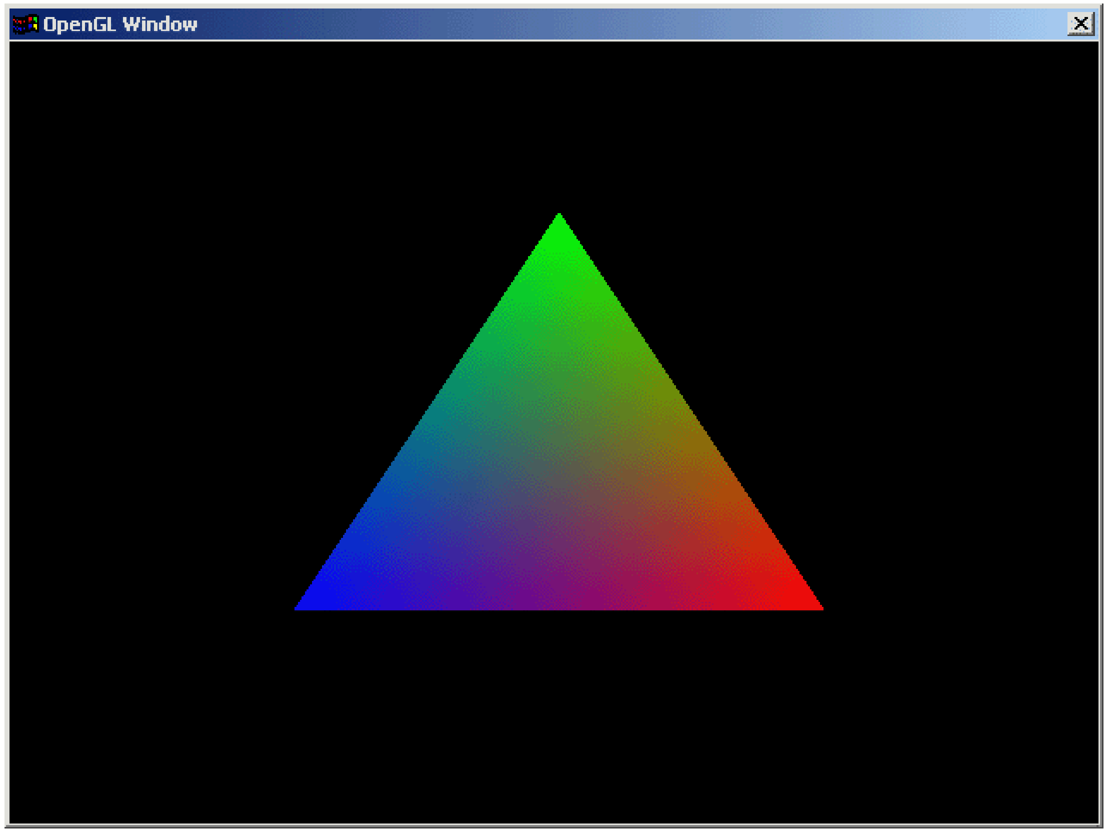
Basic OpenGL/Xwindow

Why OpenGL ?

- **Easy to learn**
- **Platform-independent**
  - OpenGL is a industry standard framework
  - Ranging from PC to mobile system

OpenGL Basic

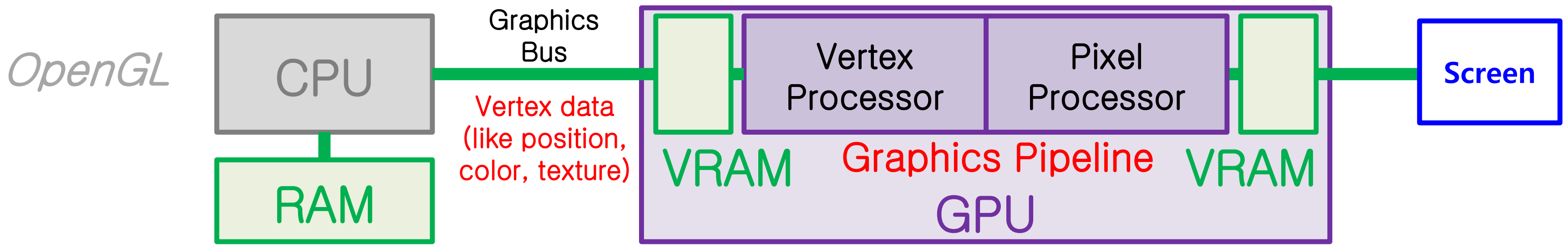


OpenGL Advanced

OpenGL Raytracing

Basic Drawing: OpenGL?

- **Basic drawing**
  - Vertex data is transferred in **serial** through graphics bus.
  - Fixed Graphics pipeline.
  - Before OpenGL 2.0, only basic drawing was possible.



- **GLSL drawing**
  - Vertex data is transferred in **parallel** in buffer unit.
  - GLSL controls the Graphics pipeline to perform parallel processing.

API & OpenGL APIs

- **API (Application Programmer's Interface)**
  - A set of functions with a well-defined interface
- **OpenGL core**
  - Based on C
- **GLU(OpenGL Utility Library)**
  - Provides functionality in OpenGL core.
  - Part of OpenGL
- **GLUT(OpenGL Utility Toolkit)**
  - Provides functionality common to all window systems.
  - Not part of OpenGL
- **GLX**
  - OpenGL Extension to the X Window System
- **OpenGL Extensions(For GLSL)**
  - Updated as graphics hardware is improved.
  - Supports direct access to graphics pipeline. (GPU accel.)

Practice: Copying Source codes

Copy XWindow.h file to your project folder

Order1:

```
cp /home/share/XWindow.h ./
```

Copy XWindow.cpp file to your project folder

Order2:

```
cp /home/share/XWindow.cpp ./
```

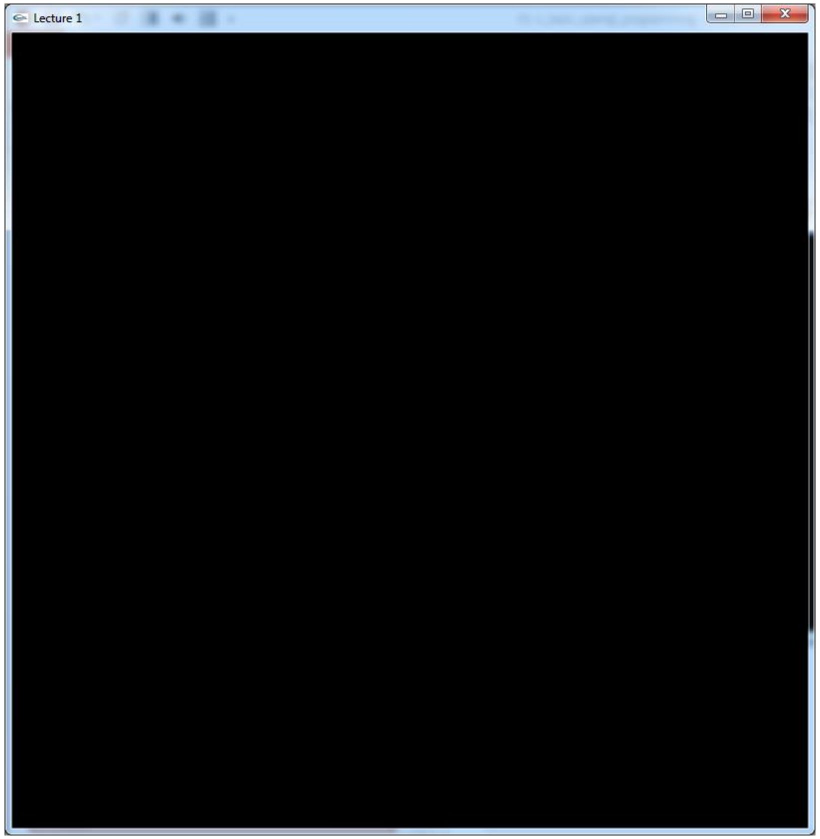
Copy EventHandle.cpp file to your project folder

Order3:

```
cp /home/share/EventHandle.cpp ./
```

Create an Empty XWindow

- **XWindow 만들기**
  - Open File “XWindow.cpp”



Sample Code: Header File

#include “XWindow.h”

#include<stdio.h>	:	Standard Input/Output library
#include<string.h>	:	C string handling library
#include<X11/Xlib.h>	:	Main library for X Window
#include<X11/XKBlib.h>	:	X Window Keyboard extension
#include<GL/gl.h>	:	OpenGL Library
#include<GL/glx.h>	:	GL Extension to the Xwindow system.
#include<GL/glu.h>	:	OpenGL Utility Library
#include<GL/glut.h>	:	OpenGL Utility Toolkit
#include<GL/glex.h>	:	OpenGL Extension Wrangler Library

Global Variables for X-Window

```
Display                 *dpy;
Window                  root;
Glint                   att[] = { GLX_RGBA, GLX_DEPTH_SIZE, 24,
                                  GLX_DOUBLEBUFFER, None };
XVisualInfo             *vi;
Colormap                cmap;
Window                   win;
GLXContext               glc;
XSetWindowAttributes          swa;
XWindowAttributes             gwa;
```

Sample Code: Setting variables

Reserved Variable	Explaination
Display	Main types of data in X-window
Window	A rectangular area on the screen that lets you view graphic output
GLint	Integer variable for openGL
XVisulInfo	Struct to list details of a Visual
XSetWindowAttributes	Create windows and window attributes st ructure
GLXContext	Context to attach X window
XWindowAttributes	Current window attributes structure

