3. Input, Interaction and Event Driven Programming

Lecture Overview

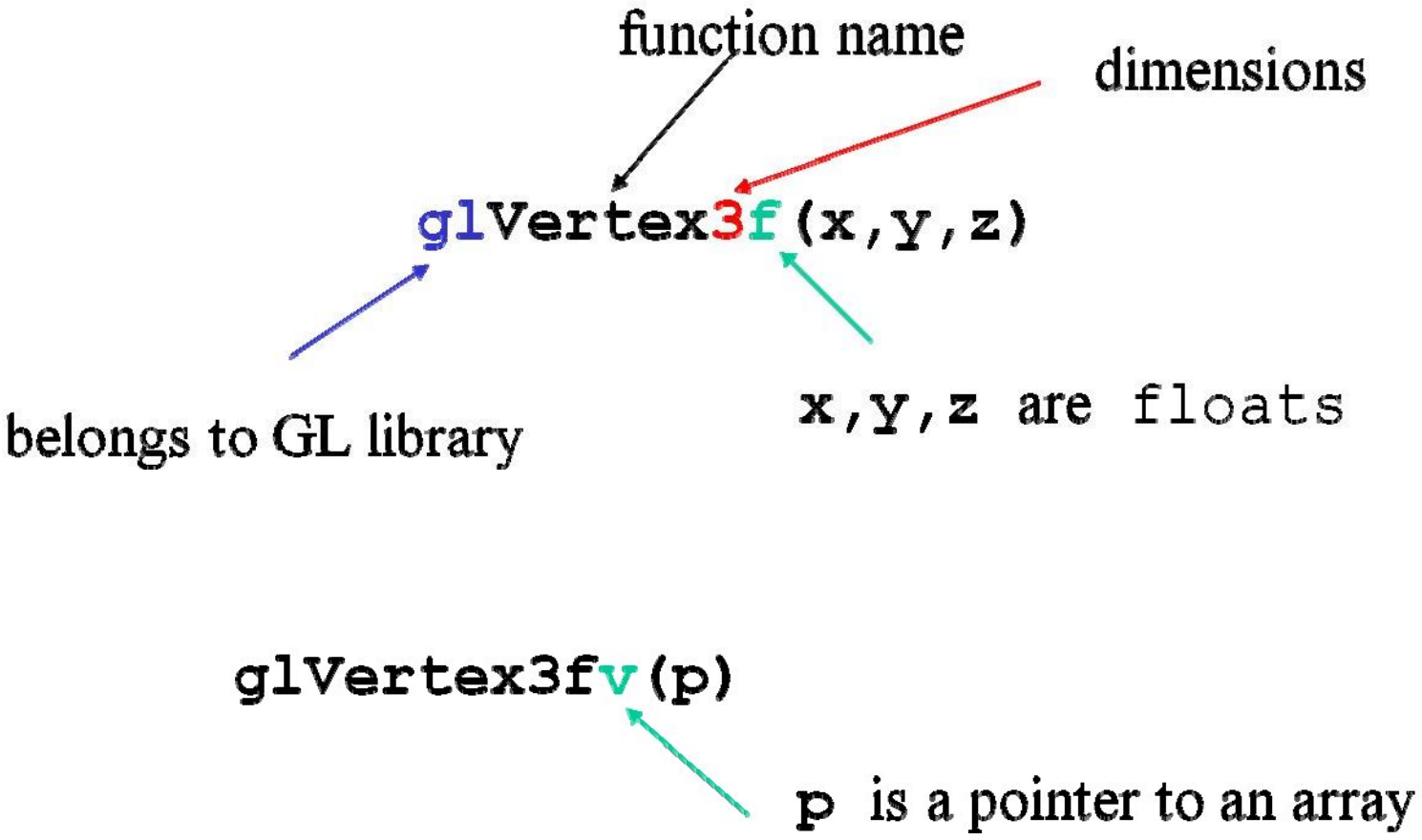
- Recap of Lecture II

- Input and Interaction
- Working with Callbacks
- Better Interactive Programs

- Reading: ANG Ch. 3, except 3.8

Recap of Lecture II

OpenGL function format



Program Structure

- Most OpenGL programs have a similar structure that consists of the following functions
  - **main():**
    - defines the callback functions
    - opens one or more windows with the required properties
    - enters event loop (last executable statement)
  - **init():** sets the state variables
    - Viewing
    - Attributes
  - callbacks
    - Display function
    - Input and window functions

main.c

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

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);

    init();

