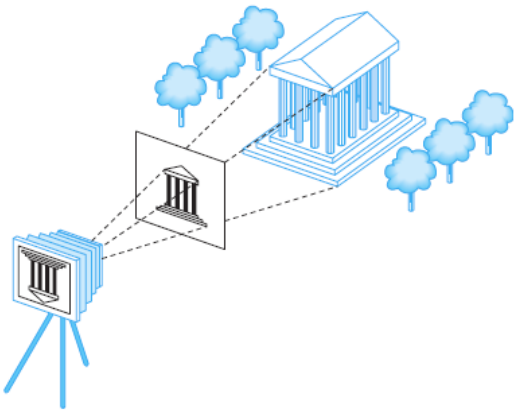
- **Graphics pipeline**

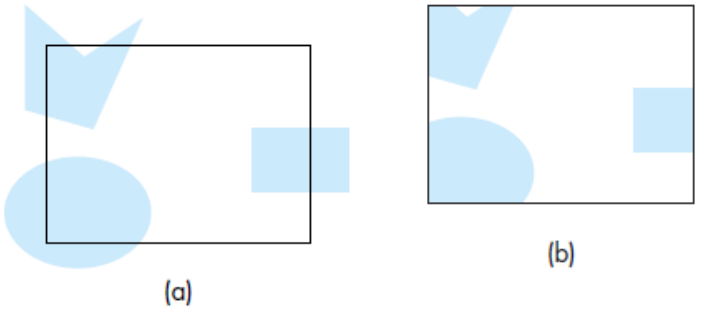
- Graphics pipeline: 컴퓨터 그래픽스 시스템에서도 기본적인 vertex로부터 최종적으로 모니터 (스크린)에 pixel로 보여지는데 까지 여러 개의 단계로 나뉘어져 있다

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

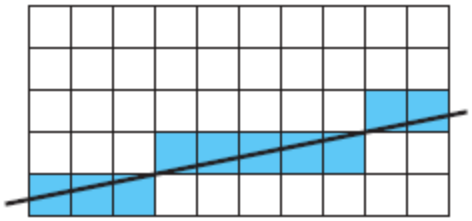
- Simplified graphics pipeline은 아래와 같이 4단계로 구성됨
- 기본적으로 물체는 여러 가지의 graphical primitive (예: 선분, 삼각형, 다각형)들로 구성
- 각각의 graphical primitive는 vertex들로 구성

- **1. Vertex processing**
- **Each vertex is processed independently**
- **Vertex의 좌표계 설정, 위치, 변환, color 등을 정함**
- **2. Clipper and primitive assembler**
- **인간의 시야각 (좌우위아래)을 생각해보면 clipping 필요**
- **Clipping은 vertex단위가 아닌 primitive (예: 선분, polygon) 단위로**