Main Function for Create X-Window

```
int main(int argc, char *argv[]) {
        dpy = XOpenDisplay(NULL);    //Connect X-Server
        root = DefaultRootWindow(dpy); //Get root window from X-Server
        vi = glXChooseVisual(dpy, 0, att); //Set glx visual information
        cmap = XCreateColormap(dpy, root, vi->visual, AllocNone);
        swa.colormap = cmap;
        //Create Window to display
        win = XCreateWindow(dpy, root, 0/*position x*/, 0/*position y*/,
                 600/*width*/, 600/*Height*/, 0,
                vi->depth, InputOutput, vi->visual,
                CWColormap | CWEventMask, &swa);
        XMapWindow(dpy, win);//Mapping window to display
        XStoreName(dpy, win, "Lecture");//Set title of Window
        glc = glXCreateContext(dpy, vi, NULL, GL_TRUE);//Create glX Context
        glXMakeCurrent(dpy, win, glc);//set glX Context to window
}
```

Display Function & Main Loop

```
void display() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        /*Draw Call Here*/
        glXSwapBuffers(dpy, win);
}

int main(int argc, char *argv[]){
        /*Create Windows*/
        glEnable(GL_DEPTH_TEST);//Use GL Depth

        while(1) {//Infinite loop for main loop
                display();//Call display function
        }
}
```

Exit function

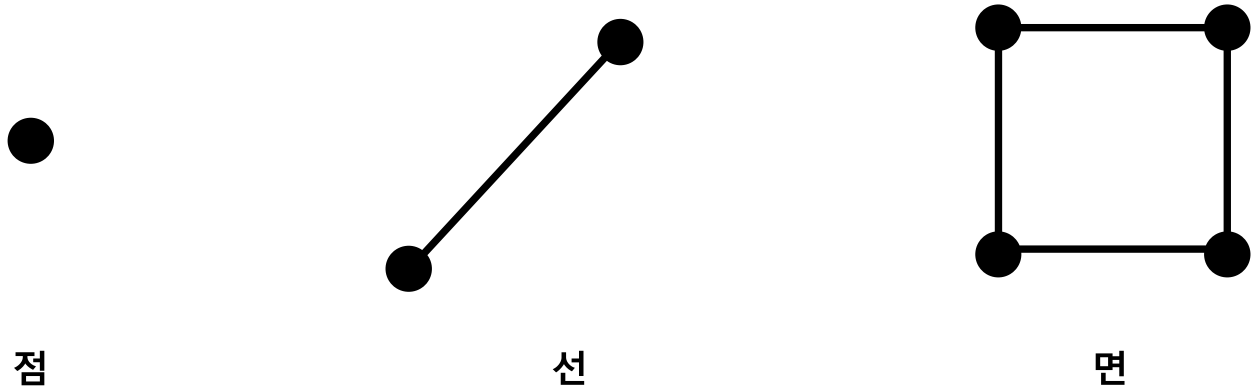
```
void ExitProgram() {
        glXMakeCurrent(dpy, None, NULL); //Clear display setting
        glXDestroyContext(dpy, glc); //Destroy glx context
        XDestroyWindow(dpy, win); //Destroy the window
        XCloseDisplay(dpy); //Close the display(X-server)
        exit(0);
}
```

- **How to call ExitProgram function in the mainloop?**
  - We'll discuss event handler on the X-Window later.

Drawing Primitives

OpenGL Basic Primitive

- 그림을 그리기 위한 기본 요소



OpenGL Drawing Function

Draw Example (Triangle 1/4)

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 0.5f, 0.0f);
        glVertex3f(-0.5f, -0.5f, 0.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
    glEnd();

    glXSwapBuffers(dpy, win);
}
```

OpenGL Drawing Function

Draw Example (Triangle 2/4)

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 0.5f, 0.0f);
        glVertex3f(-0.5f, -0.5f, 0.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
    glEnd();

    glXSwapBuffers(dpy, win);
}
```

Drawing의 시작과 끝을 알리는 함수

OpenGL Drawing Function

Draw Example (Triangle 3/4)

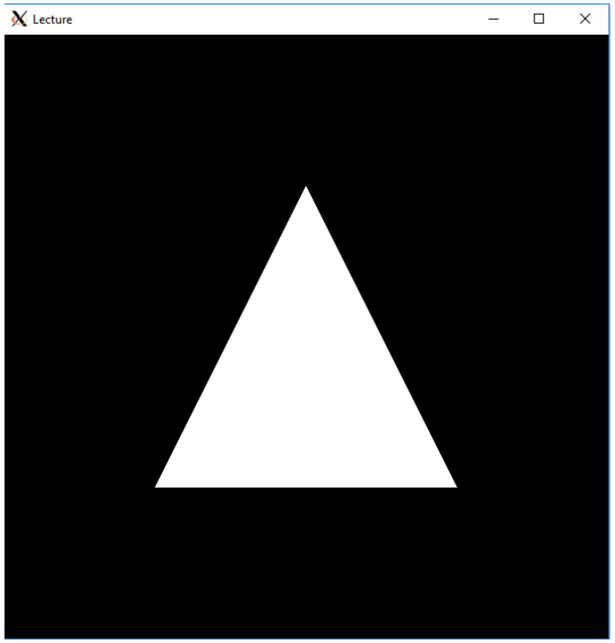
```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 0.5f, 0.0f);
        glVertex3f(-0.5f, -0.5f, 0.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
    glEnd();

    glXSwapBuffers(dpy, win);
}
```

어떤 그림을 그릴 것인가

Draw Example (Triangle 4/4)

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 0.5f, 0.0f);
        glVertex3f(-0.5f, -0.5f, 0.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
    glEnd();

    glXSwapBuffers(dpy, win);
}
```

어느 위치에 점을 찍을 것인가



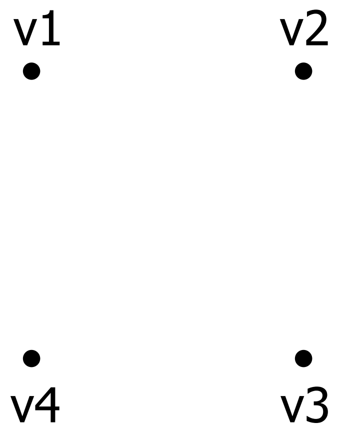
OpenGL Primitives

- **Point**
  - GL_POINTS
- **Line**
  - GL_LINES | GL_LINE_STRIP | GL_LINE_LOOP
- **Polygon**
  - GL_POLYGON
  - GL_TRIANGLES | GL_TRIANGLE_STRIP | GL_TRIANGLE_FAN
  - GL_QUAD_STRIP

Point

- **GL_POINTS**

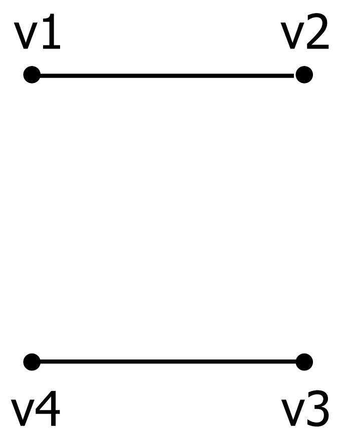
```
glBegin(GL_POINTS);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
glEnd();
```



Line

- **GL_LINES**

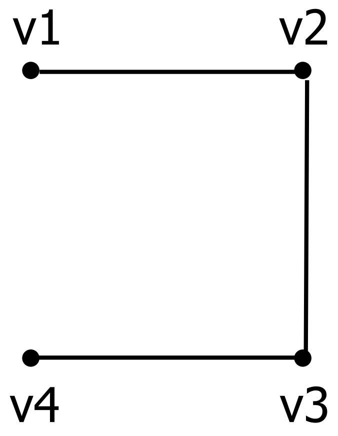
```
glBegin(GL_LINES);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
glEnd();
```



3D Connected Lines

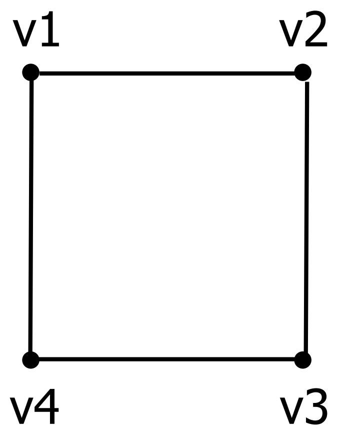
- **GL_LINE_STRIP**