    glutMainLoop();
}
```

includes **glut.h**

define window properties

display callback

set OpenGL state

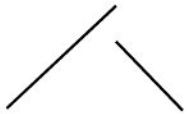
enter event loop

mydisplay.c

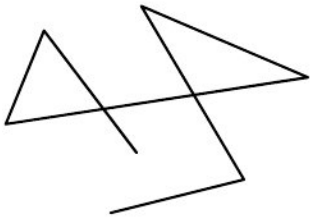
```
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
```

OpenGL Primitives

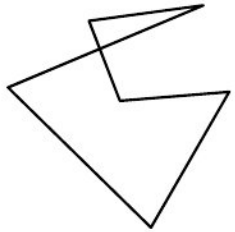
GL_POINTS



GL_LINES

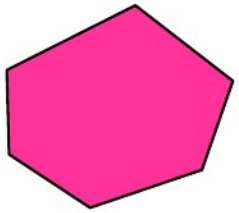


GL_LINE_STRIP

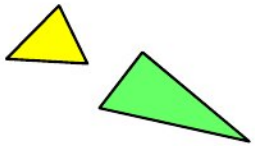


GL_LINE_LOOP

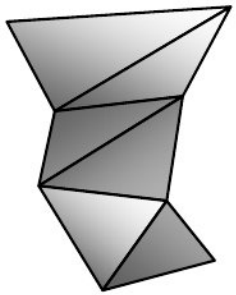
OpenGL Primitives



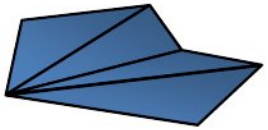
GL_POLYGON



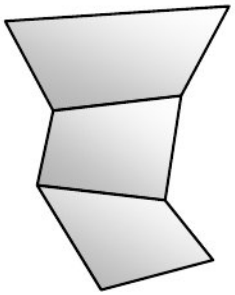
GL_TRIANGLES



GL_TRIANGLE_STRIP

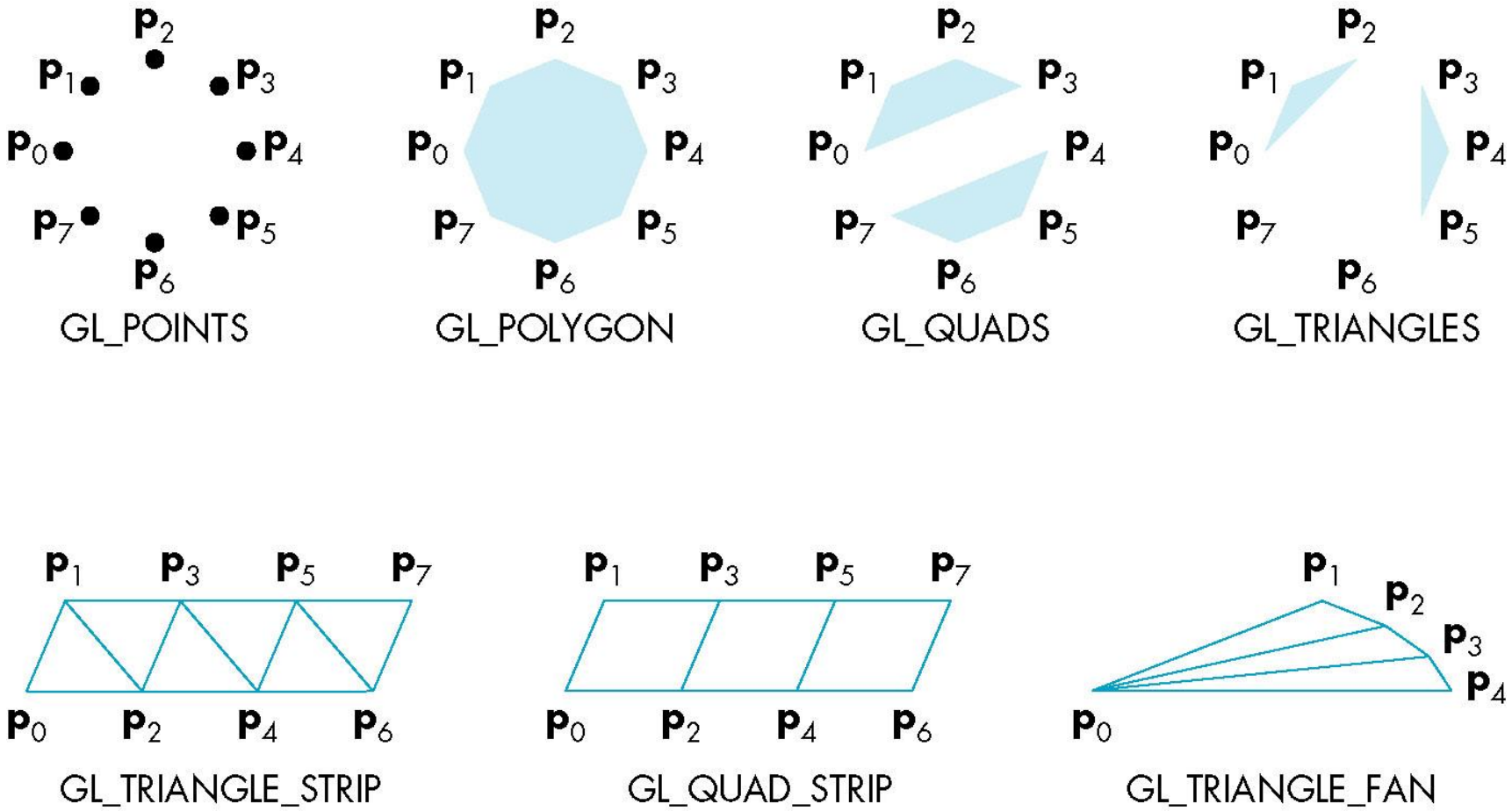


GL_TRIANGLE_FAN



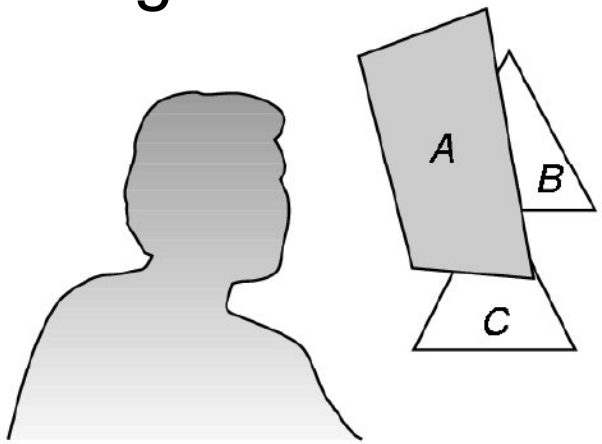
GL_QUAD_STRIP

Polygon Types in OpenGL



Hidden-Surface Removal

- We want to see only those surfaces in front of other surfaces
- OpenGL uses a hidden-surface method called the z-buffer algorithm that saves depth information as objects are rendered so that only the front objects appear in the image



Using the z-buffer algorithm

- The algorithm uses an extra buffer, the z-buffer, to store depth information as geometry travels down the pipeline
- It must be
  - Requested in **main.c**
    - **glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)**
  - Enabled in **init.c**
    - **glEnable(GL_DEPTH_TEST)**
  - Cleared in the display callback
    - **glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)**

Input and Interaction

Objectives

- Introduce the basic input devices
  - Physical Devices
  - Logical Devices
  - Input Modes
- Event-driven input
- Introduce double buffering for smooth animations
- Programming event input with GLUT

Project Sketchpad

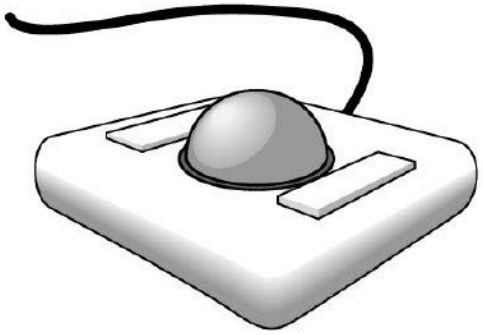
- Ivan Sutherland (MIT 1963) established the basic interactive paradigm that characterizes interactive computer graphics:
  - User sees an object on the display
  - User points to (picks) the object with an input device (light pen, mouse, trackball)
  - Object changes (moves, rotates, morphs)
  - Repeat

Graphical Input

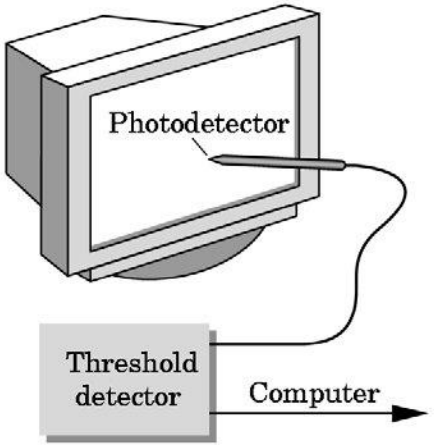
- Devices can be described either by
  - Physical properties
    - Mouse
    - Keyboard
    - Trackball
  - Logical Properties
    - What is returned to program via API
      - A position
      - An object identifier
- Modes
  - How and when input is obtained
    - Request or event

Physical Devices

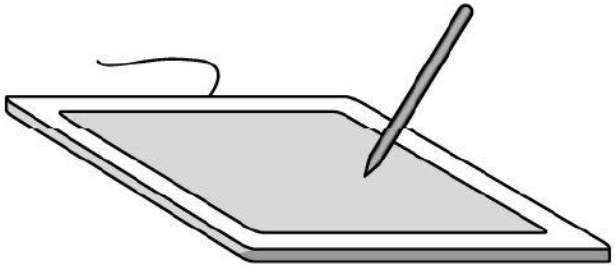
mouse



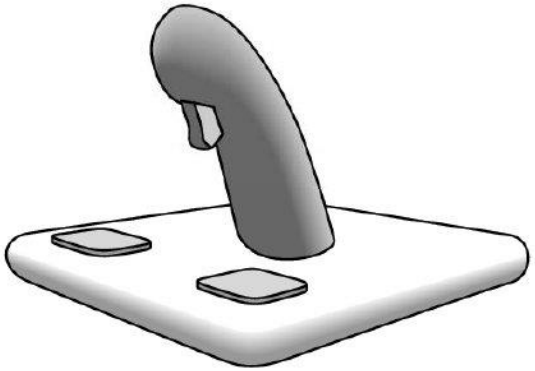
trackball



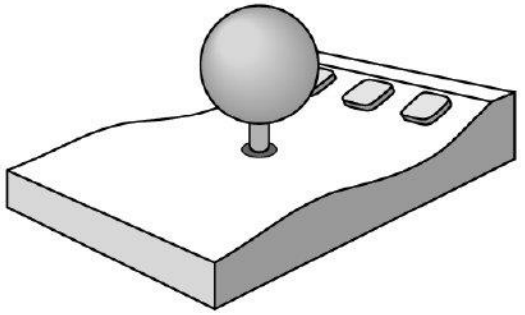
light pen



data tablet

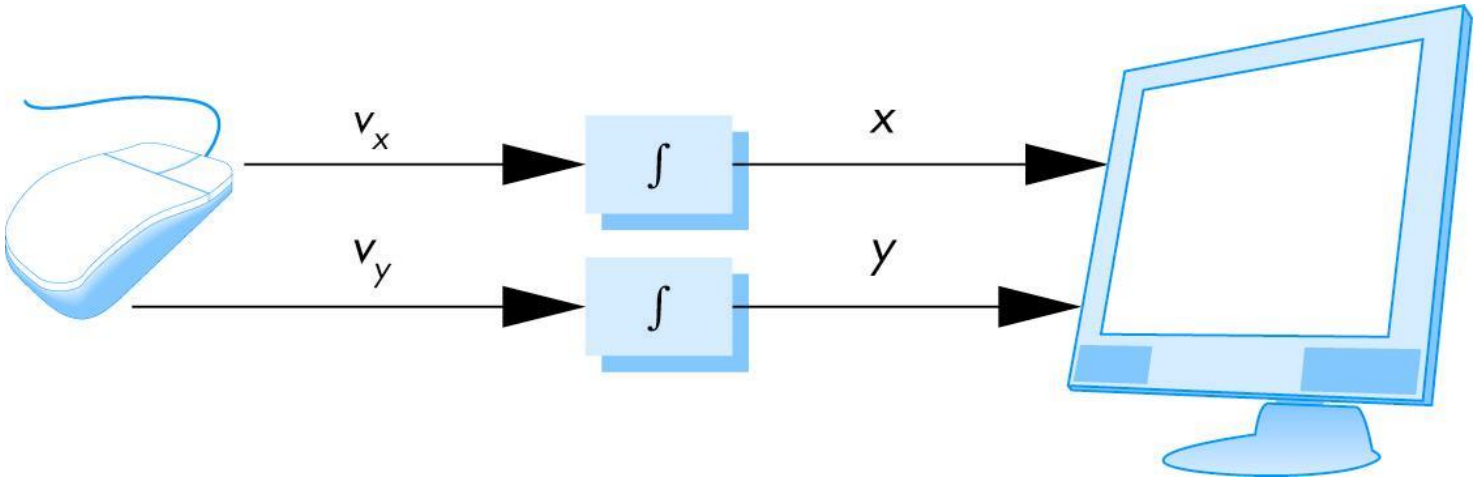


joy stick



space ball

Cursor Positioning



Incremental (Relative) Devices

- Devices such as the data tablet return an *absolute position* directly to the operating system
- Devices such as the mouse, trackball, and joy stick return *incremental inputs* (or velocities) to the operating system
  - Must integrate these inputs to obtain an absolute position
    - Rotation of cylinders in mouse
    - Roll of trackball
    - Difficult to obtain absolute position
    - Can get variable sensitivity (joysticks)

Degrees of Freedom

- Two d.o.f.: mouse, trackball, joystick, lightpen
- Three d.o.f.: spaceball, laser scanner, motion capture (mocap), data gloves (could be 4D), magnetic trackers
- Keyboard input

Logical Devices

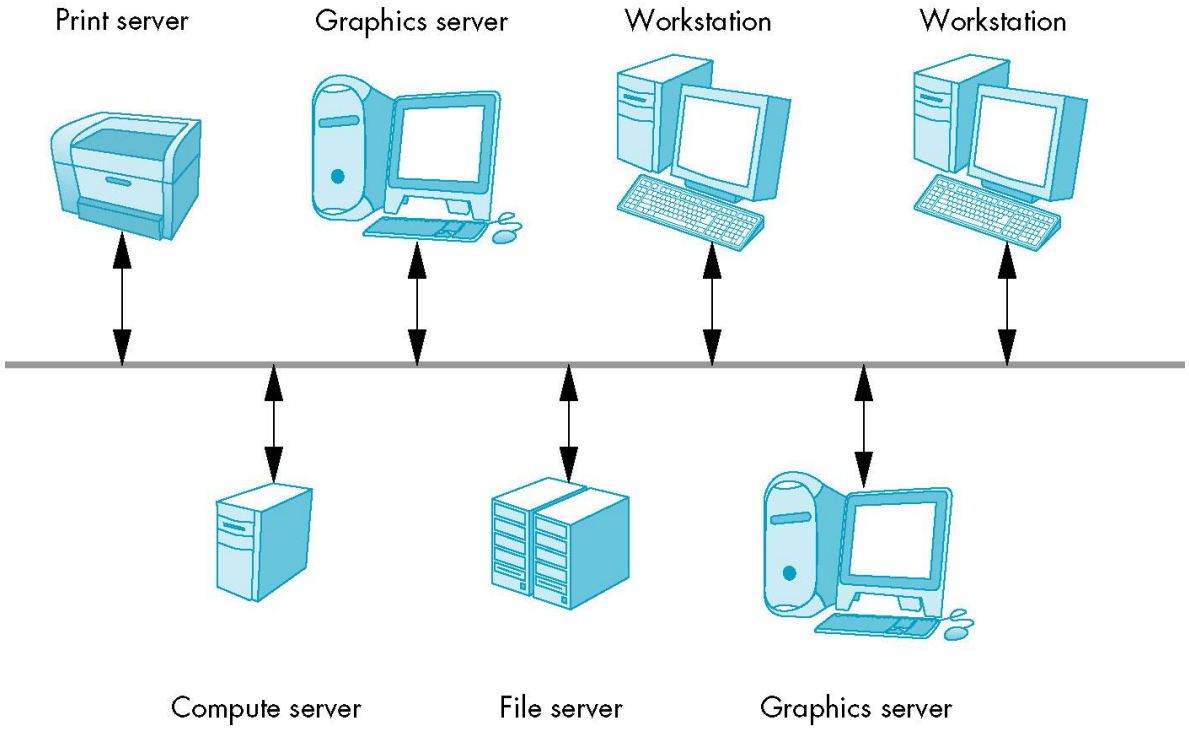
- Consider the C and C++ code
  - C++: **cin >> x;**
  - C: **scanf ("%d", &x);**
- What is the input device?
  - Cannot tell from the code
  - Could be keyboard, file, output from another program
- The code provides logical input
  - A number (an **int**) is returned to the program regardless of the physical device

Graphical Logical Devices

- Graphical input is more varied than input to standard programs which is usually numbers, characters, or bits
- Two older APIs (GKS, PHIGS) defined six types of logical input
  - Locator: return a position
  - Pick: return ID of an object
  - Keyboard or String: return strings of characters
  - Stroke: return array of positions
  - Valuator: return floating point number (widgets: slidebars)
  - Choice: return one of n items (widgets: menus, buttons)

X Window Input

- The X Window System introduced a client-server model for a network of workstations
- Client-server model applies to single user system
  - Client: OpenGL program
  - Graphics Server: bitmap display with a pointing device and a keyboard



Input Modes

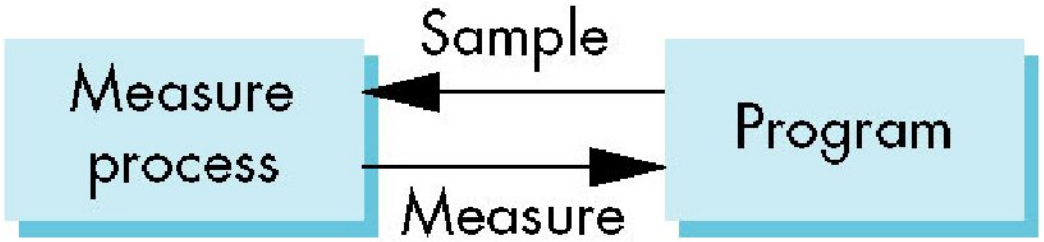
- Input devices contain a trigger which can be used to send a signal to the operating system
  - Button on mouse
  - Pressing or releasing a key
- When triggered, input devices return information (their measure) to the system
  - Mouse returns position information
  - Keyboard returns ASCII code

Request Mode

- Input provided to program only when user triggers the device
- Typical of keyboard input
  - Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed

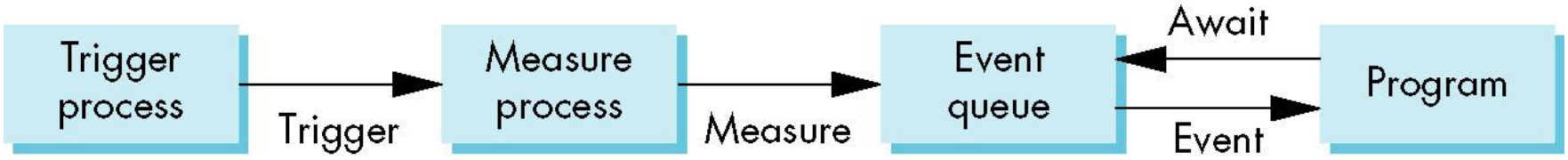
Sample Mode

- Input is immediate, no trigger necessary
- Users must have positioned pointing device or entered data using keyboard before the function is called



Event Mode

- Most systems have more than one input device, each of which can *be triggered at an arbitrary time by a user*
- Each trigger generates an *event* whose measure is put in an *event queue* which can be examined by the user program



Event Types

- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: nonevent
  - Define what should be done if no other event is in queue

Callbacks

- Programming interface for event-driven input
- Define a *callback function* for each type of event the graphics system recognizes
- This user-supplied function is executed when the event occurs
- GLUT example:
  glutMouseFunc(mymouse)

mouse callback function

GLUT callbacks

GLUT recognizes a subset of the events recognized by any particular window system (Windows, X, Macintosh)

- **glut**Display**Func**
- **glut**Mouse**Func**
- **glut**Reshape**Func**
- **glut**Keyboard**Func**
- **glut**Idle**Func**
- **glut**Motion**Func, glut**PassiveMotion**Func**

GLUT Event Loop

- Recall that *the last line in* ***main.c*** for a program using GLUT must be

  glutMainLoop();

which puts the program in an infinite event loop

- In each pass through the event loop, GLUT
  - looks at the events in the queue
  - for each event in the queue, GLUT executes the appropriate callback function if one is defined
  - if no callback is defined for the event, the event is ignored

The display callback

- The display callback is executed whenever GLUT determines that the window should be refreshed, for example
  - When the window is first opened
  - When the window is reshaped
  - When a window is exposed
  - When the user program decides it wants to change the display
- In **main.c**
  - **glutDisplayFunc(mydisplay)** identifies the function to be executed
  - Every GLUT program must have a display callback

Posting redisplays

- Many events may invoke the display callback function
  - Can lead to multiple executions of the display callback on a single pass through the event loop
- We can avoid this problem by instead using

  glutPostRedisplay();

  which sets a flag.
- GLUT checks to see if the flag is set at the end of the event loop
- If set then the display callback function is executed

Animating a Display

- When we redraw the display through the display callback, we usually start by clearing the window
  - **glClear()**

then draw the altered display

- Problem: the drawing of information in the frame buffer is decoupled from the display of its contents
  - Graphics systems use dual ported memory
- Hence we can see partially drawn display
  - See the program **single_double.c** for an example with a rotating cube

Double Buffering

- Instead of one color buffer, we use two
  - Front Buffer: one that is displayed but not written to
  - Back Buffer: one that is written to but not displayed
- Program then requests a double buffer in main.c
  - **glutInitDisplayMode(GL_RGB | GL_DOUBLE)**
  - At the end of the display callback buffers are swapped

```
void mydisplay()
{
        glClear(GL_COLOR_BUFFER_BIT|….)
.
/* draw graphics here */
.
        glutSwapBuffers()
}
```

Using the idle callback

- The idle callback is executed whenever there are no events in the event queue
  - **glutIdleFunc(myidle)**
  - Useful for animations

```
void myidle() {
/* change something */
        t += dt
        glutPostRedisplay();
}

Void mydisplay() {
        glClear();
/* draw something that depends on t */
        glutSwapBuffers();
}
```

Using globals

- The form of all GLUT callbacks is fixed
  - void **mydisplay()**
  - void **mymouse(GLint button, GLint state, GLint x, GLint y)**
- Must use globals to pass information to callbacks

```
float t; /*global */

void mydisplay()
{
/* draw something that depends on t
}
```

Working with Callbacks

Objectives

- Learn to build interactive programs using GLUT callbacks
  - Mouse
  - Keyboard
  - Reshape
- Introduce menus in GLUT

The mouse callback

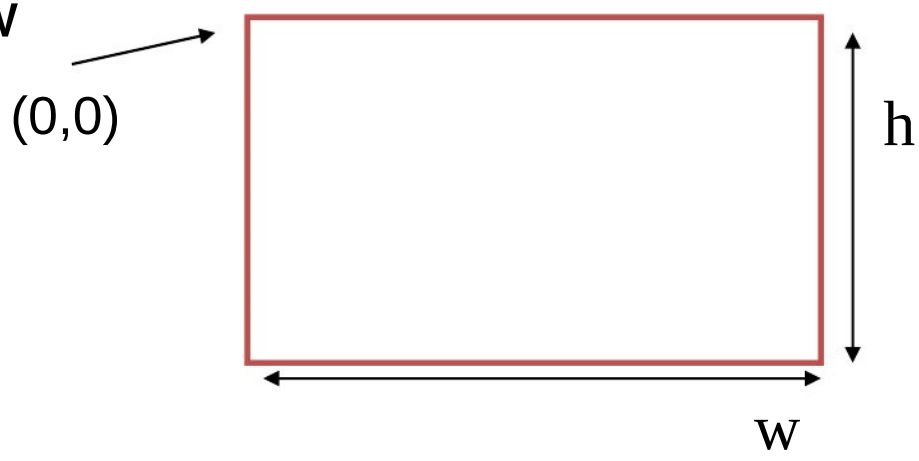
glutMouseFunc(mymouse)

void mymouse(GLint button, GLint state, GLint x, GLint y)

- Returns
  - which button (**GLUT_LEFT_BUTTON**, **GLUT_MIDDLE_BUTTON**, **GLUT_RIGHT_BUTTON**) caused event
  - state of that button (**GLUT_UP**, **GLUT_DOWN**)
  - Position in window

Positioning

- The position in the screen window is usually measured in pixels with the origin at the top-left corner
  - Consequence of refresh done from top to bottom
- OpenGL uses a world coordinate system with origin at the bottom left
  - Must invert y coordinate returned by callback by height of window
  - y = h – y;



Obtaining the window size

- To invert the y position we need the window height
  - Height can change during program execution
  - Track with a global variable
  - New height returned to reshape callback that we will look at in detail soon
  - Can also use query functions
    - **glGetIntv**
    - **glGetFloatv**

  to obtain any value that is part of the state

Terminating a program

- In our original programs, there was no way to terminate them through OpenGL
- We can use the simple mouse callback