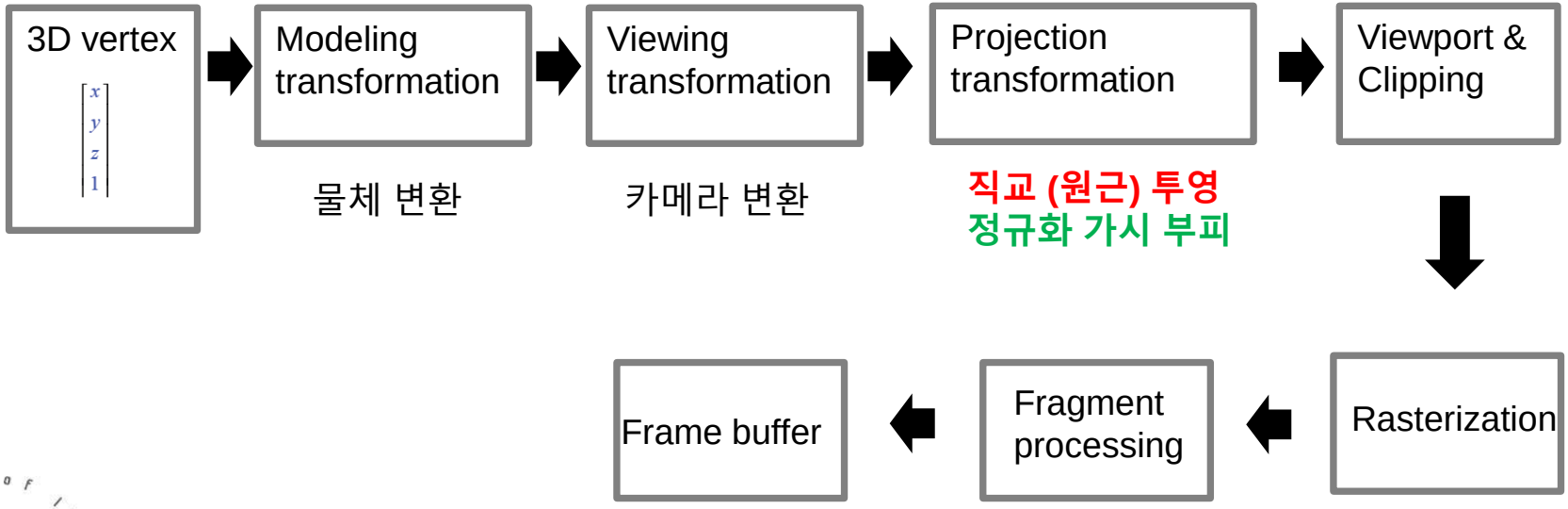


- **3. Rasterization (of primitives)**
- **Primitive into pixels in the framebuffer**

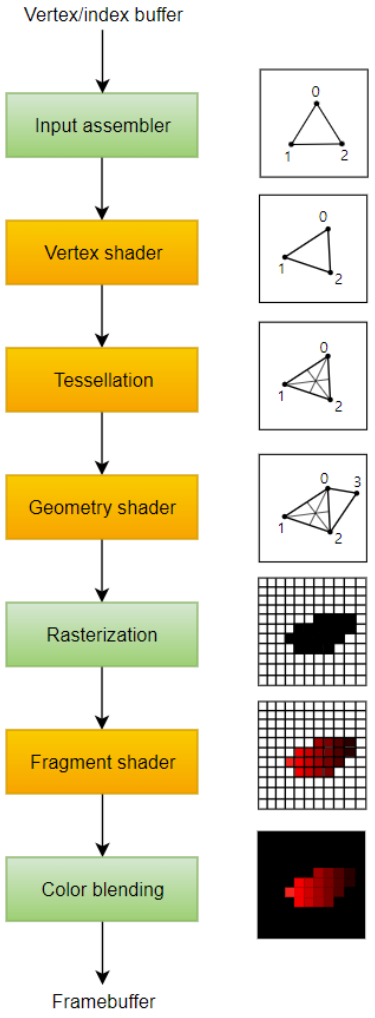


- **4. Fragment processing**
- **The output of the rasterization is a set of fragments for each primitive**
- **It has raster position, color, depth,…**

- **OpenGL 버전, GPU 사용 여부에 따라 Graphics pipeline에 차이는 있음**
- **예전 그래픽스 pipeline (OpenGL 1.x 버전)**