```
glBegin(GL_LINE_STRIP);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
glEnd();
```



- **GL_LINE_LOOP**

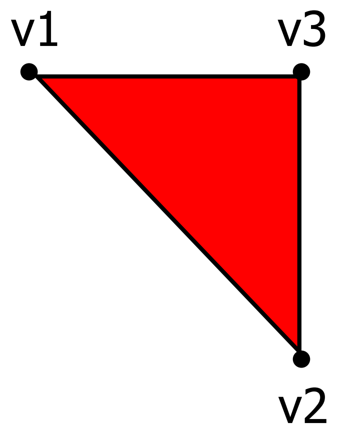
```
glBegin(GL_LINE_LOOP);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
glEnd();
```



3D Triangle

- **GL_TRIANGLES**

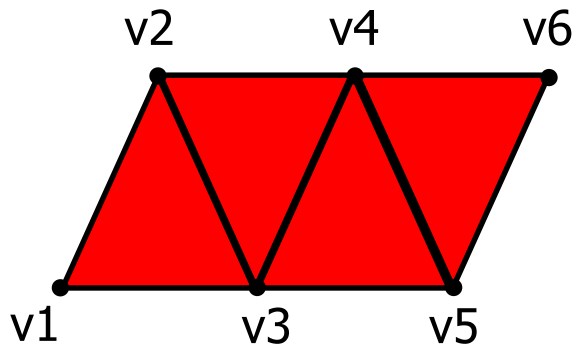
```
glBegin(GL_TRIANGLES);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
glEnd();
```



3D Triangle

- **GL_TRIANGLE_STRIP**

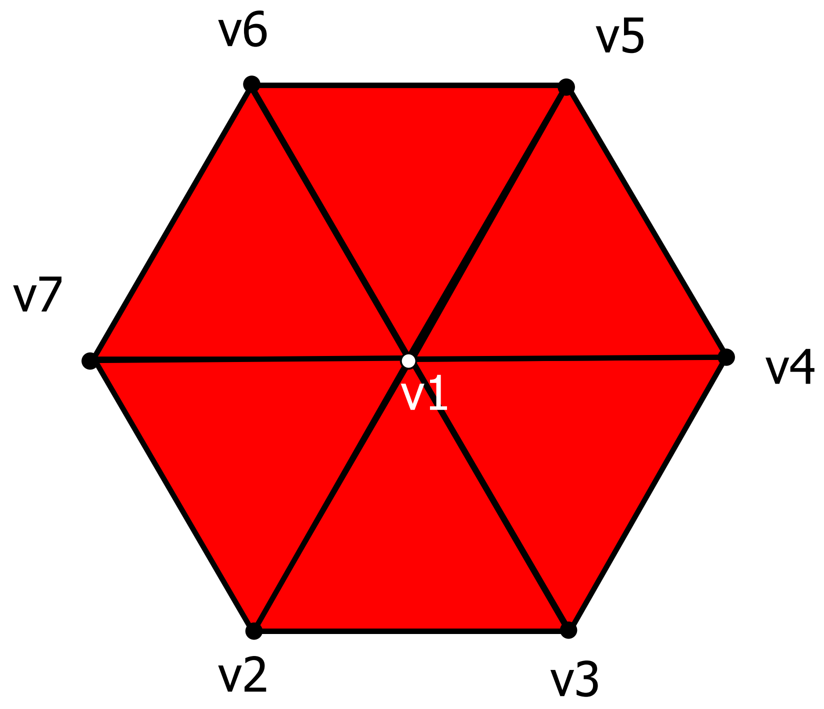
```
glBegin(GL_TRIANGLE_STRIP);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
    glVertex3f(v5x, v5y, v5z);
    glVertex3f(v6x, v6y, v6z);
glEnd();
```



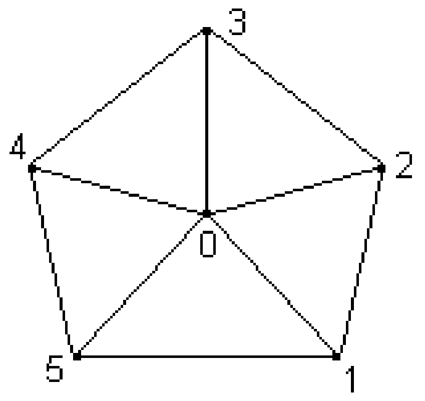
3D Triangle

- **GL_TRIANGLE_FAN**

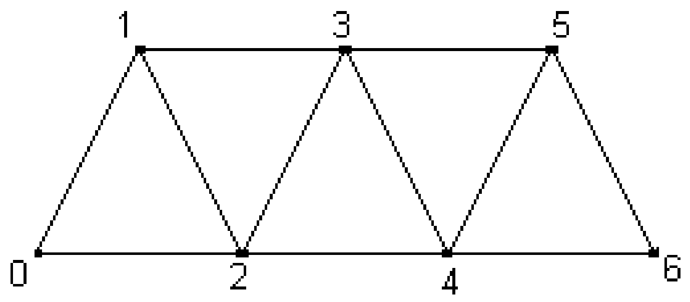
```
glBegin(GL_TRIANGLE_FAN);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
    glVertex3f(v5x, v5y, v5z);
    glVertex3f(v6x, v6y, v6z);
    glVertex3f(v7x, v7y, v7z);
glEnd();
```



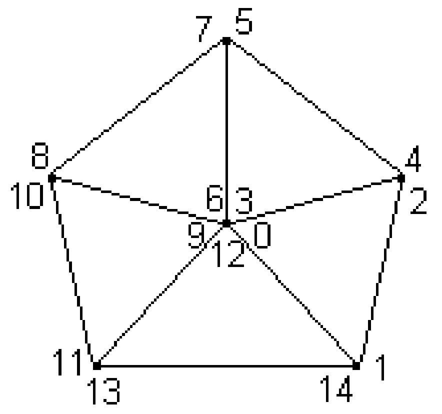
What's the difference?



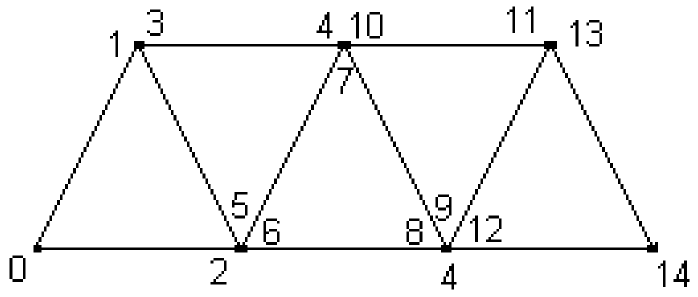
GL_TRIANGLE_FAN



GL_TRIANGLE_STRIP



Same figure with GL_TRIANGLES

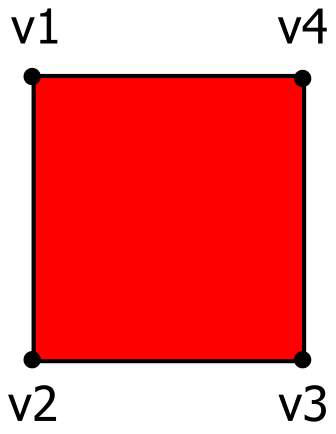


Same figure with GL_TRIANGLES

3D Quadrilateral

- **GL_QUADS**

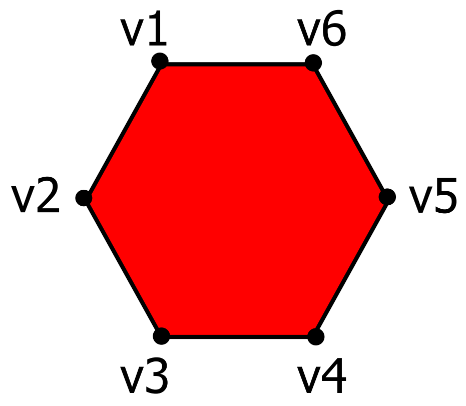
```
glBegin(GL_QUADS);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
glEnd();
```



3D Polygon

- **GL_POLYGON**

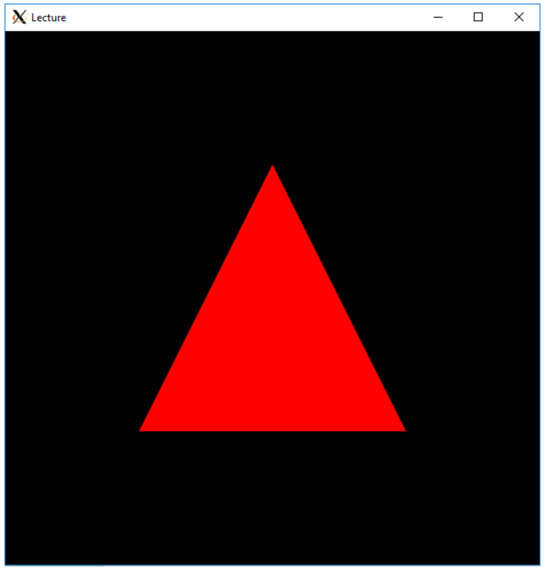
```
glBegin(GL_POLYGON);
    glVertex3f(v1x, v1y, v1z);
    glVertex3f(v2x, v2y, v2z);
    glVertex3f(v3x, v3y, v3z);
    glVertex3f(v4x, v4y, v4z);
    glVertex3f(v5x, v5y, v5z);
    glVertex3f(v6x, v6y, v6z);
glEnd();
```



OpenGL Drawing Function

Red Triangle

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 0.5f, 0.0f);
        glVertex3f(-0.5f, -0.5f, 0.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
    glEnd();

    glXSwapBuffers(dpy, win);
}
```

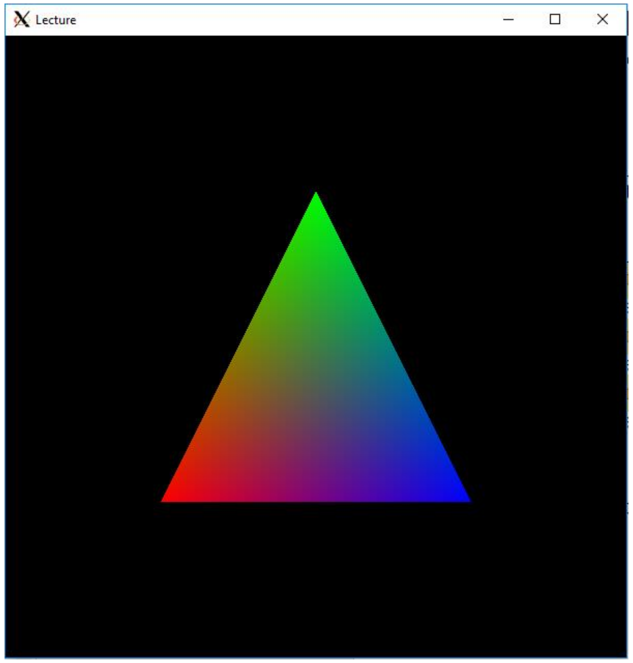


glColor3f(R, G, B);