```
void mouse(int btn, int state, int x, int y)
{
   if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        exit(0);
 }
```

Using the mouse position

- In the next example, we draw a small square at the location of the mouse each time the left mouse button is clicked
- This example does not use the display callback but one is required by GLUT
  - We can use the empty display callback function

```
mydisplay(){}
```

Drawing squares at cursor location

```
void mymouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
          exit(0);
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        drawSquare(x, y);
 }
void drawSquare(int x, int y)
{
      y=w-y; /* invert y position */
       glColor3ub( (char) rand()%256, (char) rand )%256,
          (char) rand()%256); /* a random color */
      glBegin(GL_POLYGON);
            glVertex2f(x+size, y+size);
            glVertex2f(x-size, y+size);
            glVertex2f(x-size, y-size);
            glVertex2f(x+size, y-size);
      glEnd();
 }
```

Using the motion callback

- We can draw squares (or anything else) continuously as long as a mouse button is depressed by using the motion callback
  - **glutMotionFunc(drawSquare)**
- We can draw squares without depressing a button using the passive motion callback
  - **glutPassiveMotionFunc(drawSquare)**

Using the keyboard

glutKeyboardFunc(mykey)

void mykey(unsigned char key, int x, int y)

– Returns ASCII code of key depressed and mouse location

```
void mykey()
{
        If (key == 'Q' | key == 'q')
                exit(0);
}
```

Special and Modifier Keys

- GLUT defines the special keys in **glut.h**
  - Function key 1: **GLUT_KEY_F1**
  - Up arrow key: **GLUT_KEY_UP**
    - **if (key == GLUT_KEY_F1 .....**
- Can also check of one of the modifiers
  - **GLUT_ACTIVE_SHIFT**
  - **GLUT_ACTIVE_CTRL**
  - **GLUT_ACTIVE_ALT**

  is depressed by

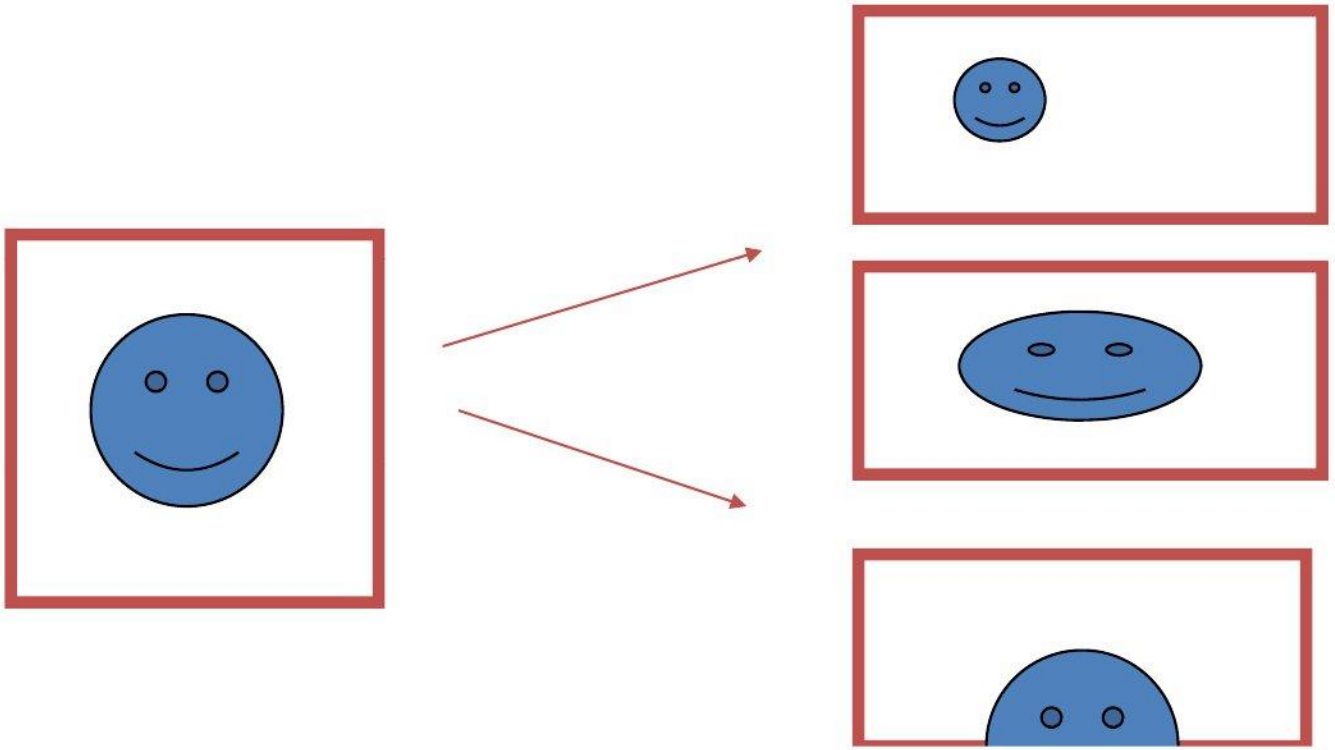
  glutGetModifiers()

  - Allows emulation of three-button mouse with one- or two-button mice

Reshaping the window

- We can reshape and resize the OpenGL display window by pulling the corner of the window
- What happens to the display?
  - Must redraw from application
  - Two possibilities
    - Display part of world
    - Display whole world but force to fit in new window
      - Can alter aspect ratio

Reshape possiblities



original

reshaped

The Reshape callback

glutReshapeFunc(myreshape)

void myreshape(int w, int h)

- Returns width and height of new window (in pixels)
- A redisplay is posted automatically at end of execution of the callback
- GLUT has a default reshape callback but you probably want to define your own

- The reshape callback is good place to put viewing functions because it is invoked when the window is first opened

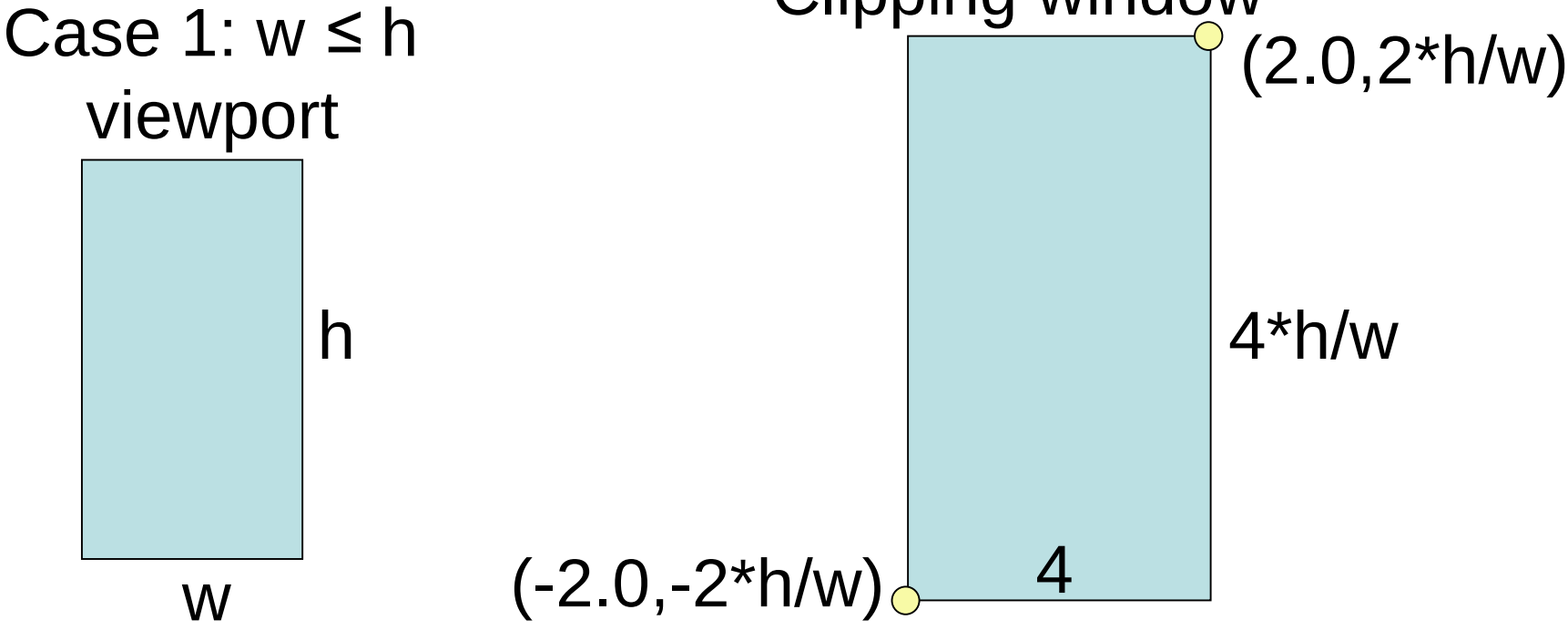
Example Reshape

- This reshape preserves shapes by making the viewport and world window have the same aspect ratio

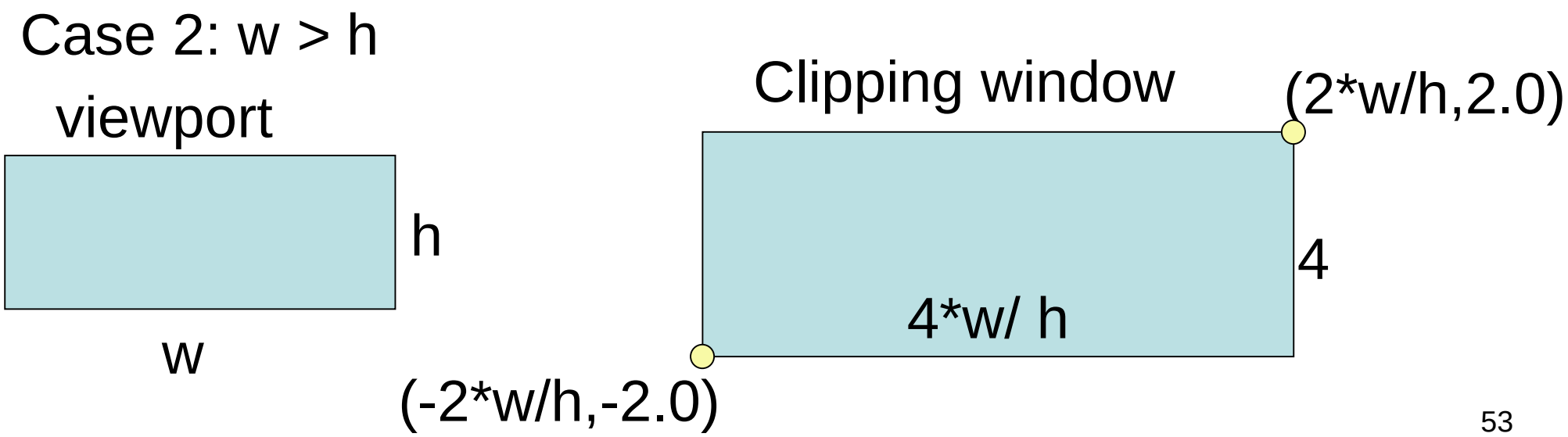
```
void myReshape(int w, int h)
{
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION); /* switch matrix mode */
        glLoadIdentity();
        if (w <= h)
                gluOrtho2D(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,
                        2.0 * (GLfloat) h / (GLfloat) w);
        else  gluOrtho2D(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 *
                        (GLfloat) w / (GLfloat) h, -2.0, 2.0);
        glMatrixMode(GL_MODELVIEW); /* return to modelview mode */
}
```

Reshape Function

Case 1: w ≤ h



Case 2: w > h



Rotating Square Example

- Based on single_double.c from Ed Angel's examples
- Simple animation of rotating square

Rot_Square: Definitions

```
static GLfloat spin = 0.0;
GLfloat x, y;

void square()
{
      glBegin(GL_QUADS);

         glVertex2f(x,y);
         glVertex2f(-y,x);
         glVertex2f(-x,-y);
         glVertex2f(y,-x);

       glEnd();
}
```

Rot_Square: Display

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

    square();

    glutSwapBuffers ();
}
```

Rot_Square: Spin Display

```
void spinDisplay (void)
{
        spin = spin + 1.0;
        if (spin > 360.0) spin = spin - 360.0;
        x= 25.0*cos(DEGREES_TO_RADIANS * spin);
        y= 25.0*sin(DEGREES_TO_RADIANS * spin);
        glutPostRedisplay();
}
```

Toolkits and Widgets

- Most window systems provide a toolkit or library of functions for building user interfaces that use special types of windows called widgets
- Widget sets include tools such as
  - Menus
  - Slidebars
  - Dials
  - Input boxes
- But toolkits tend to be platform dependent
- GLUT provides a few widgets including menus