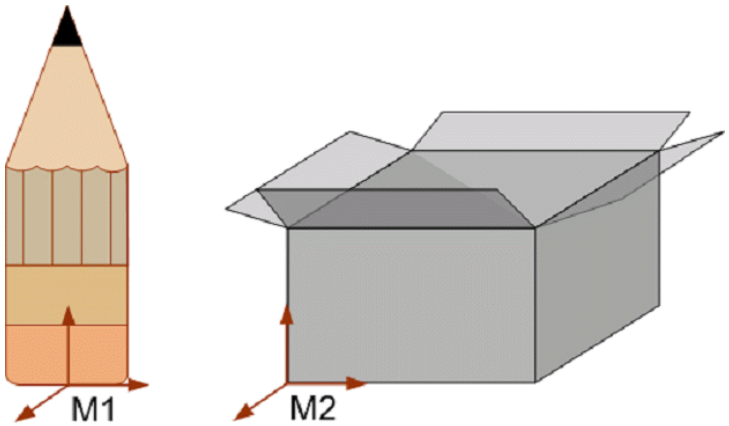
- 최근 graphics pipeline 예

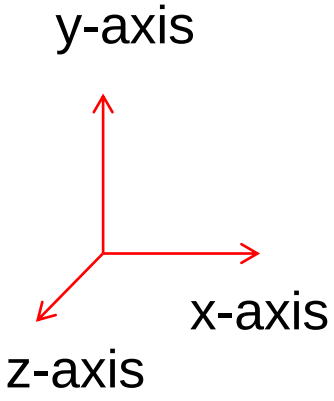


- **좌표계 (coordinate system)**

- Graphics pipeline에서는 각 pipeline 단계에서 여러 개의 서로 다른 좌표계 (coordinate system)가 사용된다
- Model coordinate, world coordinate, view coordinate, clip coordinate, normalized device coordinates, screen coordinate
- Graphics pipeline 내에서 순차적으로 작업이 수행됨에 따라서 여러 개의 다른 좌표계로 바뀌게 된다
- 많은 경우 Black-box와 같이 내부적으로 자동적으로 좌표계 변환이 이루어 져서 사용자가 변환할 필요는 없음

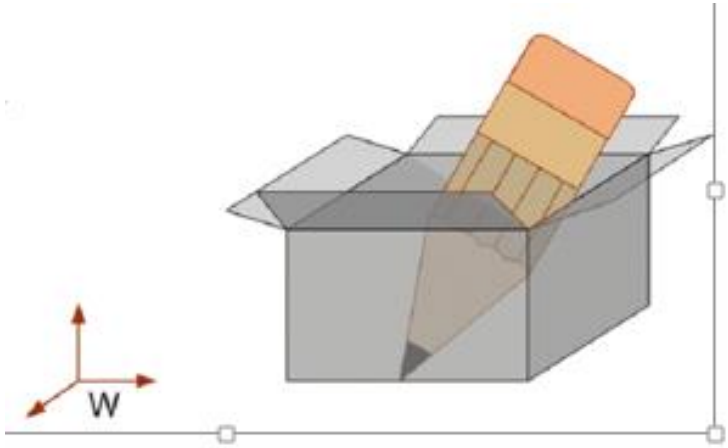
- **Local coordinate system (LCS, model coordinate system, model coordinate system)**
- **모델 자표계: 각 물체별로 물체를 모델링하기 편하게 설정된 좌표계**
- **Local coordinate system은 물체마다 좌표계의 원점 (origin: (0,0,0))및 축 방향이 (x, y, z axis) 다르다**



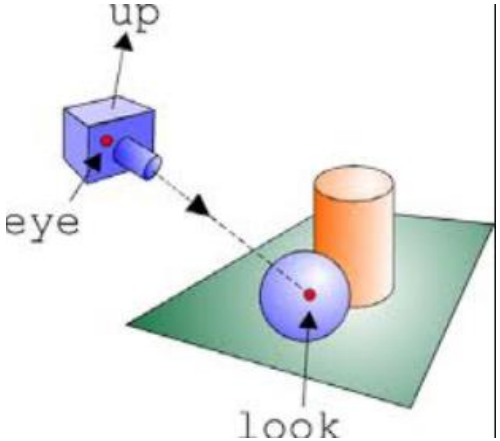


- 이렇게 물체 별로 좌표계를 각각 둔다면 각각의 물체를 모델링 하기에는 편하지만 물체가 여러 개 존재 한다면 불편할 수 있다.
- What about use one unified coordinate system for all objects?
- 즉, 여러 물체를 한꺼번에 아우를수 있는 좌표계가 전역 좌표계 (world coordinate system)이다

- **World coordinate system (세계, 전역 좌표계)**
- **Local coordinate system의 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**

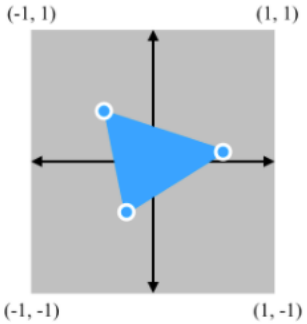


- **시점 좌표계 (View coordinate system)**
- **물체를 보는 사람 (카메라)의 위치와 보는 방향에 따라서 물체 모습이 다르게 보인다**
- **카메라 기준의 좌표계**
- **Synthetic camera model**
- **기본적으로 가상의 카메라 (눈)의 위치가 (0, 0, 0)에 있다고 가정**