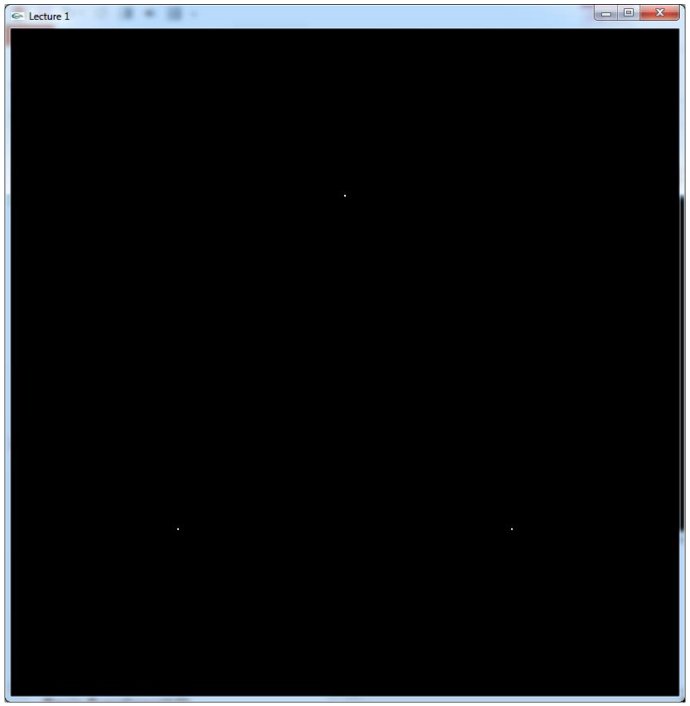
OpenGL Drawing Function

Colorful Triangle

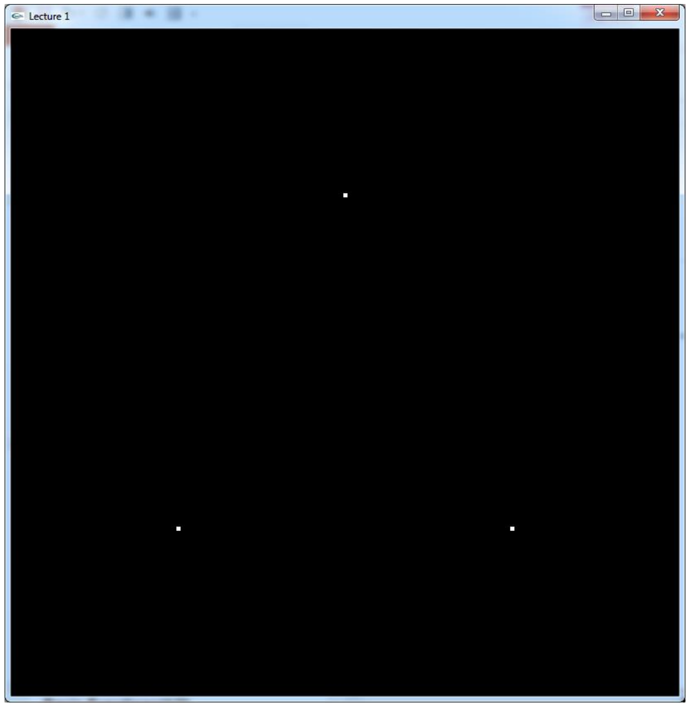
```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glBegin(GL_TRIANGLES);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 0.5f, 0.0f);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex3f(-0.5f, -0.5f, 0.0f);
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
    glEnd();

    glXSwapBuffers(dpy, win);
}
```



OpenGL Drawing Function
Point Size



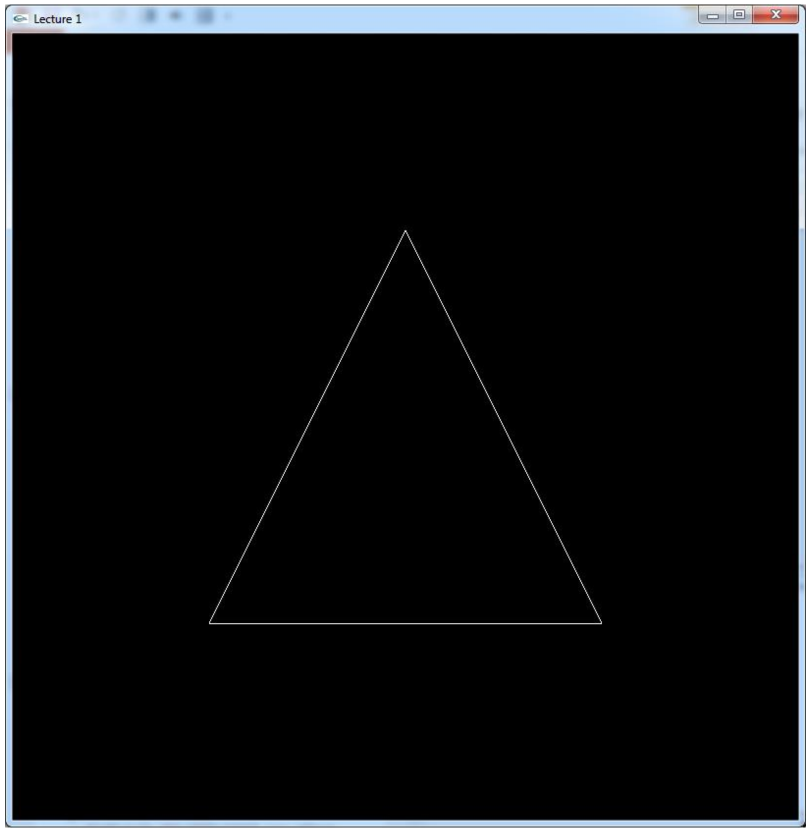
Point size : 2.0



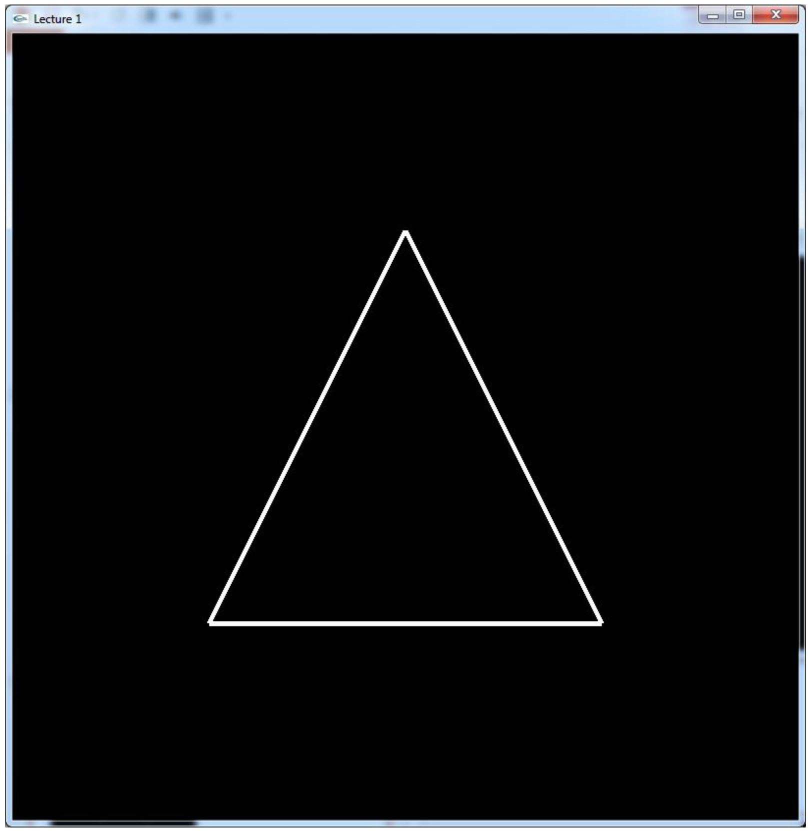
Point size : 5.0

glPointSize(GLfloat size)

OpenGL Drawing Function
Line Width



Line width : 1.0



Line width : 5.0

glLineWidth(GLfloat width)

Callback Functions@GLUT Window

Callback Functions

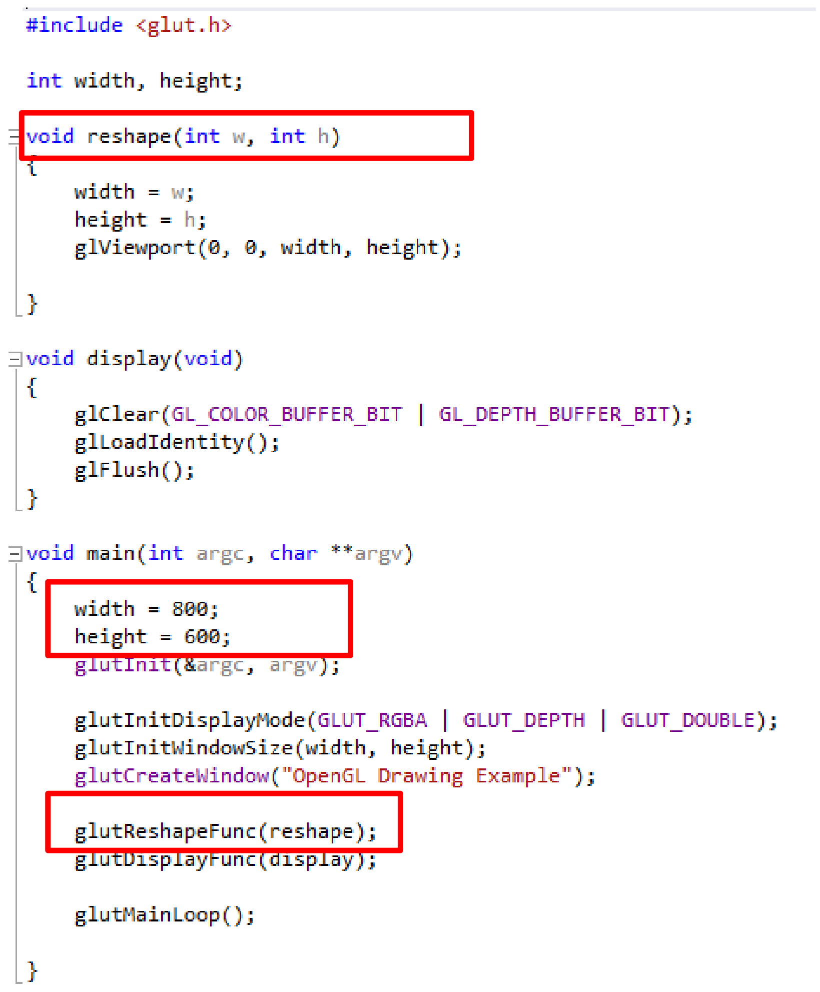
- **Registers a specialized user-defined function**
  - Called when a certain event occurs
- **Useful callback functions**
  - glutReshapeFunc()
  - glutKeyboardFunc()
  - glutMouseFunc()
  - glutMotionFunc()
  - glutPassiveMotionFunc()
  - glutIdleFunc()
  - glutTimerFumc()