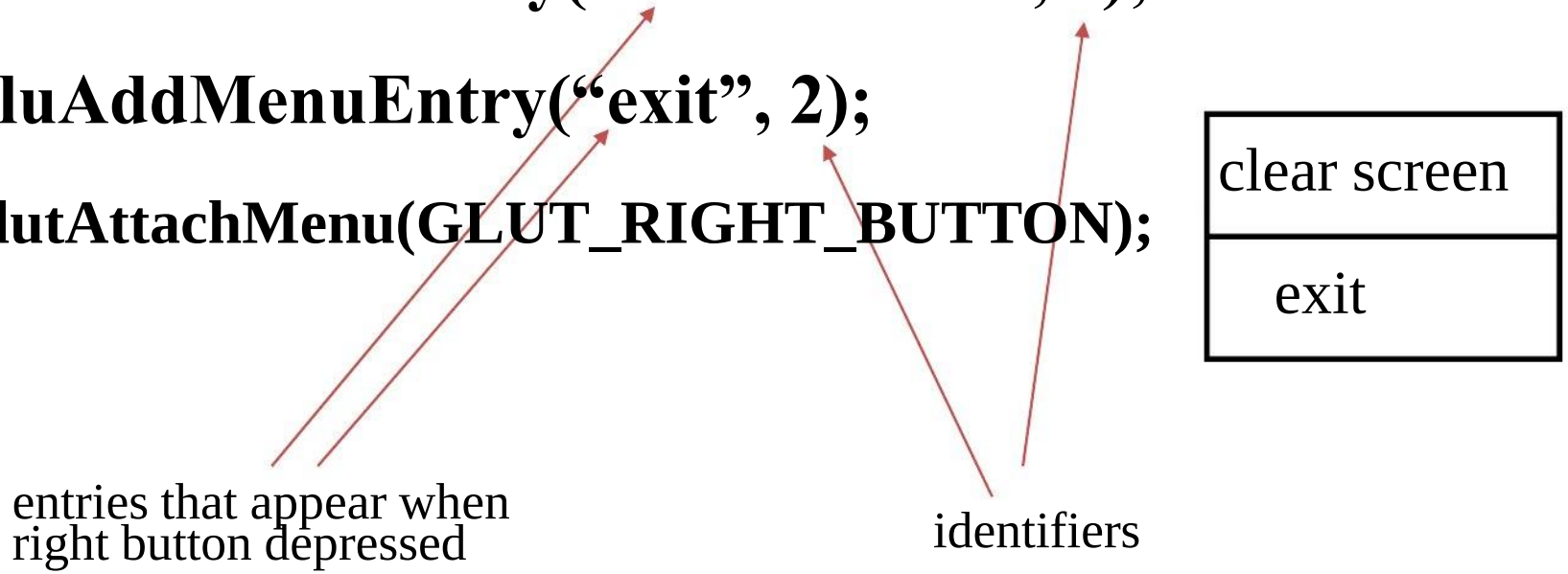
Menus

- GLUT supports pop-up menus
  - A menu can have submenus
- Three steps
  - Define entries for the menu
  - Define action for each menu item
    - Action carried out if entry selected
  - Attach menu to a mouse button

Defining a simple menu

- In **main.c**

```
menu_id = glutCreateMenu(mymenu);
glutAddmenuEntry("clear Screen", 1);

gluAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



Menu actions

– Menu callback

```
void mymenu(int id)
{
        if(id == 1) glClear();
        if(id == 2) exit(0);
}
```

– Note each menu has an id that is returned when it is created

– Add submenus by

glutAddSubMenu(char *submenu_name, submenu id)

entry in parent menu

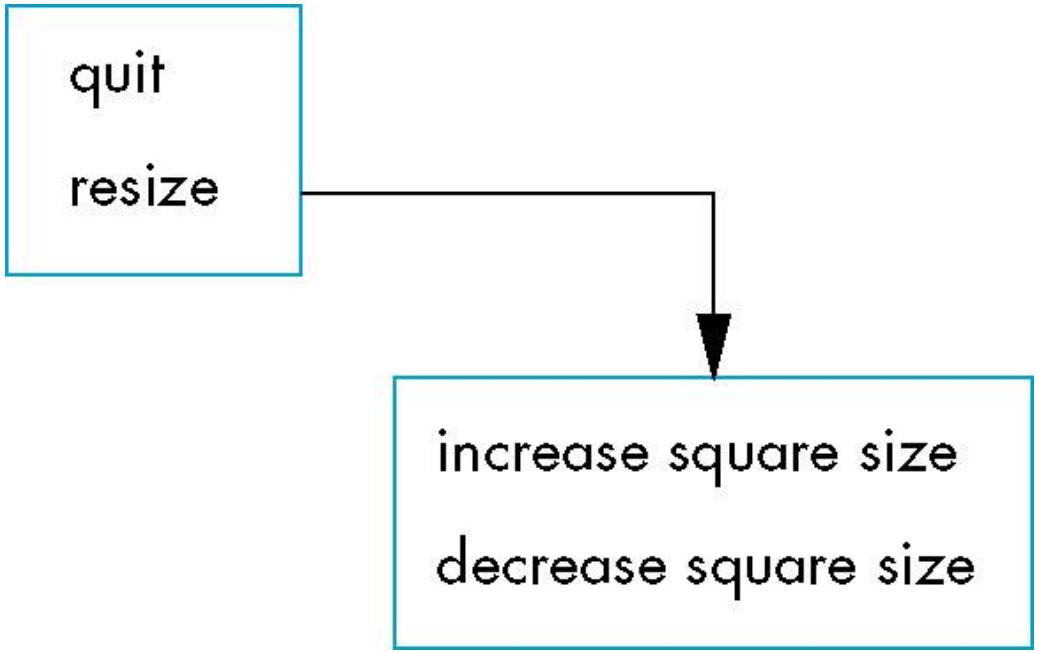
Menu actions

```
glutCreateMenu(demo_menu);
glutAddMenuEntry(“quit”,1);
glutAddMenuEntry(“Increase square size”, 2);
glutAddMenuEntry(“Decrease square size”, 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

The callback function demo_menu

```
void demo_menu(int id)
{
  switch(id)
    {
      case 1: exit(0); break;
      case 2: size=2*size; break;
      case 3: if (size>1) size=size/2; break;
     }
  glutPostReDisplay();
}
```

Hierarchical Menus




```
submenu=glutCreateMenu(size_menu);
glutAddMenuEntry("Increase square size", 2);
glutAddMenuEntry("Decrease square size", 3);

glutCreateMenu(top_menu);
glutAddMenuEntry("quit",1);
glutAddSubMenu("resize", submenu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Other functions in GLUT

- Dynamic Windows
  - Create and destroy during execution
- Subwindows
- Multiple Windows
- Changing callbacks during execution
- Timers
- Portable fonts
  - **glutBitmapCharacter**
  - **glutStrokeCharacter**

Better Interactive Programs

Objectives

- Learn to build more sophisticated interactive programs using
  - Picking
    - Select objects from the display
    - Three methods
  - Rubberbanding
    - Interactive drawing of lines and rectangles
  - Display Lists
    - Retained mode graphics

Using Regions of the Screen

- Many applications use a simple rectangular arrangement of the screen
  - Example: paint/CAD program

- Easier to look at mouse position and determine which area of screen it is in than using selection mode picking

A Simple CAD Program

- Create polygons
- Delete polygons
- Select and move polygons

Code available as polygon.c at:

http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/FIFTH_EDITION/PROGRAMS/CHAPTER03/

(Not perfect)

CAD Program: Definitions

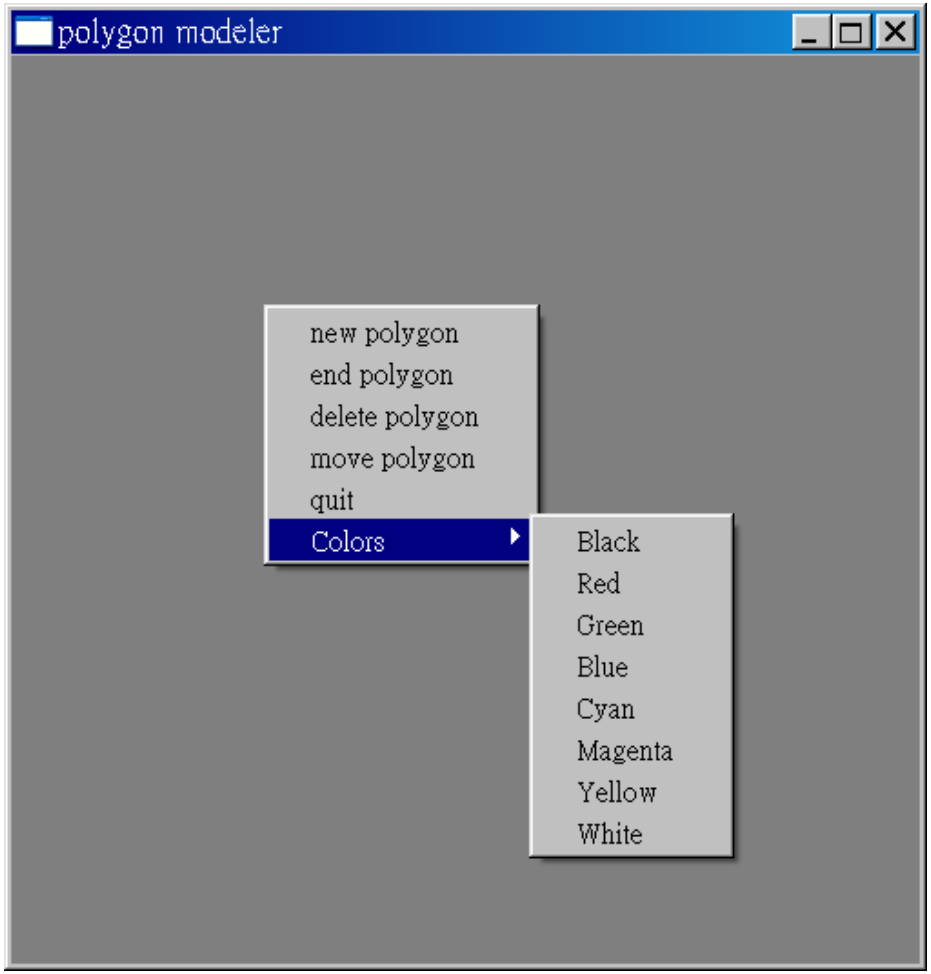
```
#define MAX_POLYGONS 8
#define MAX_VERTICES 10

typedef struct polygon
{
        int color; /* color index */
        bool used; /* TRUE if polygon exists */
        int xmin, xmax, ymin, ymax; /* bounding box */
        float xc, yc; /* center of polygon */
        int nvertices; /* number of vertices */
        int x[MAX_VERTICES]; /* vertices */
        int y[MAX_VERTICES];
} polygon;
```

CAD Program: Display

```
int i, j;

glClear(GL_COLOR_BUFFER_BIT);
for(i=0; i<MAX_POLYGONS; i++)
{
    if(polygons[i].used)
    {
        glColor3fv(colors[polygons[i].color]);
        glBegin(GL_POLYGON);
          for(j=0; j<polygons[i].nvertices; j++)
            glVertex2i(polygons[i].x[j], polygons[i].y[j]);
        glEnd();
    }
}
glFlush();
```

CAD Program: Menus

```
c_menu = glutCreateMenu(color_menu);
glutAddMenuEntry("Black",0);
glutAddMenuEntry("Red",1);
glutAddMenuEntry("Green",2);
glutAddMenuEntry("Blue",3);
glutAddMenuEntry("Cyan",4);
glutAddMenuEntry("Magenta",5);
glutAddMenuEntry("Yellow",6);
glutAddMenuEntry("White",7);

glutCreateMenu(main_menu);
glutAddMenuEntry("new polygon", 1);
glutAddMenuEntry("end polygon", 2);
glutAddMenuEntry("delete polygon", 3);
glutAddMenuEntry("move polygon", 4);
glutAddMenuEntry("quit",5);
glutAddSubMenu("Colors", c_menu);
glutAttachMenu(GLUT_MIDDLE_BUTTON);
```



CAD Program: Controlling State

- Global variables:

```
bool picking = FALSE; /* true while picking */
bool moving = FALSE; /* true while moving polygon */
int in_polygon = -1; /* not in any  polygon */
int present_color = 0; /* default color */
```

- Interactivity through main_menu callback

```
void main_menu(int index)
{
    int i;
    switch(index)
    {
        case(1):  /* create a new polygon */
          {
                /* code for creation of polygon */
                break;
          }
          /* rest of cases */
```

CAD Program: Create Polygon

```
case(1):   /* create a new polygon */
  {

          moving = FALSE;
          for(i=0; i<MAX_POLYGONS; i++)
                    if(polygons[i].used == FALSE) break;
          if(i == MAX_POLYGONS)
          {
                    printf("exceeeded maximum number of polygons\n");
                    exit(0);
          }
         polygons[i].color = present_color;
         polygons[i].used = TRUE;
         polygons[i].nvertices = 0;
         in_polygon = i;
         picking = FALSE;
         break;
 }
```

CAD Program: Ending a Polygon

```
case(2):    /* end polygon and find bounding box and center */
   {
        moving = FALSE;
        if(in_polygon>=0)
        {
            polygons[in_polygon].xmax =
                    polygons[in_polygon].xmin =
                    polygons[in_polygon].x[0];
            polygons[in_polygon].ymax =
                    polygons[in_polygon].ymin =
                            polygons[in_polygon].y[0];
            polygons[in_polygon].xc =  polygons[in_polygon].x[0];
            polygons[in_polygon].yc =  polygons[in_polygon].y[0];
```

CAD Program: Ending a Polygon

```
for(i=1;i<polygons[in_polygon].nvertices;i++)
{

      if(polygons[in_polygon].x[i]<polygons[in_polygon].xmin)
                polygons[in_polygon].xmin = polygons[in_polygon].x[i];
      else if(polygons[in_polygon].x[i]>polygons[in_polygon].xmax)
                polygons[in_polygon].xmax = polygons[in_polygon].x[i];

      if(polygons[in_polygon].y[i]<polygons[in_polygon].ymin)
                polygons[in_polygon].ymin = polygons[in_polygon].y[i];
      else if(polygons[in_polygon].y[i]>polygons[in_polygon].ymax)
                polygons[in_polygon].ymax = polygons[in_polygon].y[i];

       polygons[in_polygon].xc += polygons[in_polygon].x[i];
       polygons[in_polygon].yc += polygons[in_polygon].y[i];
}
```

CAD Program: Ending a Polygon

```
    polygons[in_polygon].xc =
          polygons[in_polygon].xc  / polygons[in_polygon].nvertices;
    polygons[in_polygon].yc =
          polygons[in_polygon].yc / polygons[in_polygon].nvertices;
     }
    in_polygon = -1;
    glutPostRedisplay();
    break;
}
```

CAD Program: Entering Vertices

```
void myMouse(int btn, int state, int x, int y)
{
    int i,j;
    y = wh-y;
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN &&!picking&&!moving){
    /* adding vertices */
        moving = FALSE;
        if(in_polygon>=0) {
            if(polygons[in_polygon].nvertices ==  MAX_VERTICES){
                printf("exceeds maximum number of vertices\n");
                exit(0);
            }
            i = polygons[in_polygon].nvertices;
            polygons[in_polygon].x[i] = x;
            polygons[in_polygon].y[i] = y;
            polygons[in_polygon].nvertices++;
        }
    }
```

CAD Program: Picking a Polygon

```
int pick_polygon(int x, int y){
/* find first polygon in which we are in bounding box */
int i;
for(i=0; i<MAX_POLYGONS; i++)
{
    if(polygons[i].used)
        if((x>=polygons[i].xmin)&&(x<=polygons[i].xmax)
            &&(y>=polygons[i].ymin)&&(y<polygons[i].ymax))
        {
            in_polygon = i;
            moving = TRUE;
            return(i);
        }
        printf("not in a polygon\n");
        return(-1);
    }
}
```

CAD Program: Deleting a Polygon