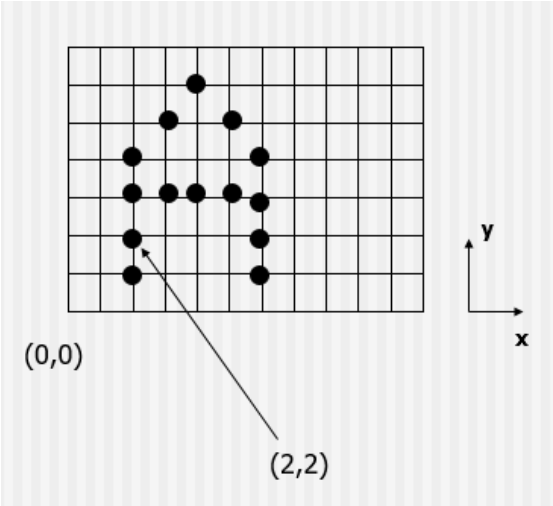


- **절단 좌표계:** 물체를 보는 사람 (카메라) 입장에서 보이지 않는 물체를 잘라내기 편하게 하기 위해 만든 좌표계로 정해진 좌표 범위 바깥에 있는 것은 clip (절단)

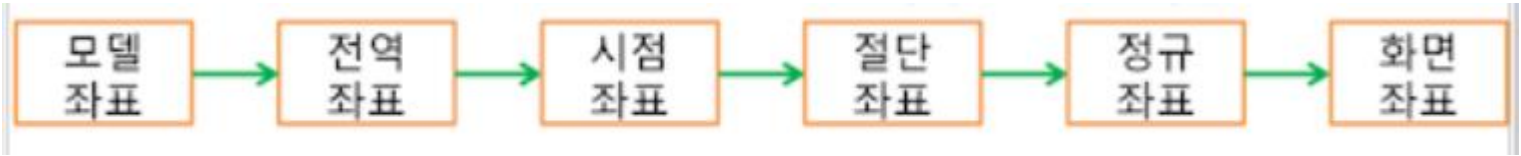
- **정규 좌표계 (NDC, normalized device coordinates):** 4D 동차좌표계를 3D 좌표계로 바꿈



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



- **Pre-shader OpenGL에서의 좌표계 변환 순서**



- **OpenGL**

- OpenGL: OpenGL : Open graphics library
- OpenGL이란 실리콘 그래픽스 (SGI)사가 개발한 3차원 컴퓨터 그래픽스 API (1992)
- OpenGL 그래픽스 함수는 프로그래밍 언어에 독립적인 기능으로 지정되어 있어서 C/C++/Java등 다수 언어와 사용 가능
- OpenGL은 cross-platform해서 OS에 독립적인 API 이다
- OpenGL은 프로그래밍 언어가 아니다 (수백 개의 함수 및 명령어로 이루어짐)
- https://www.opengl.org/

- OpenGL의 발전 과정
- 현재 OpenGL 4.6 버전 release
- https://www.opengl.org/
- 최근 버전에는 OpenGL shading language(GLSL) 추가 (GPU support)
- 본 수업에서는 이전 버전의 OpenGL인 pre-shader OpenGL 기준의 문법으로 실습할 예정임 (OpenGL 2.x 버전 기준)
- Why use old version of OpenGL?
- 본 수업의 목적은 OpenGL을 배우는 것이 목적이 아니라 그래픽스 이론 및 알고리즘을 배우는 것이 목적으로 처음 접하는 학생들에게는 이전 버전의 OpenGL이 더 이해하기가 쉽다

- **OpenGL libraries**
- **freeglut: freeglut takes care of the system-specific chores required for creating windows, initializing OpenGL contexts, and handling input events, to allow for portable OpenGL programs**
- **freeglut은 예전의 glut library를 대체**
- **glew library: glew is a cross-platform open-source C/C++ extension loading library. OpenGL core and extension functionality is exposed in a single header file**
- **glm library: glm is a header only C++ mathematics library for graphics software**

- **First OpenGL code**

- 교재에서 제공하는 소스코드 다운로드 가능
- sumantaguha – Just another WordPress site
- Downloads/Experimeter Source (book programs)
- 다운받고 압축 품
- 최초 실습 예
- ExperimenterSource/Chapter2/Square/square.cpp