Reshape Callback Functions

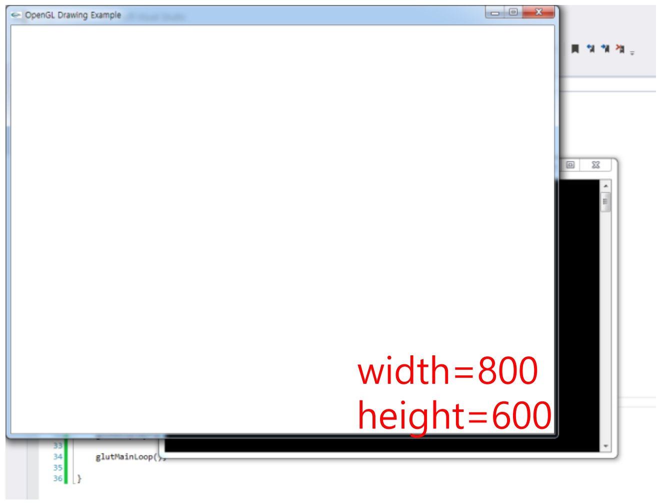
- void **glutReshapeFunc**(void(*func)(int width, int height))
  - Called when the window size or shape is changed
    - width : new width of a window
    - height : new height of a window

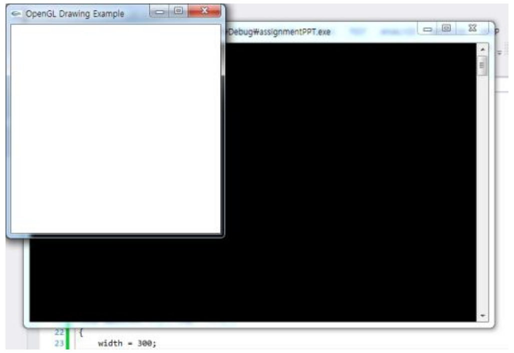
  - Main 함수 위에 정의
    - Void Reshape(int w, int h);
  - Main 함수 내에 작성
    - glutReshapeFunc(Reshape);

glReshapeFunc()



```
#include <glut.h>

int width, height;

void reshape(int w, int h)
{
    width = w;
    height = h;
    glViewport(0, 0, width, height);

}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glFlush();
}

void main(int argc, char **argv)
{
    width = 800;
    height = 600;
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
    glutInitWindowSize(width, height);
    glutCreateWindow("OpenGL Drawing Example");

    glutReshapeFunc(reshape);
    glutDisplayFunc(display);

    glutMainLoop();

}
```

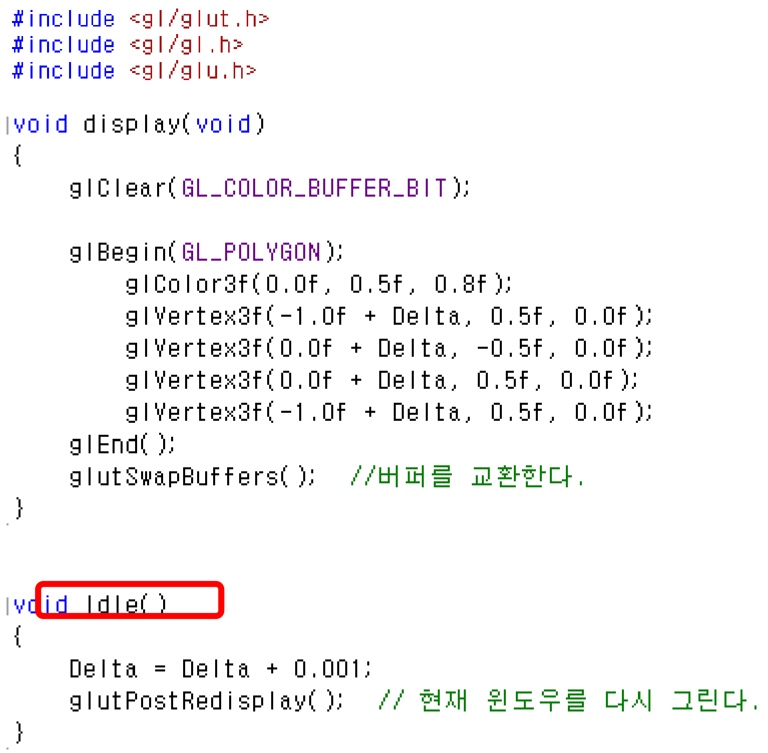




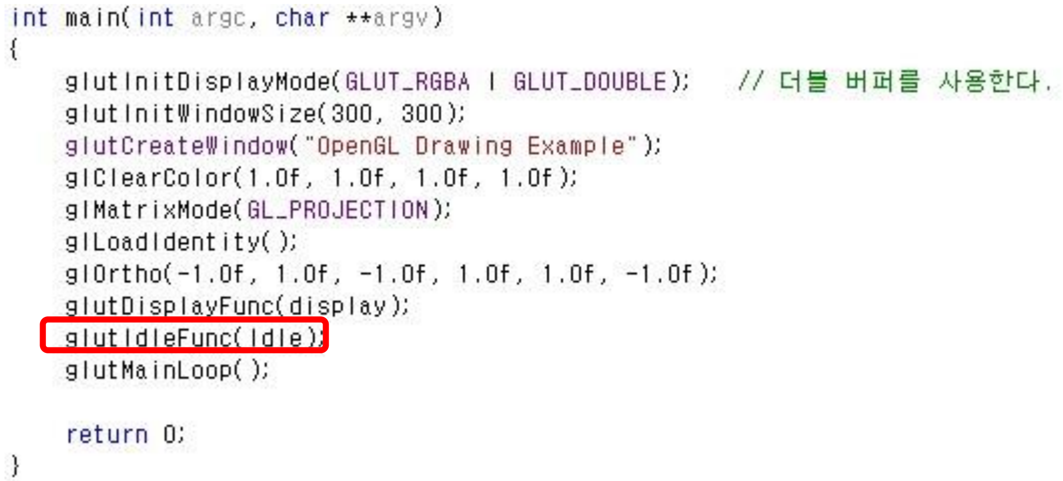
Idle Callback Function

- void **glutIdleFunc**(void (*func)(void))
  - Called when there are no events to be processed
    → idle time

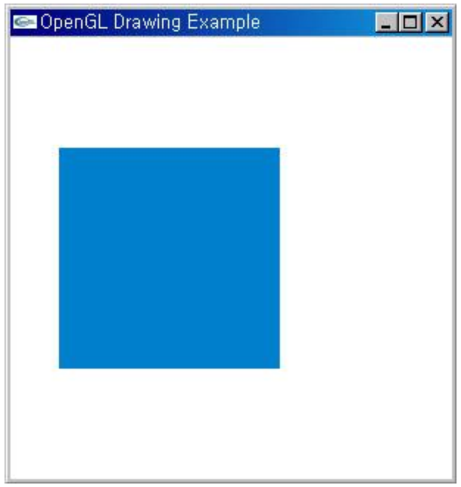
glutIdleFunc()

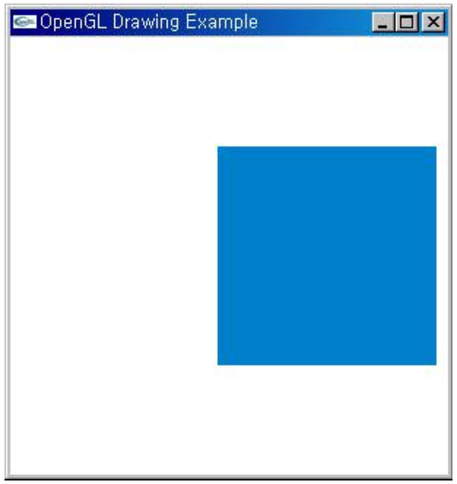


```
#include <gl/glut.h>
#include <gl/gl.h>
#include <gl/glu.h>

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_POLYGON);
        glColor3f(0.0f, 0.5f, 0.8f);
        glVertex3f(-1.0f + Delta, 0.5f, 0.0f);
        glVertex3f(0.0f + Delta, -0.5f, 0.0f);
        glVertex3f(0.0f + Delta, 0.5f, 0.0f);
        glVertex3f(-1.0f + Delta, 0.5f, 0.0f);
    glEnd();
    glutSwapBuffers();  //버퍼를 교환한다.
}


void Idle()
{
    Delta = Delta + 0.001;
    glutPostRedisplay();  // 현재 윈도우를 다시 그린다.
}
```



```
int main(int argc, char **argv)
{
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);   // 더블 버퍼를 사용한다.
    glutInitWindowSize(300, 300);
    glutCreateWindow("OpenGL Drawing Example");
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f);
    glutDisplayFunc(display);
    glutIdleFunc(Idle);
    glutMainLoop();

    return 0;
}
```