```
void myMouse(int btn, int state, int x, int y)
…
if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN
    && picking && !moving)
   {
      /* delete polygon */
            picking = FALSE;
            moving = FALSE;
            j = pick_polygon(x,y);
            if(j >= 0)
            {
                  polygons[j].used = FALSE;
                  in_polygon = -1;
                  glutPostRedisplay();
            }
      }
```

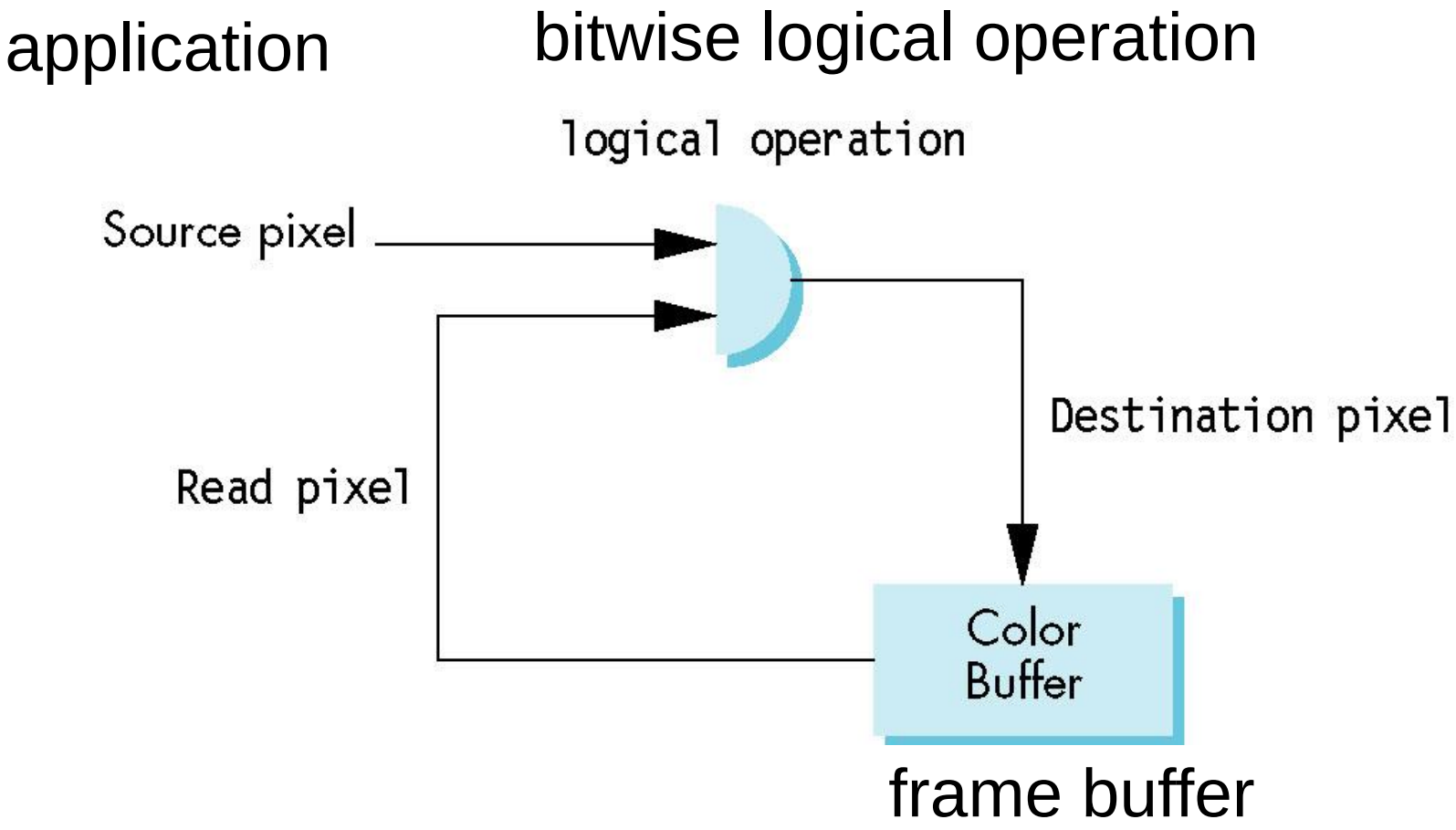
CAD Program: Moving a Polygon

- Enter moving mode by:

```
case(4):  /* set moving mode */
{
        moving = TRUE;
        break;
}
```

- Compute displacement by reading mouse coordinates relative to polygon center
- Displace all vertices
- Modify bounding box
- Call **glutPostRedisplay()**

Writing Modes



XOR write

- Usual (default) mode: source replaces destination (d' = s)
  - Cannot write temporary lines this way because we cannot recover what was "under" the line in a fast simple way
- Exclusive OR mode (XOR))((d' = d $\oplus$ s)
  - x $\oplus$ y $\oplus$ x =y
  - Hence, if we use XOR mode to write a line, we can draw it a second time and line is erased!

XOR write

$Y = 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1$ (Background)

$X = 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0$ (Drawing)

$Y \oplus X = 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1$ (Display)

$X = 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0$ (Redrawing)

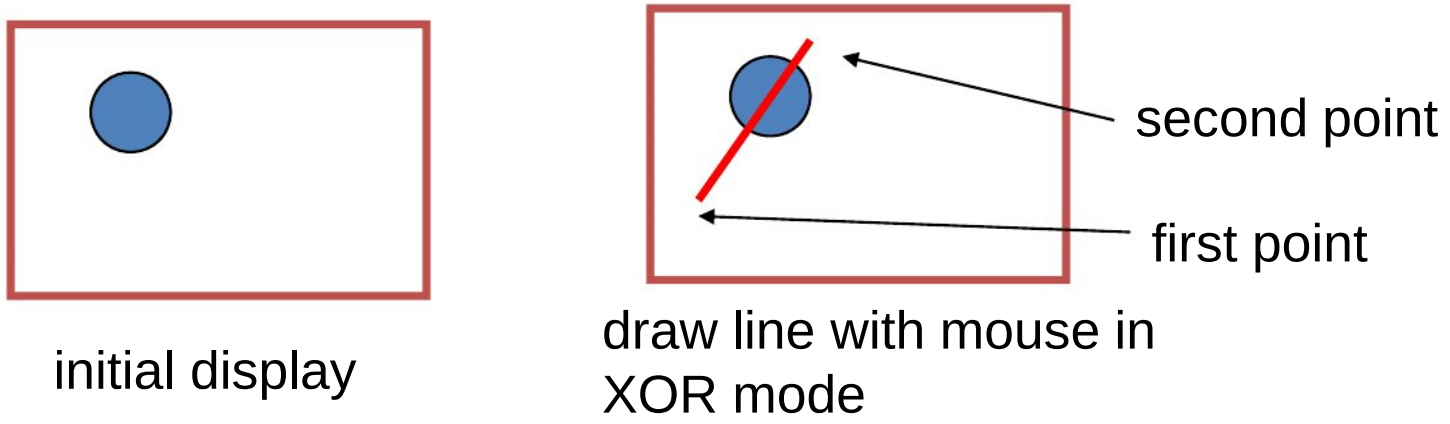
$(Y \oplus X) \oplus X = 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1$ (Recovery)

Background

Rubberbanding

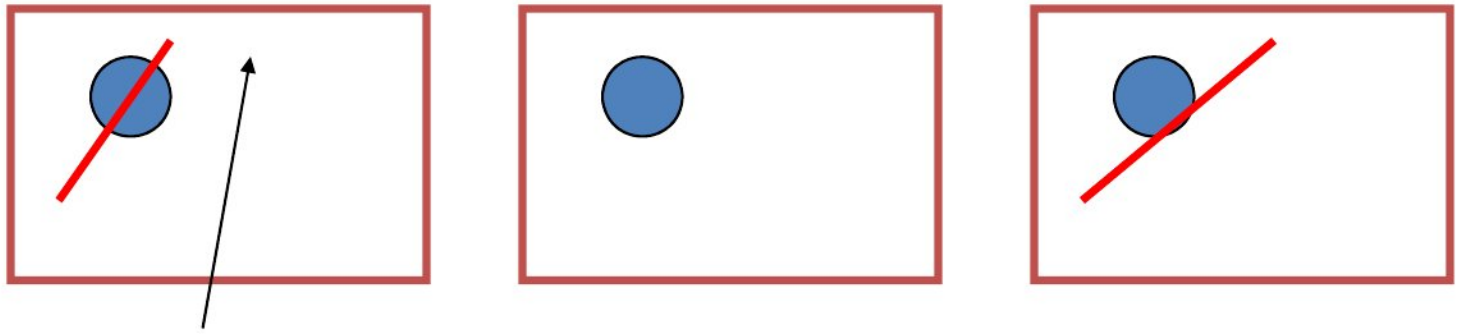
- Switch to XOR write mode
- Draw object
  - For line can use first mouse click to fix one endpoint and then use motion callback to continuously update the second endpoint
  - Each time mouse is moved, redraw line which erases it and then draw line from fixed first position to to new second position
  - At end, switch back to normal drawing mode and draw line
  - Works for other objects: rectangles, circles

Rubberband Lines



initial display

draw line with mouse in XOR mode

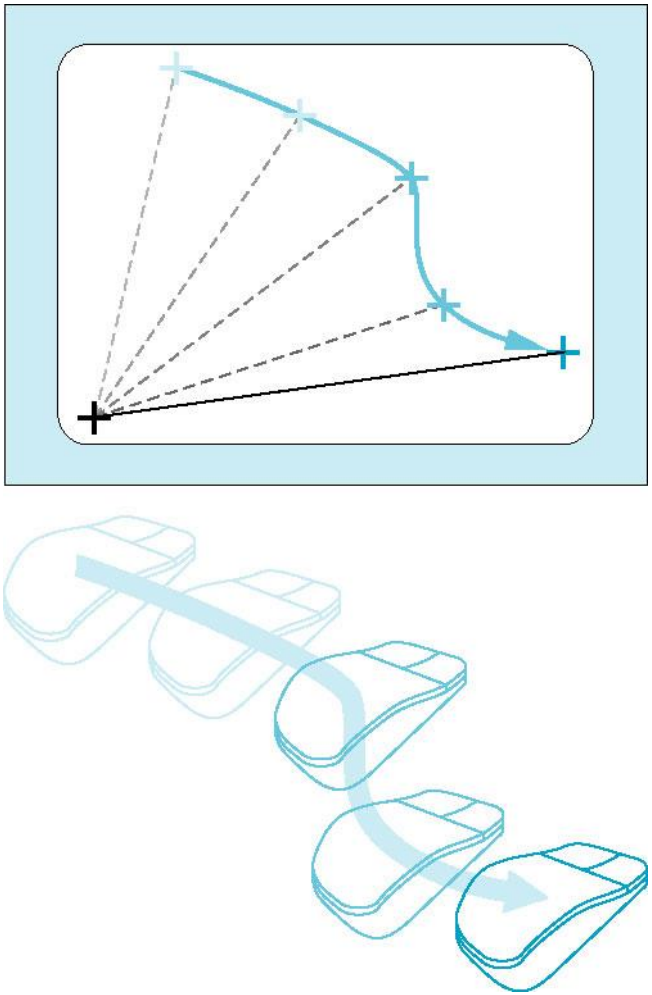


mouse moved to new position

original line redrawn with XOR

new line drawn with XOR

Rubberband Lines



XOR in OpenGL

- There are 16 possible logical operations between two bits
- All are supported by OpenGL
  - Must first enable logical operations
    - **glEnable(GL_COLOR_LOGIC_OP)**
  - Choose logical operation
    - **glLogicOp(GL_XOR)**
    - **glLogicOp(GL_COPY)** (default)

Drawing Erasable Lines

OpenGL window: 500x500 pixels

Clipping window: a unit square with origin at the lower-left corner

The first endpoint screen coordinates: (x,y)

The first point's world coordinates:

```
xm=x/500.;
ym=(500-y)/500.;
```

We then get the second point and draw a line segment in XOR mode:

```
xmm=x/500.;
ymm=(500-y)/500.;
glLogicOP(GL_XOR);
glBegin(GL_LINES);
   glvertex2f(xm,ym);
   glVertex2f(xmm,ymm);
glLogicOP(GL_COPY);
glEnd();
glFlush();
```

Drawing Erasable Lines

If we enter another point with the mouse, we first draw the same line in XOR mode and then draw a second line using the first endpoint and the mouse input as follows:

```
glLogicOP(GL_XOR);
glBegin(GL_LINES);
   glvertex2f(xm,ym);
   glVertex2f(xmm,ymm);
glEnd();
glFlush();
xmm=x/500.;
ymm=(500-y)/500.;
glBegin(GL_LINES);
   glvertex2f(xm,ym);
   glVertex2f(xmm,ymm);
glEnd();
glLogicOP(GL_COPY); /* in normal mode */
glFlush();
```

Erase the first line

Draw a new line

Drawing Erasable Rectangles

```
/* globals */
Float xm, ym, xmm, ymm;  /* the corners of the rectangle */
Int first = 0; /* vertex the counter */
```

The callbacks are registered as follows:

```
glutMouseFunc(mouse);
glutMotionFunc(move);
```

The code for the callbacks

```
void move(int x, int y)
{
  if (first = 1)
   {
     glRectf(xm, ym, xmm, ymm);
     glFlush();
   }
  xmm= x/500.;
  ymm=(500-y)/500.;
  glRectf(xm, ym, xmm, ymm);
  glFlush();
  first = 1;
}
```

Erase a rectangle

Draw a new rectangle

Drawing Erasable Rectangles