- OpenGL 초기화시에 OpenGL context 만듬
- Rendering과 관계된 모든 데이터를 저장하는 state machine
- OpenGL은 (finite) state machine으로 다양한 state를 바꾸기 전까지 그 상태에 남아 있음 (예: color)
- Each state variable has a default value
- 이전 OpenGL 함수들 support

```
// Main routine.
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("square.cpp");

    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);

    glewExperimental = GL_TRUE;
    glewInit();

    setup();

    glutMainLoop();
}
```

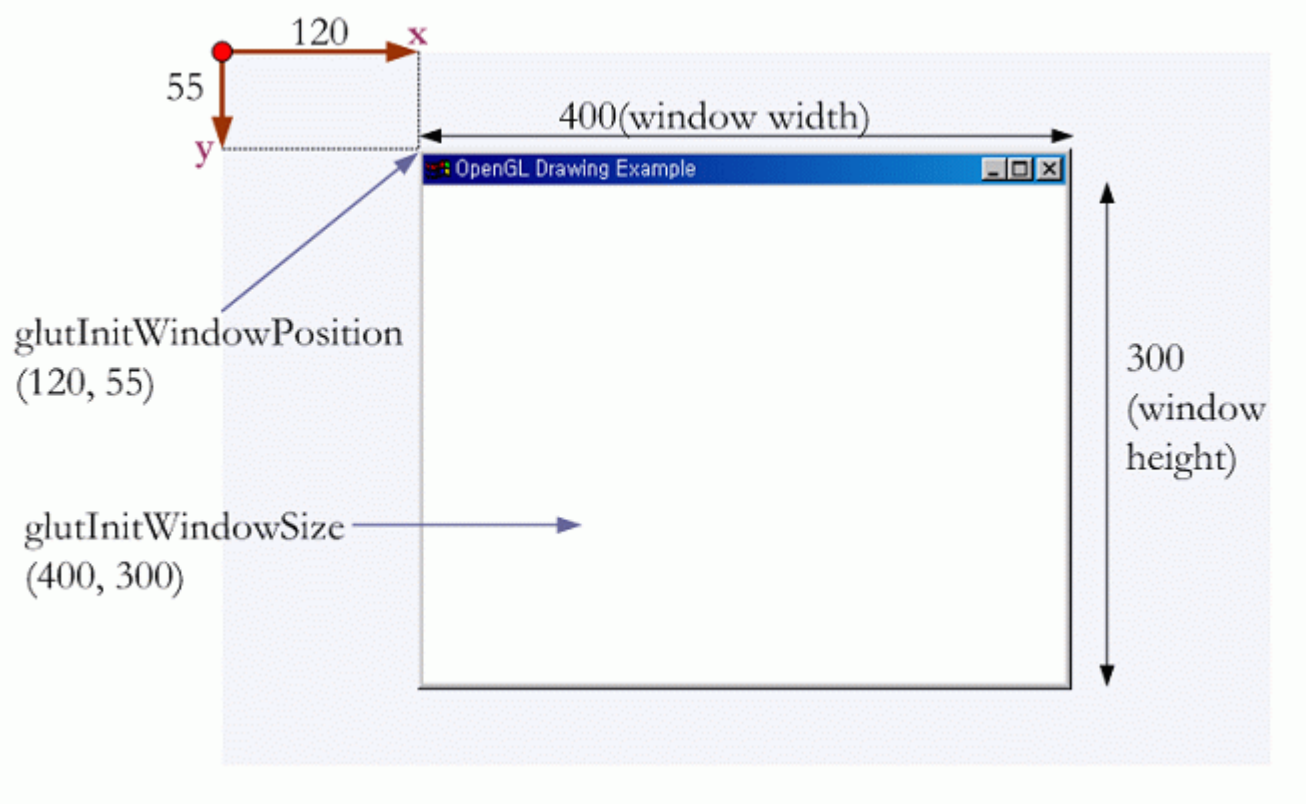
- 최초 display mode 설정
- single buffer (기본값)
- RGBA 모드
- (기본값)

```
// Main routine.
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("square.cpp");

    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);

    glewExperimental = GL_TRUE;
    glewInit();

    setup();

    glutMainLoop();
}
```

- OpenGL window 결과가 화면에 출력되는 window 가 가로 640 pixel , 세로 480 pixel 로 되게 설정 한다
- 위치도 설정 (픽셀)
- Window에 문자열

```
// Main routine.
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("square.cpp");

    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);

    glewExperimental = GL_TRUE;
    glewInit();

    setup();

    glutMainLoop();
}
```