Timer Function Callback Function

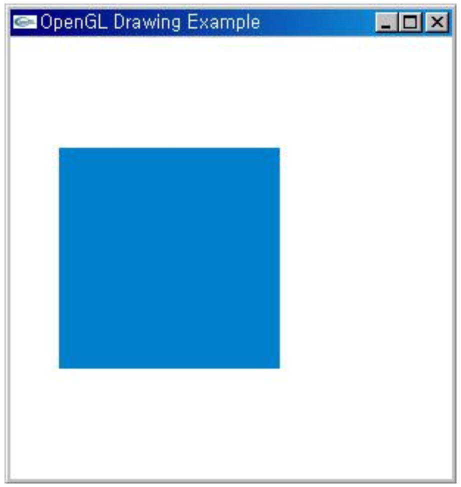
- void **glutTimerFunc**
  (unsigned int msecs, void (*func)(int value), value)
  - Called in every **msecs**
    - msecs: milliseconds
    - value: user defined value

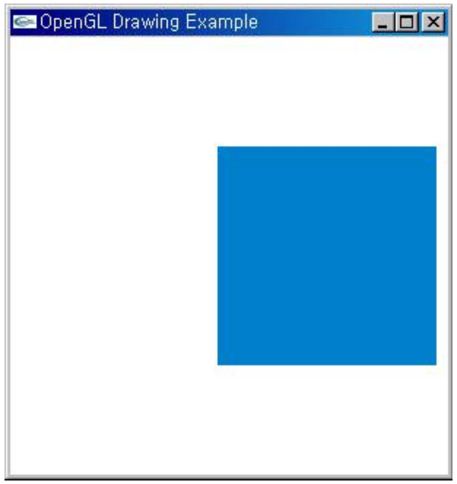
glutTimerFunc()

```
#include <gl/glut.h>
#include <gl/gl.h>
#include <gl/glu.h>

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_POLYGON);
        glColor3f(0.0f, 0.5f, 0.8f);
        glVertex3f(-1.0f + Delta, 0.5f, 0.0f);
        glVertex3f(0.0f + Delta, -0.5f, 0.0f);
        glVertex3f(0.0f + Delta, 0.5f, 0.0f);
        glVertex3f(-1.0f + Delta, 0.5f, 0.0f);
    glEnd();
    glutSwapBuffers();  //버퍼를 교환한다.
}


void TimerFunc(int value)
{
    Delta = Delta + 0.001;


    glutPostRedisplay();
    glutTimerFunc(1, TimerFunc, 1);
}
```

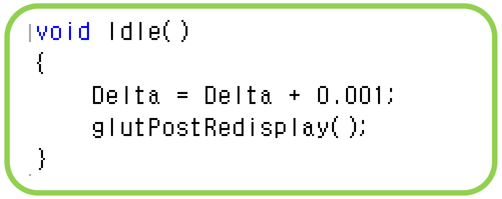
타이머 재등록

```
int main(int argc, char **argv)
{
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);   // 더블 버퍼를 사용한다.
    glutInitWindowSize(300, 300);
    glutCreateWindow("OpenGL Drawing Example");
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f);
    glutDisplayFunc(display);
    glutTimerFunc(100, TimerFunc, 1);
    glutMainLoop();

    return 0;
}
```



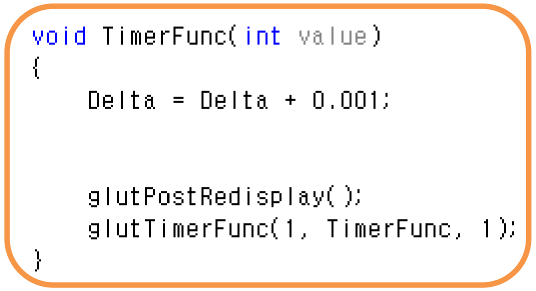


Various Way to Make Animation



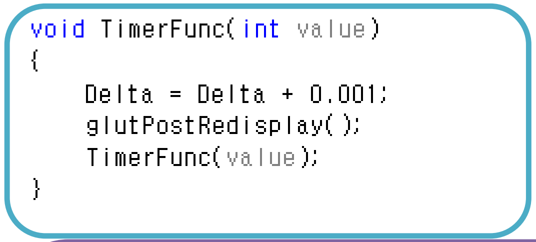
```
void Idle()
{
    Delta = Delta + 0.001;
    glutPostRedisplay();
}
```

Idle Time에만 작동



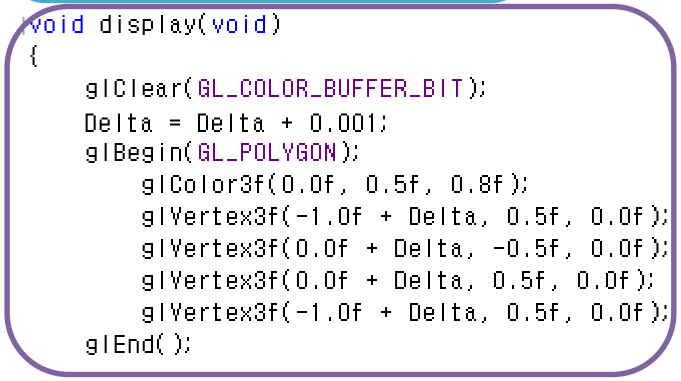
```
void TimerFunc(int value)
{
    Delta = Delta + 0.001;


    glutPostRedisplay();
    glutTimerFunc(1, TimerFunc, 1);
}
```

1 millisecond마다 작동



```
void TimerFunc(int value)
{
    Delta = Delta + 0.001;
    glutPostRedisplay();
    TimerFunc(value);
}
```

정상 작동 하지 않음



```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    Delta = Delta + 0.001;
    glBegin(GL_POLYGON);
        glColor3f(0.0f, 0.5f, 0.8f);
        glVertex3f(-1.0f + Delta, 0.5f, 0.0f);
        glVertex3f(0.0f + Delta, -0.5f, 0.0f);
        glVertex3f(0.0f + Delta, 0.5f, 0.0f);
        glVertex3f(-1.0f + Delta, 0.5f, 0.0f);
    glEnd();
```

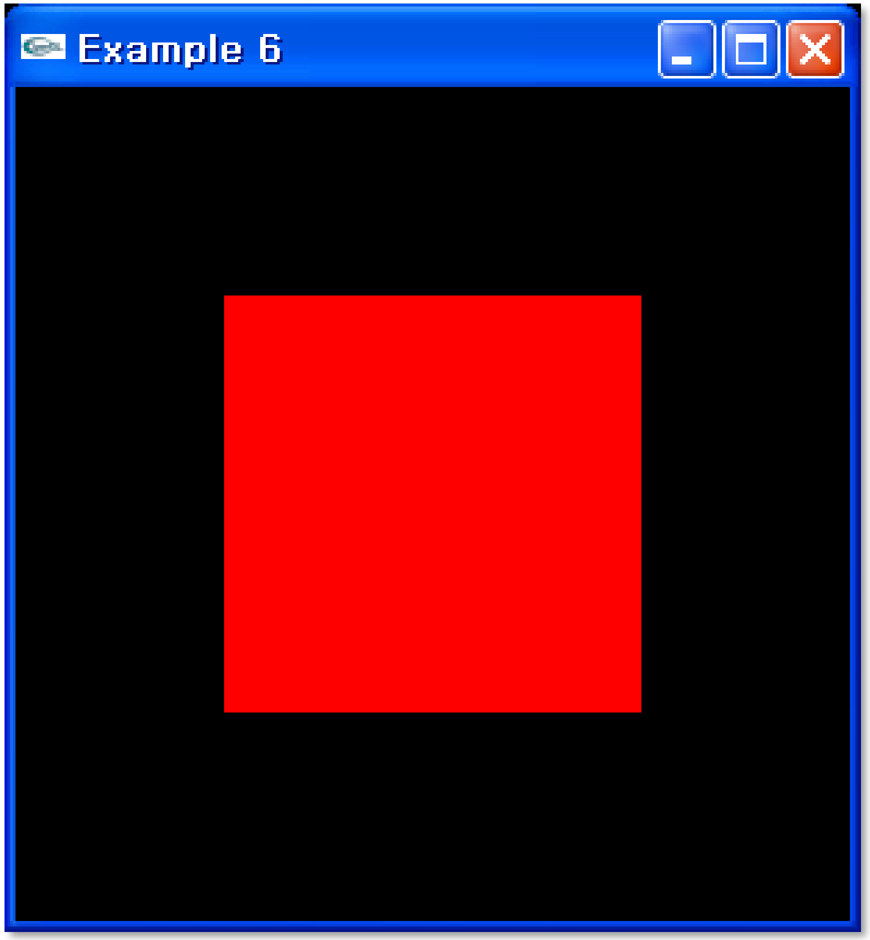
매 프레임 마다

Keyboard Callback Function

- void **glutKeyboardFunc**(void(*func)(unsigned char key, int x, int y))
  - Called when a key is pressed
    - key : ASCII code of the pressed key
    - (x, y) : the coordinate of a mouse

glutKeyboardFunc()

```
#include <glut.h>

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_POLYGON);
        glVertex3f(0.5f , -0.5f, 0.0f);
        glVertex3f(0.5f , 0.5f, 0.0f);
        glVertex3f(-0.5f , 0.5f, 0.0f);
        glVertex3f(-0.5f , -0.5f, 0.0f);
    glEnd( );
    glFlush( );
}
void onKeyPress(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 'Q' :
        case 'q' :
        case 27 :  // ESC
            exit(0);
            break;
    }
}
void main(int argc, char **argv)
{
    glutInitWindowSize(300, 300);
    glutCreateWindow("Example 6");
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glutDisplayFunc(display);
    glutKeyboardFunc(onKeyPress);
    glutMainLoop( );

}
```