```
void mouse(int btn, int state, int x, int y)
{  if (btn==GL_LEFT_BUTTON && state==GL_DOWN)
    {  xm=x/500.;  ym=(500-y)/500.;
       glColor3f(0.0, 0.0, 1.0);
       glLogicOP(GL_XOR);
       first=0;
     }
    if (btn==GL_LEFT_BUTTON && state==GL_UP)
     { glRectf(xm, ym, xmm, ymm);
        glFlush();
        glColor3f(0.0, 1.0, 0.0);
        glLoginOP(GL_COPY);
        xmm=x/500.; ymm=(500-y)/500.;
        glLoginOP(GL_COPY);
        glRectf(xm, ym, xmm, ymm);
       glFlush();
      }
 }
```

Erase a rectangle

Draw a new rectangle

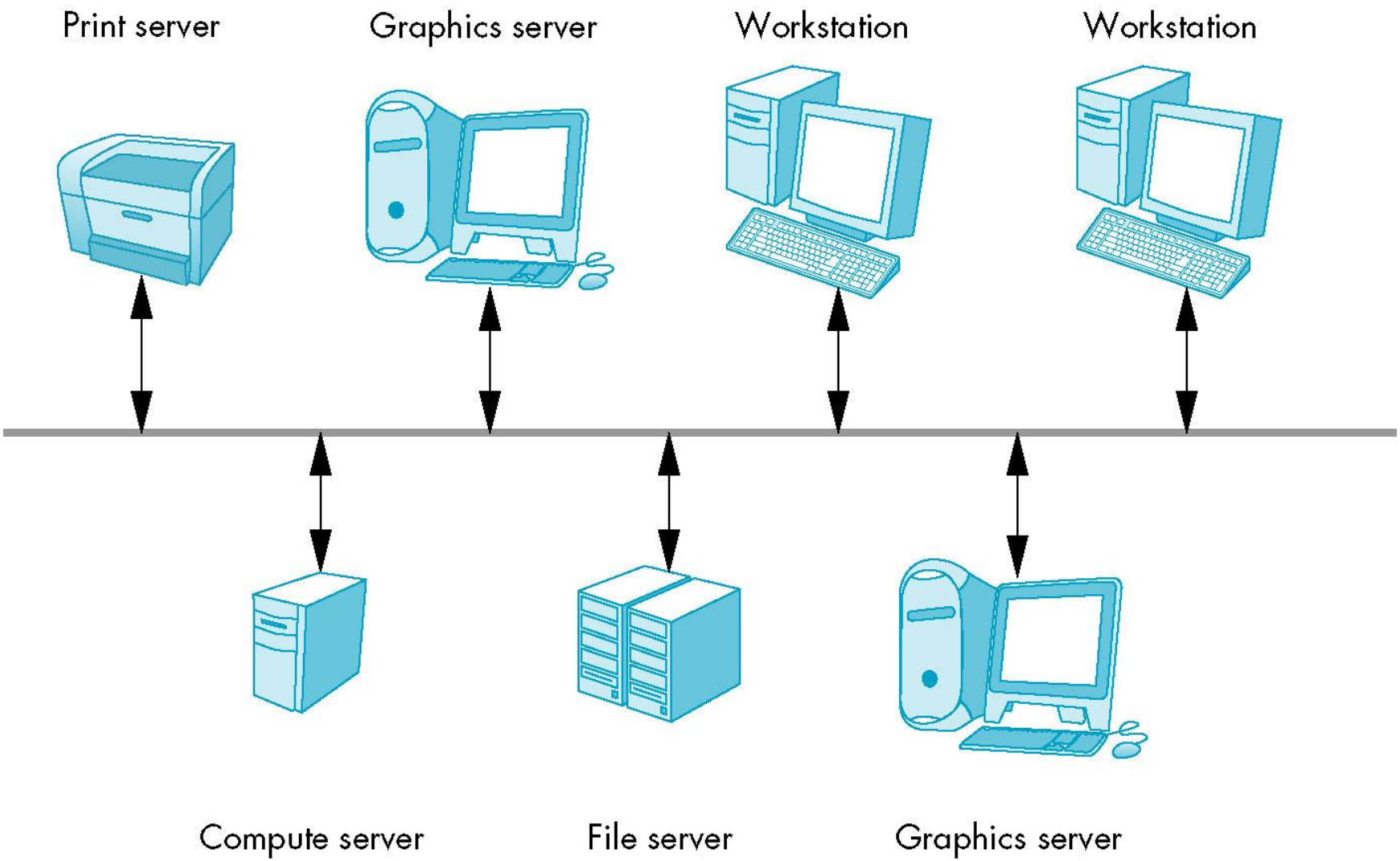
Immediate and Retained Modes

- Recall that in a standard OpenGL program, once an object is rendered there is no memory of it and to redisplay it, we must re-execute the code for it
  - Known as immediate mode graphics
  - Can be especially slow if the objects are complex and must be sent over a network
- Alternative is define objects and keep them in some form that can be redisplayed easily
  - Retained mode graphics
  - Accomplished in OpenGL via display lists

Display Lists

- Conceptually similar to a graphics file
  - Must define (name, create)
  - Add contents
  - Close
- In client-server environment, display list is placed on server
  - Can be redisplayed without sending primitives over network each time

Network


Print server
Graphics server
Workstation
Workstation
Compute server
File server
Graphics server

Display List Functions

- Creating a display list

```
GLuint id;

void init()
{
      id = glGenLists( 1 );
      glNewList( id, GL_COMPILE );
      /* other OpenGL routines */
      glEndList();
 }
```

returns id of consecutive free lists, equal to the argument (1, here)

- Call a created list

```
void display()
{
      glCallList( id );
}
```

Display List Example

```
#define BOX 1
glNewList( BOX, GL_COMPILE );
    glBegin( GL_POLYGON );
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-1.0, -1.0);
        glVertex2f( 1.0, -1.0);
        glVertex2f( 1.0,  1.0);
        glVertex2f(-1.0, -1.0);
    glEnd();
glEndList();
…

glCallList( BOX );
```

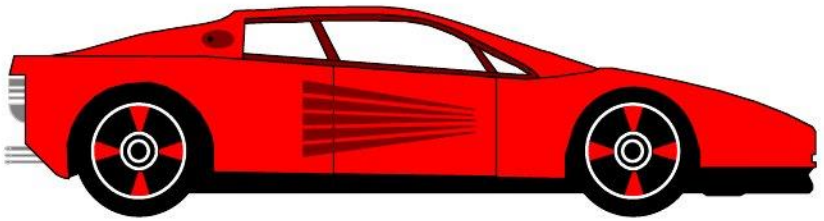
```
glMatrixMode(GL_Projection);
for (i=1; i<5; i++) {
    glLoadIdentity();
    gluortho2D(-2.0*i, 2.0*i, -2.0*i, 2.0*i);
    glCallList(BOX);
}
```

The box will appear different places or will no longer appear

Hierarchy and Display Lists

- Consider model of a car
  - Create display list for chassis
  - Create display list for wheel

```
glNewList( CAR, GL_COMPILE );
  glCallList( CHASSIS );
  glTranslatef( … );
  glCallList( WHEEL );
  glTranslatef( … );
  glCallList( WHEEL );
          …
glEndList();
```



Display of Display Lists

- **glNewList(CAR, GL_COMPILE);**
- Compiles the instructions, but does not display list contents
- **glNewList(CAR, GL_COMPILE_AND EXECUTE);**
- Compiles and displays

Calling Display Lists

- Current state determines transformations
- User can change model view or projection matrices between executions of display list
  - E.g. redraw box with increasingly larger clipping rectangle

```
glMatrixMode(GL_Projection);
for (i=1; i<5; i++) {
    glLoadIdentity();
    gluortho2D(-2.0*i, 2.0*i, -2.0*i, 2.0*i);
    glCallList(BOX);
}
```

Display Lists and State

- Most OpenGL functions can be put in display lists
- State changes made inside a display list persist after the display list is executed
- Can avoid unexpected results by using **glPushAttrib** and **glPushMatrix** upon entering a display list and **glPopAttrib** and **glPopMatrix** before exiting

Preserving the State

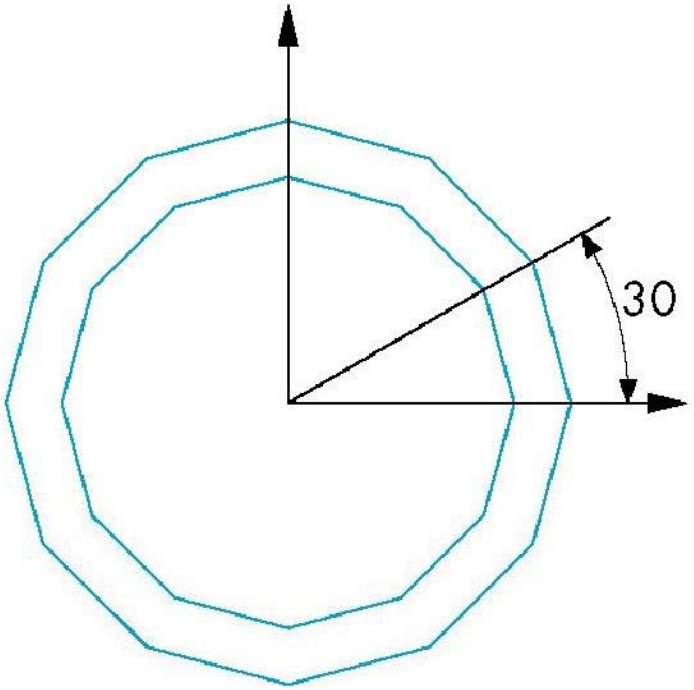
```
glPushAttrib(GL_ALL_ATTRIB_BITS);
glPushMatrix();
…
glPopAttrib();
glPopMatrix();
```

Text and Display Lists

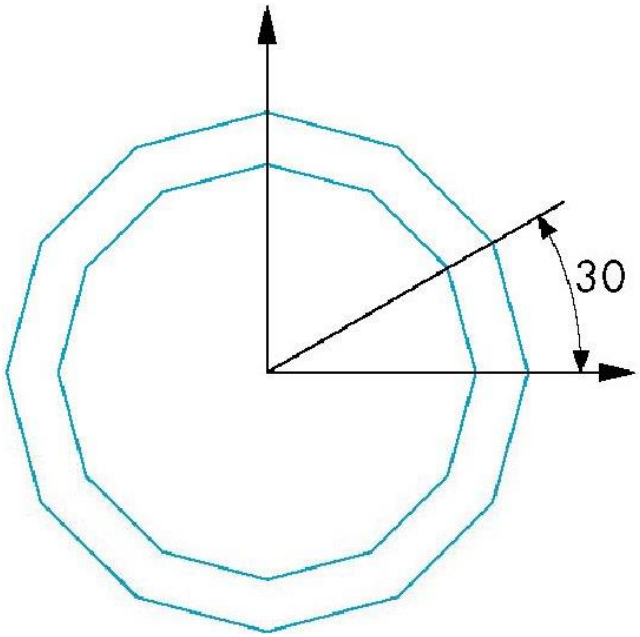
- Both stroke and raster fonts require several bytes per character
  - Sending characters to the display one by one is impractical
- Instead, define fonts once using a display list for each character
- Translate vertices appropriately and call display list

Text and Display Lists

- E.g. define O as quad strip
  - Stroke or raster font?



Drawing the letter of “0”




```
void Ourfont(char c)
{
    switch (c)
     {
        case ‘a’:
              …
              break;
        case ‘b’:
              …
              break;
        case ‘0’:
              glTranslatef(0.5,0.5,0.0); /* move to center */
              glBegin(GL_QUAD_STRIP);
              for ( i=0; i< 12; i++)
                { angel=3.14159/6.0*i; /* 30 degrees on radians */
                  glVertex2f(0.4*cos(angel)+0.5, 0.4*sin(angel)+0.5);
                  glVertex2f(0.5*cos(angel)+0.5, 0.5*sin(angel)+0.5);
                 }
               glEnd();
               break;
        …
     }
}
```

Generating a 256-character set

```
base=glGenLists(256);  /* return index of first of
                          256 consecutive available ids */
for ( i=0; i<256; i++)
 {
   glNewList(base+i, GL_COMPILE);
   Ourfont(i);
   glEndeList();
 }
```

For drawing individual characters, we do not have to offset the identifier of the display lists by base each time. We can set an offset as follows:

```
glListBase(base);
```

Finally, our drawing of a string is accomplished in the sever by the function call

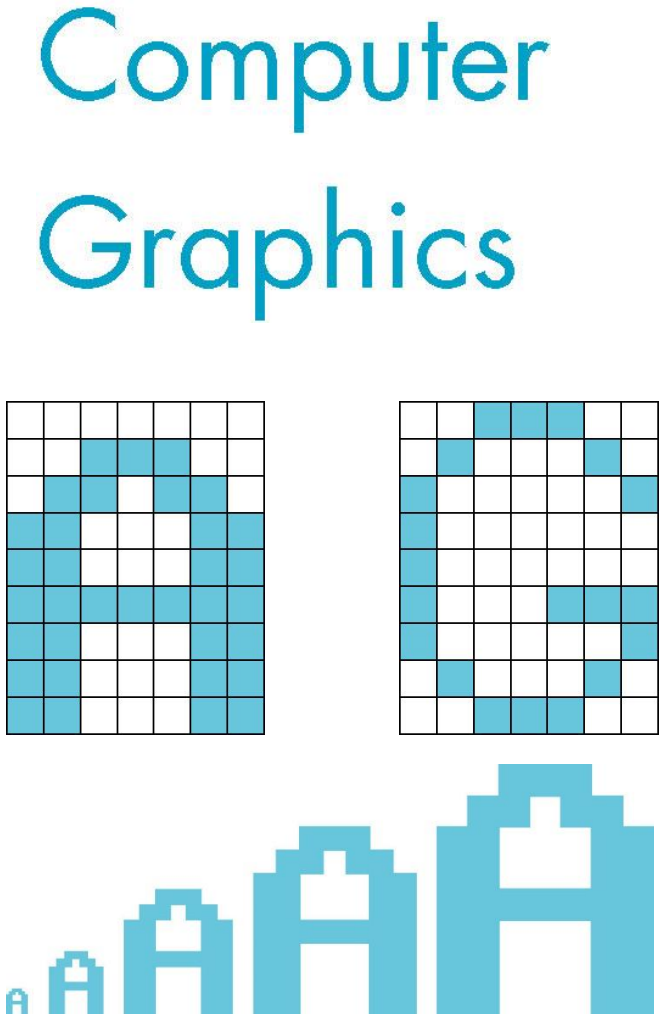
```
char *text_string;
glCallLists((GLint) strlen(text_string),GL_BYTE,text_string);
```

Fonts in GLUT

- Stroke characters
  - E.g. access a single character from a monotype, or evenly spaces font by the following function call

    glutStrokeCharacter(GLUT_STROKE_MONO-ROMAN, int character)
- Raster and Bitmap characters
  - E.g. access a single 8x13 character by the following function call

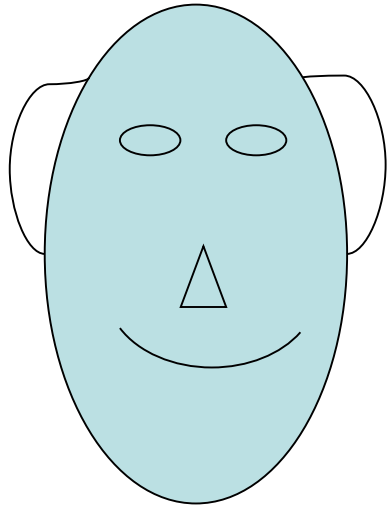
    glutBitmapCharacter(GLUT_BITMAP_8_BY_13, int character)



Display Lists and Modeling: Simple Animated Face