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



- myDisplay 라는 함수를 display 이벤트에 대한 콜백 함수 (callback function)으로 등록 (register)해라
- This is called registering the callback (event listener) function
- Keyboard: 키보드 입력을 받을때의 콜백 함수
- Reshape: window 모양이 변화되었을때 콜백 함수

```
// Main routine.
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("square.cpp");

    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);

    glewExperimental = GL_TRUE;
    glewInit();

    setup();

    glutMainLoop();
}
```

- The mechanism employed by most graphics system is to use event processing, which gives us interactive control in our programs
- Events are changes that are detected by the operating system and include such actions as a user pressing a key on the keyboard
- When events occur they are placed in queue, the event queue
- We will be able to identify the events to which we want to react through functions called callback functions (event listeners)
- A callback function is associated with a specific type of events (e.g., mouse callback, display callback등)

- Glew 초기화
- glew에서 이용가능한 모든 OpenGL extension에 대한 정보를 가져옴

```
// Main routine.
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("square.cpp");

    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);

    glewExperimental = GL_TRUE;
    glewInit();

    setup();

    glutMainLoop();
}
```

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

- Drawing routine
- Color 정함
- **'glColor'** set the current color

```
void glColor3f( GLfloat red,
                GLfloat green,
                GLfloat blue);
```

- 1.0 (full intensity) and 0.0 (zero intensity)
- Polygon을 이루는 vertex 위치 정함

```
void drawScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 0.0, 0.0);

    // Draw a polygon with specified vertices
    glBegin(GL_POLYGON);
    glVertex3f(20.0, 20.0, 0.0);
    glVertex3f(80.0, 20.0, 0.0);
    glVertex3f(80.0, 80.0, 0.0);
    glVertex3f(20.0, 80.0, 0.0);
    glEnd();

    glFlush();
}
```

- **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**

glClearColor() => color 초기화

glClearColor(red, green, blue, alpha)

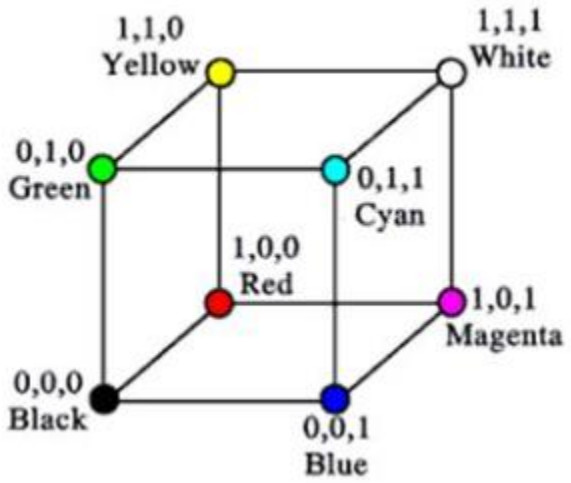
각각의 red, green, blue에서

1.0 (full intensity), 0.0 (zero intensity)

alpha값: 불투명도 (opacity) , 현재는 그냥 0.0으로 둔다

```
// Initialization routine.
void setup(void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
}
```

```
setup();
```