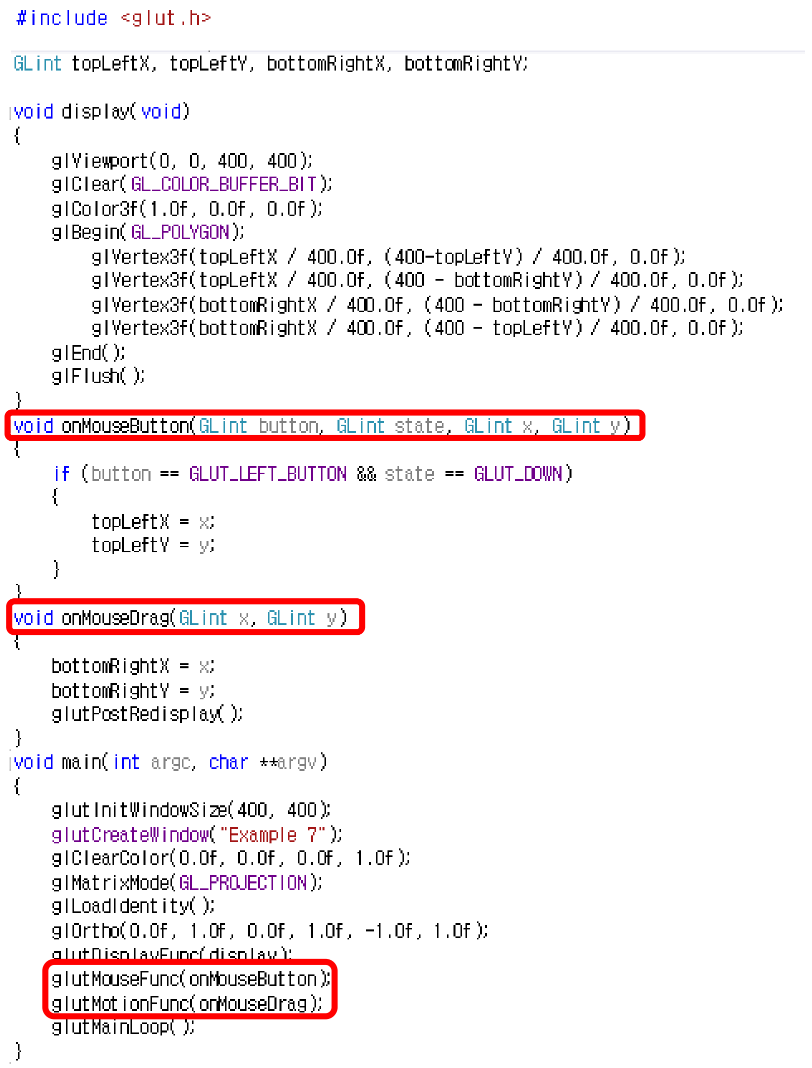


Mouse Callback Functions

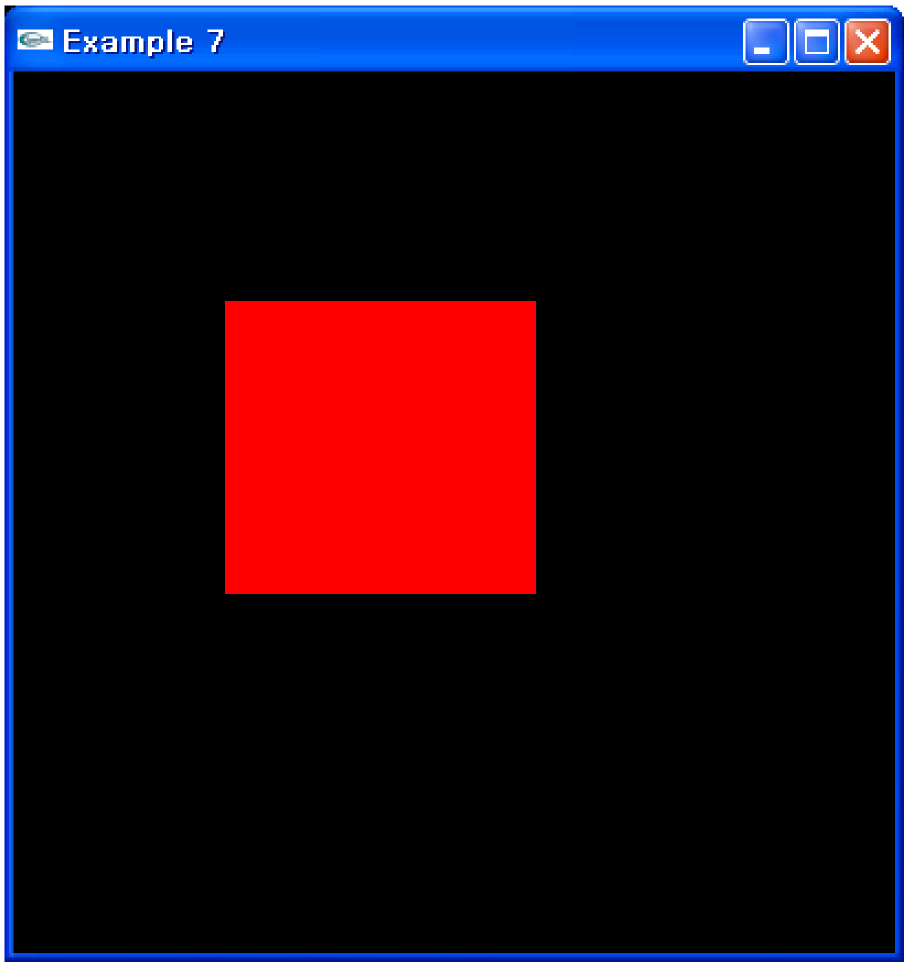
- void **glutMouseFunc**(void(*func)(int button, int state, int x, int y))
  - Called when a mouse button is pressed
    - button
      - GLUT_LEFT_BUTTON , GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON
    - state
      - GLUT_DOWN, GLUT_UP
    - (x, y) : the coordinate of a mouse

- void **glutMotionFunc**(void(*func)(int x, int y))
  - Called when a mouse is dragged
- void **glutPassiveMotionFunc**(void(*func)(int x, int y))
  - Called when a mouse is moving without being pressed

glutMouseFunc() / glutMotionFunc()



```
#include <glut.h>

GLint topLeftX, topLeftY, bottomRightX, bottomRightY;

void display(void)
{
    glViewport(0, 0, 400, 400);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_POLYGON);
        glVertex3f(topLeftX / 400.0f, (400-topLeftY) / 400.0f, 0.0f);
        glVertex3f(topLeftX / 400.0f, (400 - bottomRightY) / 400.0f, 0.0f);
        glVertex3f(bottomRightX / 400.0f, (400 - bottomRightY) / 400.0f, 0.0f);
        glVertex3f(bottomRightX / 400.0f, (400 - topLeftY) / 400.0f, 0.0f);
    glEnd();
    glFlush();
}
void onMouseButton(GLint button, GLint state, GLint x, GLint y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        topLeftX = x;
        topLeftY = y;
    }
}
void onMouseDrag(GLint x, GLint y)
{
    bottomRightX = x;
    bottomRightY = y;
    glutPostRedisplay();
}
void main(int argc, char **argv)
{
    glutInitWindowSize(400, 400);
    glutCreateWindow("Example 7");
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0f, 1.0f, 0.0f, 1.0f, -1.0f, 1.0f);
    glutDisplayFunc(display);
    glutMouseFunc(onMouseButton);
    glutMotionFunc(onMouseDrag);
    glutMainLoop();
}
```



X Window Event handler

Event Handler

- **GLX attaches context to X-window(X-lib)**
  - So, we will handle window events with X-lib
  - Open File “EventHandle.cpp”

Event Types

- **X-lib can handle many events, but we will discuss some useful event handler.**

Event Category	Event Type
Keyboard events	KeyPress, KeyRelease
Pointer(Mouse) events	ButtonPress, ButtonRelease, MotionNotify
Structure(Window) control events	ConfigureNotify

  - Event type is classified by **enumerated type**

Event Masking

- **Clients must select event reporting of most events relative to a window.**

Event Mask	Circumstances
KeyPressMask	Keyboard down events wanted
KeyReleaseMask	Keyboard up events wanted
ButtonPressMask	Pointer button down events wanted
ButtonReleaseMask	Pointer button up events wanted
PointerMotionMask	Pointer motion events wanted
StructureNotifyMask	Any change in window structure wanted

Event Masking Example

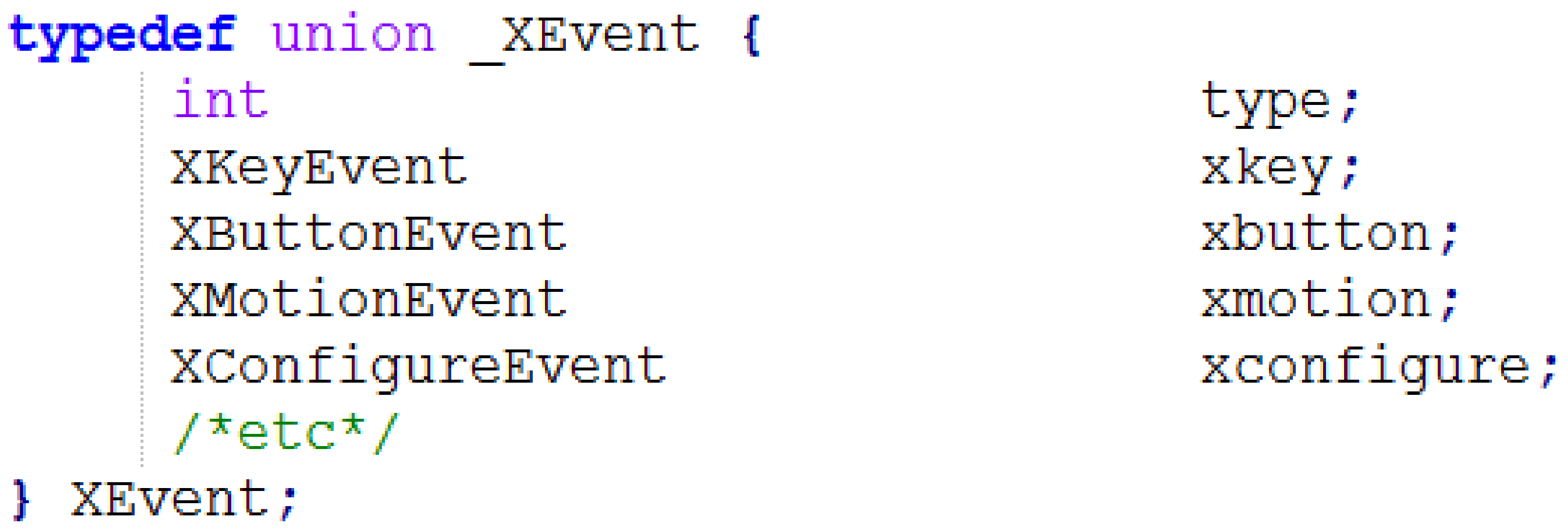
- If you want to report some events, you need to set event mask at **XSetWindowAttributes**

```
XSetWindowAttributes swa;
int main(int argc, char *argv[]) {
        /*Create Window    Start */
        swa.event_mask = KeyPressMask|KeyReleaseMask;
        /*Create Window     Cont....  */

        Xevent    xev;
        while(1){
                display();
                XNextEvent(dpy, &xev);
        }
}
```