```
#define EYE 1  /* or some other integer */

glNewList(EYE);
   /* eye code */
glEndList();
```

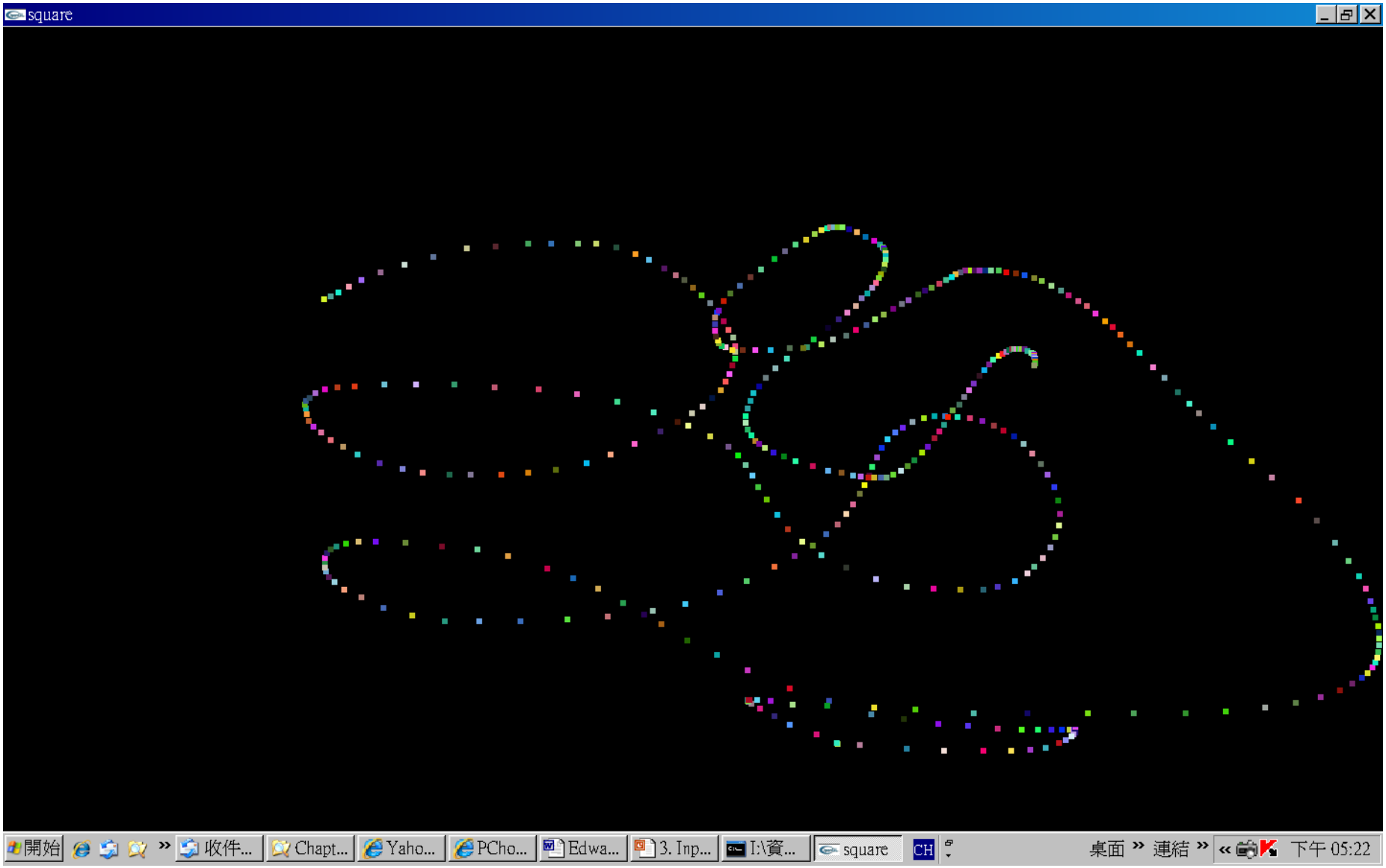
```
#define FACE 2 /* or some other integer */
glNewList(FACE);
  /* draw the outline */
  glTranslatef(…);  /* right eye position */
  glCalllist(EYE);
  glTranslatef(…);  /* left eye position */
  glCallList(EYE);
  glTranslatef(…)   /* nose position */
  glCallList(NOSE);

  /* similar code for ears and mouth */
glEndList();
```



Sample Programs

- Square Drawing Program
  - square.c
- A.5 Polygon Modeling Program
  - polygon.c
- A.6 Double-Buffering Program
  - simple-double.c

square.c (1/6)


square
開始
收件...
Chapt...
Yaho...
PCho...
Edwa...
3. Inp...
I:\資...
square
CH
桌面
連結
下午 05:22

square.c (2/6)

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <stdlib.h>

/* globals */

GLsizei wh = 500, ww = 500; /* initial window size */
GLfloat size = 3.0;   /* half side length of square */
```

square.c (3/6)

```
void drawSquare(int x, int y)
{

        y=wh-y;
        glColor3ub( (char) rand()%256, (char) rand()%256, (char) rand()%256);
        glBegin(GL_POLYGON);
                glVertex2f(x+size, y+size);
                glVertex2f(x-size, y+size);
                glVertex2f(x-size, y-size);
                glVertex2f(x+size, y-size);
        glEnd();
        glFlush();
}
```

square.c (4/6)

```
/* rehaping routine called whenever window is resized
or moved */

void myReshape(GLsizei w, GLsizei h)
{
/* adjust clipping box */

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, -1.0, 1.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

/* adjust viewport and clear */

        glViewport(0,0,w,h);
        glClearColor (0.0, 0.0, 0.0, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();

/* set global size for use by drawing routine */
        ww = w;
        wh = h;
}
```

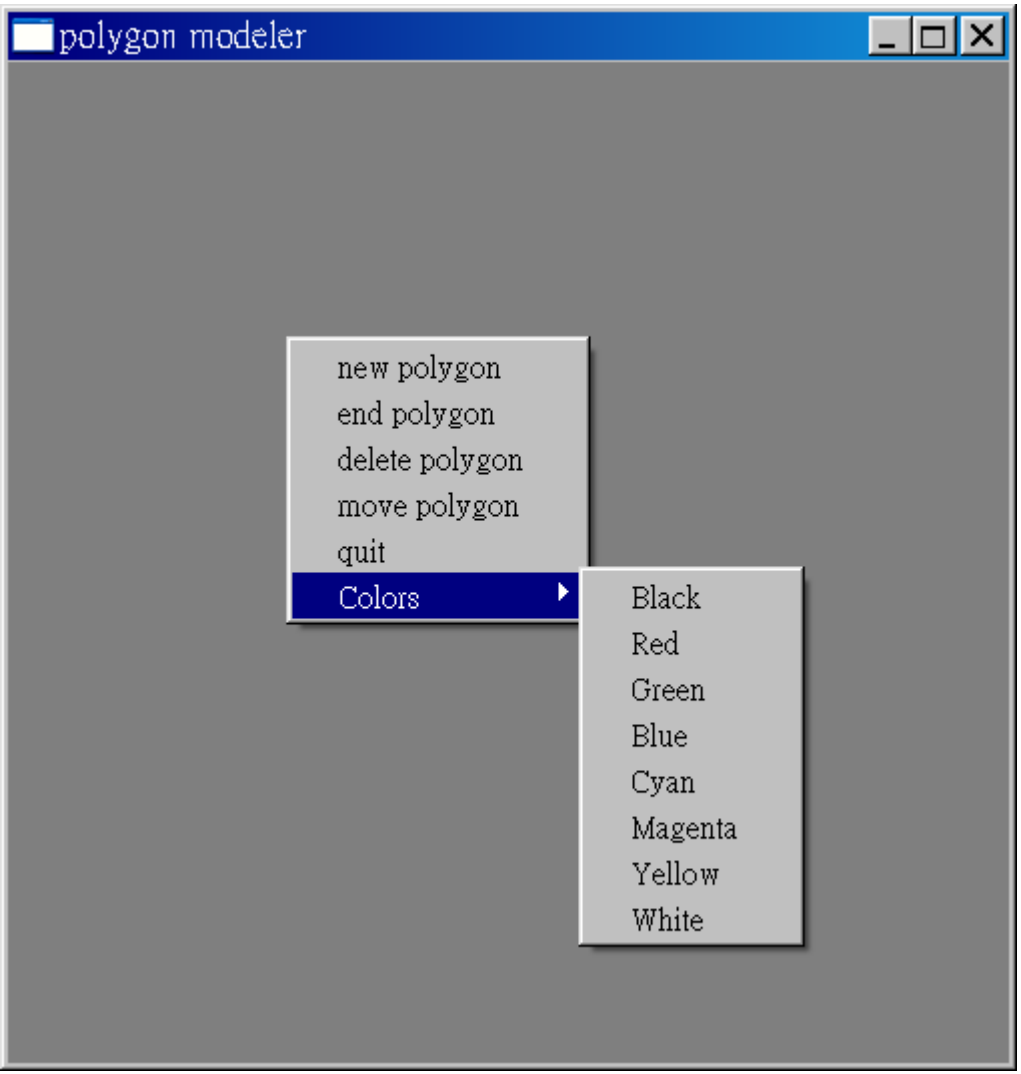
square.c (5/6)

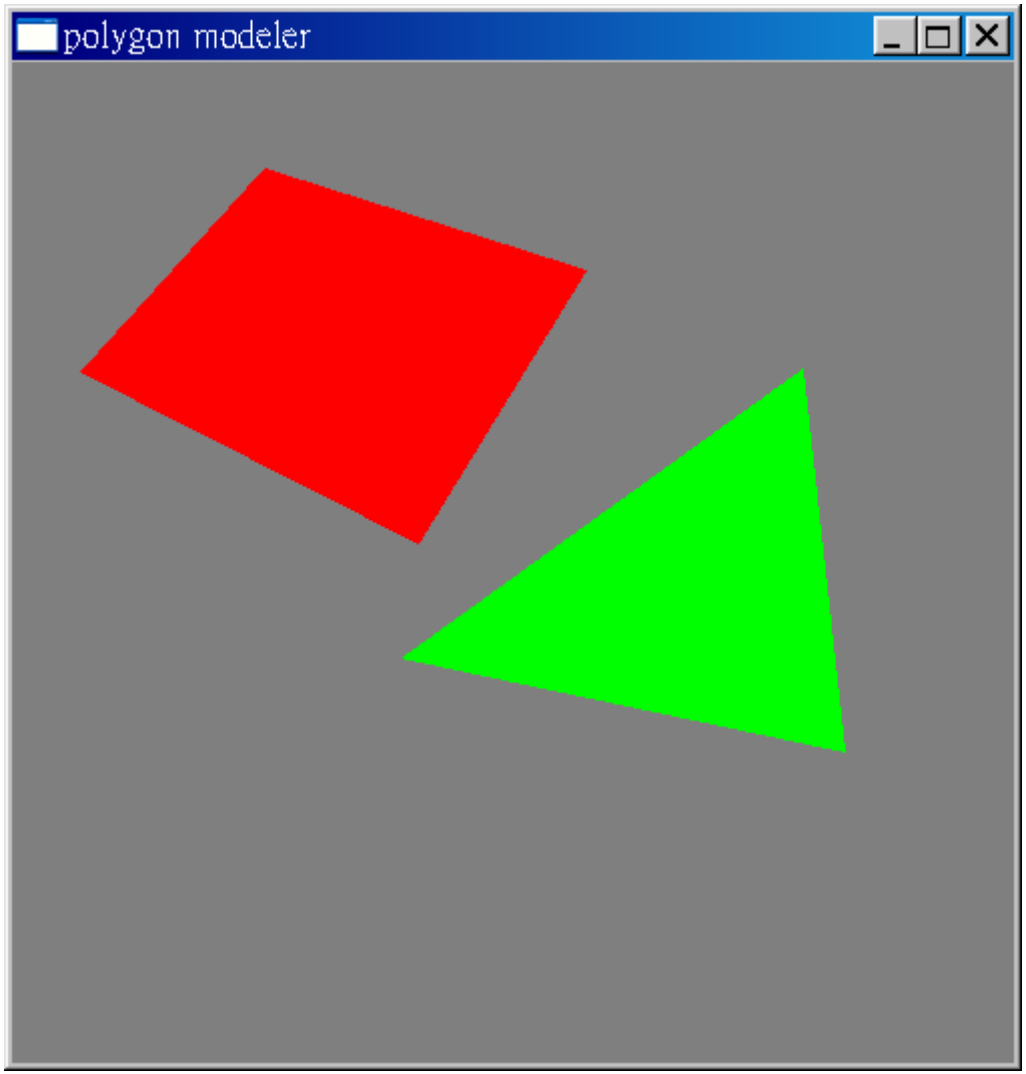
```
void myinit(void)
{

        glViewport(0,0,ww,wh);

/* Pick 2D clipping window to match size of screen window
This choice avoids having to scale object coordinates
each time window is resized */

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, (GLdouble) ww , 0.0, (GLdouble) wh , -1.0, 1.0);

/* set clear color to black and clear window */

         glClearColor (0.0, 0.0, 0.0, 1.0);
         glClear(GL_COLOR_BUFFER_BIT);
         glFlush();

/* callback routine for reshape event */

         glutReshapeFunc(myReshape);

}
```

square.c (6/6)

```
void mouse(int btn, int state, int x, int y)
{
   if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)   exit(0);
}

/* display callback required by GLUT 3.0 */

void display(void)
{}

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

        glutInit(&argc,argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutCreateWindow("square");
        myinit ();
        glutReshapeFunc (myReshape);
        glutMouseFunc (mouse);
        glutMotionFunc(drawSquare);
        glutDisplayFunc(display);

        glutMainLoop();

}
```

A.5 polygon.c (1/17)





A.5 polygon.c (2/17)

```
/* polygon modeler */

#define MAX_POLYGONS 8
#define MAX_VERTICES 10

typedef int bool;
#define TRUE 1
#define FALSE 0
#include <stdlib.h>
#include <stdio.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
void myMouse(int,int, int, int);
void myMotion(int, int);
void myDisplay();
void myReshape(int, int);
void color_menu(int);
void main_menu(int);
int pick_polygon(int x, int y);
void myinit();
```