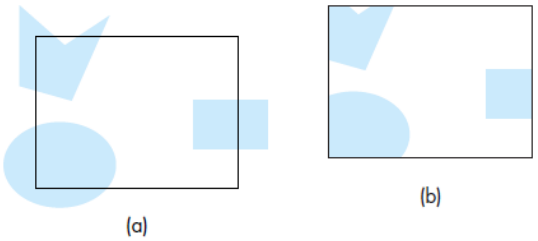
- Viewport와
- 가시 부피 설정 부분
- glOrtho 함수

- 다음에 이어서 설명

```
void resize(int w, int h)
{
    glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 100.0, 0.0, 100.0, -1.0, 1.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

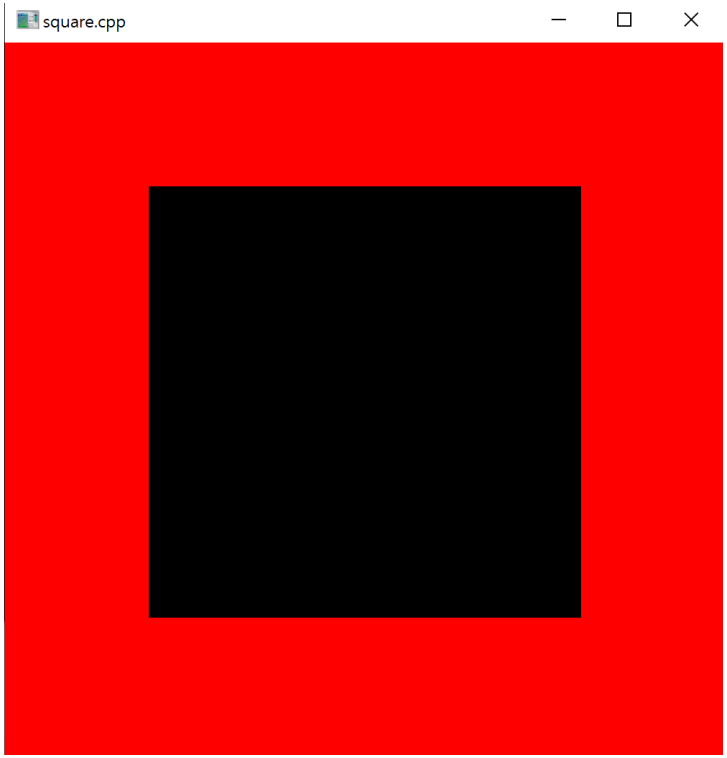
- 가시 공간, 가시 부피 설정

- **가시 공간 (2D: viewing rectangle)**
- **We consider 2D viewing directly by taking a rectangular area of our 2D world and transferring its contents to the display**
- **It is known as the viewing rectangle (clipping rectangle)**
- **Objects inside the rectangle are displayed**
- **Objects outside are clipped out and are not displayed**

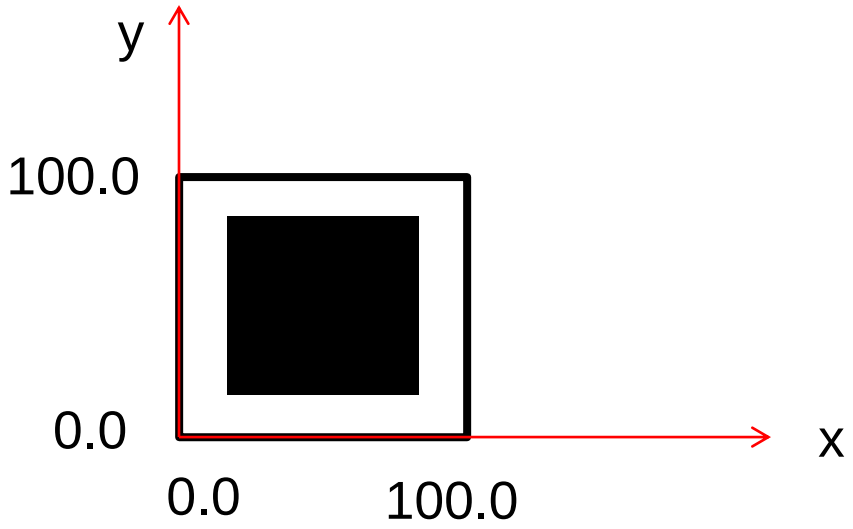


2D viewing rectangle (a) objects before clipping (b) after clipping

- 앞의 코드에서
- glOrtho 부분을 아래와 같이 바꿔보고 코드를 실행해 보고
- gluOrtho2D(0.0, 100.0, 0.0, 100.0)
- glClearColor도 아래와 같이 바꾸어보자
- glClearColor(1.0, 0.0, 0.0, 0.0);



- **gluOrtho2D(left, right, bottom, top); // 가시공간 (직사각형) 설정**
- **gluOrtho2D(0.0, 100.0, 0.0, 100.0); // 2D로 직사각형 가시 공간 설정**
- **left, right: specify the coordinates for the left and right vertical clipping planes**
- **Bottom, top: specify the coordinates for the bottom and top horizontal clipping planes**



- 이번에는
- gluOrtho2D(20.0, 80.0, 20.0, 80.0)
- 와 같이 바꿔보자.
- OpenGL window 창에서 어떻게 보일까?