  - This code reports only key press and release events

Event Union

- Every event are received by **XEvent** union(structure).



```
typedef union _XEvent {
        int                                   type;
        XKeyEvent                             xkey;
        XButtonEvent                          xbutton;
        XMotionEvent                          xmotion;
        XConfigureEvent                       xconfigure;
        /*etc*/
} XEvent;
```

C Union Types

- A **union** is a special data type available in C that allows to store different data types in the same memory location.

```
struct structJob
{
    char name[32];
    float salary;
    int workerNo;
} sJob;
```

```
union unionJob
{
    //defining a union
    char name[32];
    float salary;
    int workerNo;
} uJob;
```

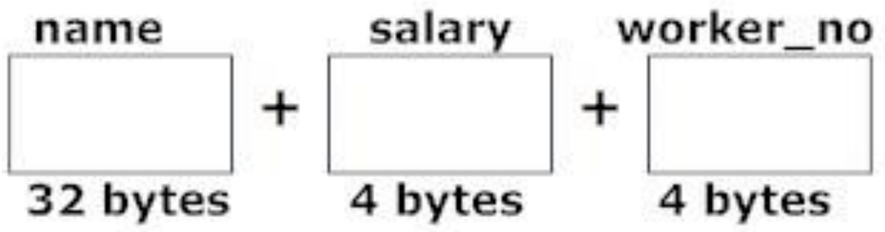


Fig: Memory allocation in case of structure

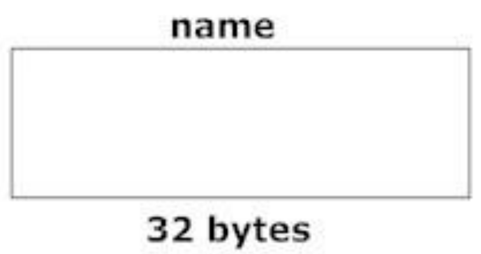


Fig: Memory allocation in case of union

Structure vs. Union Example

```
#include <stdio.h>
union unionJob
{
   //defining a union
   char name[32];
   float salary;
   int workerNo;
} uJob;

struct structJob
{
   char name[32];
   float salary;
   int workerNo;
} sJob;
```

```
int main()
{
   printf("size of union = %d", sizeof(uJob));
   printf("₩nsize of structure = %d", sizeof(sJob));
   return 0;
}
```

Result Print:

```
size of union = 32
size of structure = 40
```

Union Application Example

- **Only one union member can be accessed at a time**

```
#include <stdio.h>
union job
{
    char name[32];
    float salary;
    int workerNo;
} job1;
```

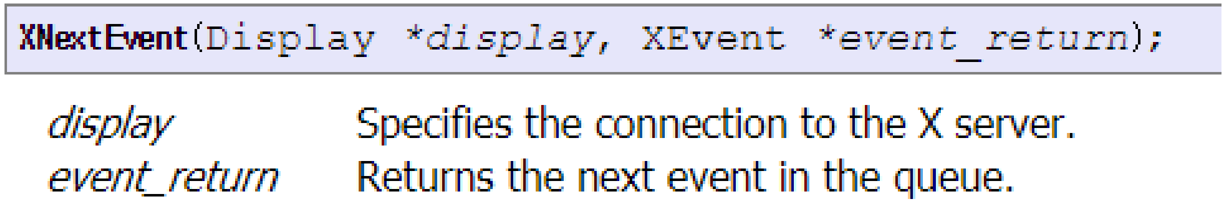
```
int main(){
    printf("Enter name:₩n");
    scanf("%s", &job1.name);
    printf("Enter salary: ₩n");
    scanf("%f", &job1.salary);
    printf("Displaying₩nName :%s₩n", job1.name);
    printf("Salary: %.1f", job1.salary);
    return 0;
}
```

Result Print:

```
Enter name
Hillary
Enter salary
1234.23
Displaying
Name: f%Bary
Salary: 1234.2
```

Get the Event

- You need to pop the event from event queue to handle the events



```
XNextEvent(Display *display, XEvent *event_return);
```

display	Specifies the connection to the X server.
event_return	Returns the next event in the queue.

```
/*Create Window*/
Xevent xev;
while(1){
        display();
        XNextEvent(dpy, &xev);
}
```

Get the Event Type

- You can get the event type by XEvent structure

```
/*Create Window*/
Xevent xev;
while(1){
		display();
		XNextEvent(dpy, &xev);
		if(xev.type == KeyPress){ /*do something*/}
		else if(xev.type == KeyRelease) {/*do something*/}
		else if(xev.type == /*etc*/) {/*do something*/}
}
```

Window Event: XConfigureEvent

- **XConfigureEvent is reported when window configure is changed.**
  - Move
  - Resize
  - Etc.

- **XConfigureEvent contains window information**
  - int x, y; //window position
  - int width, height; //window size

- **Mask : StructureNotifyMask**
- **Type : ConfigureNotify**

X-Window Event Check Code

```
swa.event_mask = StructureNotifyMask;

…………

XNextEvent(dpy, &xev);
if(xev.type == ConfigureNotify){
            printf("ConfigureNotify₩n");
            printf("Window Position: (%d,%d)₩n",
                        xev.xconfigure.x,xev.xconfigure.y);
            printf("Window Size: (%d,%d)₩n",
                        xev.xconfigure.width,xev.xconfigure.height);
}
```

KeyBoard Event: XKeyEvent

- **XKeyEvent is reported when keyboard is pressed or released**

- **XKeyEvent contains key code information**
  - unsigned int keycode;

- **Mask : KeyPressMask, KeyReleaseMask**
- **Type : KeyPress, KeyRelease**

KeyBoard Event Check Code

```
swa.event_mask = KeyPressMask | KeyReleaseMask;

…………

XNextEvent(dpy, &xev);
if(xev.type == KeyPress){
            printf("KeyPress Call₩n");
            printf("[%s] Key is Pressed₩n",
            XKeysymToString(XkbKeycodeToKeysym(dpy, xev.xkey.keycode, 0, 0)));
}else if(xev.type == KeyRelease){
            printf("KeyRelease Call₩n");
            printf("[%s] Key is Released₩n",
            XKeysymToString(XkbKeycodeToKeysym(dpy, xev.xkey.keycode, 0, 0)));
}
```

1st “0” in **XkbKeycodeToKeysym** is Key Group

2nd “0” in **XkbKeycodeToKeysym** is Shift Level(Pressed or not)

Mouse Button: XButtonEvent

- XButtonEvent is reported when mouse button is pressed or released

- XButtonEvent contains mouse button and pointer position information
  - int x, y;
  - unsigned int button;

- Mask : ButtonPressMask, ButtonReleaseMask
- Type : ButtonPress, ButtonRelease

Mouse Button Event Check

```
swa.event_mask = ButtonPressMask | ButtonReleaseMask;

............

XNextEvent(dpy, &xev);
if(xev.type == ButtonPress){
            printf("ButtonPress Call₩n");
            printf("Pointer Position: (%d,%d)₩n",xev.xbutton.x,xev.xbutton.y);
            printf("Mouse %d Button is pressed₩n",xev.xbutton.button);
}else if(xev.type == ButtonRelease){
            printf("ButtonRelease Call₩n");
            printf("Pointer Position: (%d,%d)₩n",xev.xbutton.x,xev.xbutton.y);
            printf("Mouse %d Button is Released₩n",xev.xbutton.button);
}
```

Mouse Motion: XMotionEvent

- **XMotionEvent is reported when mouse pointer is moved**

- **XMotionEvent contains mouse pointer position information**
  - int x, y;

- **Mask : PointerMotionMask**
- **Type : MotionNotify**

Mouse Motion Check Code

```
swa.event_mask = PointerMotionMask;

…………

XNextEvent(dpy, &xev);
if(xev.type == MotionNotify){
        printf("MotionNotify Call₩n");
        printf("Pointer Position: (%d,%d)₩n", xev.xmotion.x, xev.xmotion.y);
 }
```

Exit Process with Event Handler

```
 void ExitProgram() {
           glXMakeCurrent(dpy, None, NULL); //Clear display setting
           glXDestroyContext(dpy, glc); //Destroy glx context
           XDestroyWindow(dpy, win); //Destroy the window
           XCloseDisplay(dpy); //Close the display(X-server)
           exit(0);
 }
```

Write Following Code

```
if(xev.type == KeyPress){
          char *key_string = XKeysymToString(XkbKeycodeToKeysym(dpy,
                                    xev.xkey.keycode, 0, 0));
          if(strncmp(key_string, "Escape", 6) == 0) ExitProgram();
}
```

- Run modified program and press “ESC” key
- Window may close without error message

Learning More about XWindow

- https://tronche.com/gui/x/xlib/
- ftp://www.x.org/pub/current/doc/libX11/libX11/libX11.html