A.5 polygon.c (3/17)

```
/* globals */

/* polygon struct */
typedef struct polygon
{
    int color; /* color index */
    bool used; /* TRUE if polygon exists */
    int xmin, xmax, ymin, ymax; /* bounding box */
    float xc, yc; /* center of polygon */
    int nvertices; /* number of vertices */
    int x[MAX_VERTICES]; /* vertices */
    int y[MAX_VERTICES];
} polygon;

/* flags */
bool picking = FALSE; /* true while picking */
bool moving = FALSE; /* true while moving polygon */
int in_polygon = -1; /* not in any  polygon */
int present_color = 0; /* default color */
GLsizei wh = 500, ww = 500; /* initial window size */

int draw_mode = 0; /* drawing mode */
```

A.5 polygon.c (4/17)

```
/* globals */

int draw_mode = 0; /* drawing mode */

GLfloat colors[8][3]={{0.0, 0.0, 0.0}, {1.0, 0.0, 0.0},{0.0, 1.0, 0.0},
    {0.0, 0.0, 1.0}, {0.0, 1.0, 1.0}, {1.0, 0.0, 1.0}, {1.0, 1.0, 0.0},
    {1.0, 1.0, 1.0}};

polygon polygons[MAX_POLYGONS];

void myReshape(int w, int h)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glViewport(0,0,w,h);
        ww = w;
        wh = h;
}
```

A.5 polygon.c (5/17)

```
void myinit()
{
        int  i;

/* set clear color to grey */

        glClearColor(0.5, 0.5, 0.5, 1.0);

/* mark all polygons unused */

 for(i = 0; i<MAX_POLYGONS; i++) polygons[i].used = FALSE;

}
```

A.5 polygon.c (6/17)

```
void myMouse(int btn, int state, int x, int y)
{
    int i,j;
    y = wh-y;
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN &&!picking&&!moving)

    /* adding vertices */

    {
        moving = FALSE;
        if(in_polygon>=0)
        {
            if(polygons[in_polygon].nvertices == MAX_VERTICES)
            {
                printf("exceeds maximum number vertices\n");
                exit(0);
            }
            i = polygons[in_polygon].nvertices;
            polygons[in_polygon].x[i] = x;
            polygons[in_polygon].y[i] = y;
            polygons[in_polygon].nvertices++;
        }
    }
```

A.5 polygon.c (7/17)

```
if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN && picking&&!moving)
    {

    /* delete polygon */

        picking = FALSE;
        moving = FALSE;
        j = pick_polygon(x,y);
        if(j >= 0)
        {
                polygons[j].used = FALSE;
                in_polygon = -1;
                glutPostRedisplay();
        }
    }
}
```

A.5 polygon.c (8/17)

```
int pick_polygon(int x, int y)
{

    /* find first polygon in which we are in bounding box */

    int i;
    for(i=0; i<MAX_POLYGONS; i++)
    {
        if(polygons[i].used)

if((x>=polygons[i].xmin)&&(x<=polygons[i].xmax)&&(y>=polygons[i].ymin)&&
(y<polygons[i].ymax))
            {
                in_polygon = i;
                moving = TRUE;
                return(i);
            }
    }
    printf("not in a polygon\n");
    return(-1);

}
```

A.5 polygon.c (9/17)

```
void myMotion(int x, int y)
{
   /* find if we are inside a polugon */

   float dx, dy;
   int i,j;
   if(moving)
   {
      y = wh - y;
      j = pick_polygon(x, y);
      if(j<0)
        {
            printf("not in a polygon\n");
            return;
         }
      /* if inside then move polygon */
      dx = x - polygons[j].xc;
      dy = y - polygons[j].yc;
      for(i = 0; i< polygons[j].nvertices; i++)
      {
         polygons[j].x[i] += dx;
         polygons[j].y[i] += dy;
      }
```

A.5 polygon.c (10/17)

```
/* update bounding box */

        polygons[j].xc += dx;
        polygons[j].yc += dy;
        /*
          if(dx>0) polygons[j].xmax += dx;
            else polygons[j].xmin += dx;
          if(dy>0) polygons[j].ymax += dy;
            else polygons[j].ymin += dy;
          */
         polygons[j].xmax += dx;
         polygons[j].xmin += dx;
         polygons[j].ymax += dy;
         polygons[j].ymin += dy;
        glutPostRedisplay();
    }
}
```

A.5 polygon.c (11/17)

```
void color_menu(int index)
{
    present_color =  index;
    if (in_polygon>=0) polygons[in_polygon].color = index;
}
```

A.5 polygon.c (12/17)

```
void main_menu(int index)
{
  int i;
  switch(index)
  {
    case(1):  /* create a new polygon */
          {

            moving = FALSE;
                  for(i=0; i<MAX_POLYGONS; i++) if(polygons[i].used == FALSE) break;
                  if(i == MAX_POLYGONS)
                  {
                  printf("exceeeded maximum number of polygons\n");
                  exit(0);
                  }
            polygons[i].color = present_color;
            polygons[i].used = TRUE;
                  polygons[i].nvertices = 0;
                  in_polygon = i;
            picking = FALSE;
            break;
          }
```

```
case(2):   /* end polygon and find bounding box and center */
 { moving = FALSE;
   if (in_polygon>=0)
   { polygons[in_polygon].xmax = polygons[in_polygon].xmin = polygons[in_polygon].x[0];
     polygons[in_polygon].ymax = polygons[in_polygon].ymin = polygons[in_polygon].y[0];
     polygons[in_polygon].xc = polygons[in_polygon].x[0];
     polygons[in_polygon].yc = polygons[in_polygon].y[0];
     for(i=1;i<polygons[in_polygon].nvertices;i++)
       {  if (polygons[in_polygon].x[i]<polygons[in_polygon].xmin)
                  polygons[in_polygon].xmin = polygons[in_polygon].x[i];
            else if (polygons[in_polygon].x[i]>polygons[in_polygon].xmax)
                       polygons[in_polygon].xmax = polygons[in_polygon].x[i];
          if (polygons[in_polygon].y[i]<polygons[in_polygon].ymin)
                      polygons[in_polygon].ymin = polygons[in_polygon].y[i];
            else if (polygons[in_polygon].y[i]>polygons[in_polygon].ymax)
                       polygons[in_polygon].ymax = polygons[in_polygon].y[i];
         polygons[in_polygon].xc += polygons[in_polygon].x[i];
         polygons[in_polygon].yc += polygons[in_polygon].y[i];
        }
     polygons[in_polygon].xc = polygons[in_polygon].xc/polygons[in_polygon].nvertices;
     polygons[in_polygon].yc = polygons[in_polygon].yc/polygons[in_polygon].nvertices;
   }
   in_polygon = -1;
   glutPostRedisplay();
   break; }
```

A.5 polygon.c (14/17)

```
case(3):  /* set picking mode */
            {
              picking = TRUE;
              moving = FALSE;
                    break;
            }
        case(4):  /* set moving mode */
        {
              moving = TRUE;
              break;
        }
            case(5):  /* exit */
            {
                    exit(0);
                    break;
            }
    }
}
```

A.5 polygon.c (15/17)

```
void myDisplay()
{

    /* display all active polygons */

    int i, j;

    glClear(GL_COLOR_BUFFER_BIT);
    for(i=0; i<MAX_POLYGONS; i++)
    {
       if(polygons[i].used)
       {
          glColor3fv(colors[polygons[i].color]);
          glBegin(GL_POLYGON);
          for(j=0; j<polygons[i].nvertices; j++) glVertex2i(polygons[i].x[j], polygons[i].y[j]);
          glEnd();
       }
    }
    glFlush();
}
```

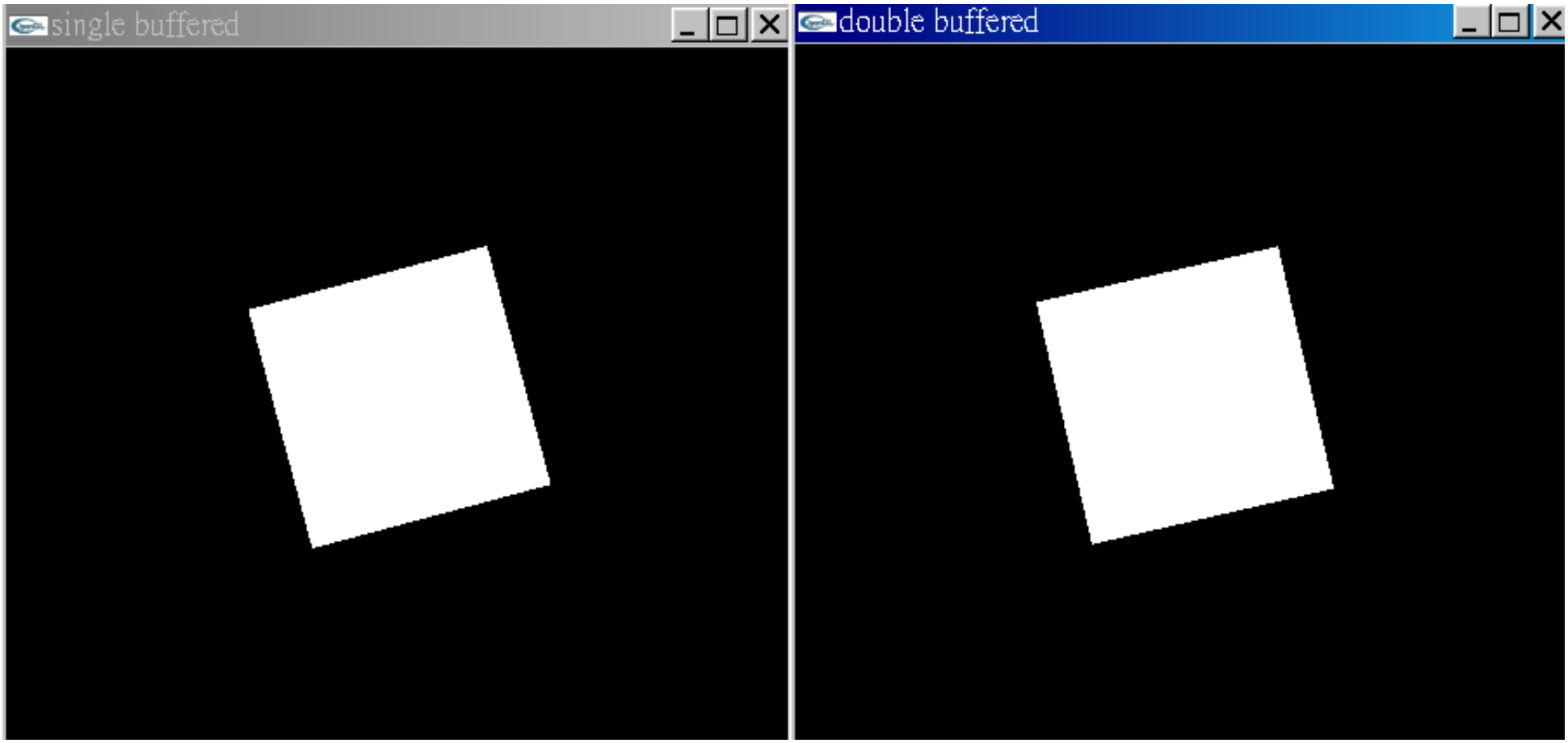
A.5 polygon.c (16/17)

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

    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("polygon modeler");
    glutDisplayFunc(myDisplay);
    myinit ();
    c_menu = glutCreateMenu(color_menu);
    glutAddMenuEntry("Black",0);
    glutAddMenuEntry("Red",1);
    glutAddMenuEntry("Green",2);
    glutAddMenuEntry("Blue",3);
    glutAddMenuEntry("Cyan",4);
    glutAddMenuEntry("Magenta",5);
    glutAddMenuEntry("Yellow",6);
    glutAddMenuEntry("White",7);
```

A.5 polygon.c (17/17)

```
    glutCreateMenu(main_menu);
    glutAddMenuEntry("new polygon", 1);
    glutAddMenuEntry("end polygon", 2);
    glutAddMenuEntry("delete polygon", 3);
    glutAddMenuEntry("move polygon", 4);
    glutAddMenuEntry("quit",5);
    glutAddSubMenu("Colors", c_menu);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);

    glutReshapeFunc (myReshape);
    glutMouseFunc (myMouse);
    glutMotionFunc(myMotion);
    glutMainLoop();
    return 0;

}
```

A.6 simple-double.c (1/7)

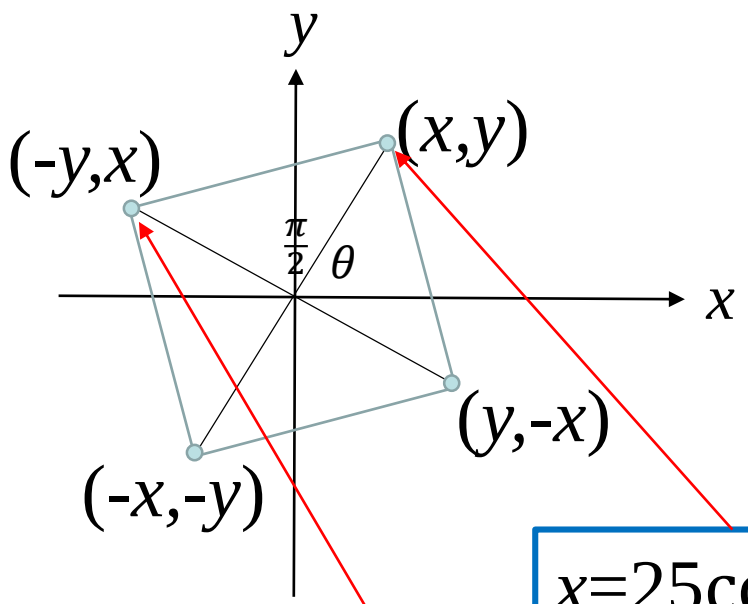


A.6 simple-double.c (2/7)

```
/*
 *  double.c
 *  This program demonstrates double buffering for
 *  flicker-free animation.  The left and middle mouse
 *  buttons start and stop the spinning motion of the square.
 */

#include <stdlib.h>

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <math.h>

#define DEGREES_TO_RADIANS 3.14159/180.0

static GLfloat spin = 0.0;
GLfloat x, y;
int singleb, doubleb;
```

A.6 simple-double.c (3/7)

```
void square()
{
    glBegin(GL_QUADS);
            glVertex2f(x,y);
            glVertex2f(-y,x);
            glVertex2f(-x,-y);
            glVertex2f(y,-x);
    glEnd();
}

void displayd()
{
    glClear (GL_COLOR_BUFFER_BIT);
    square();
    glutSwapBuffers ();
}

void displays()
{
    glClear (GL_COLOR_BUFFER_BIT);
    square();
    glFlush();
}
```



$$x = 25\cos\theta$$
$$y = 25\sin\theta$$

$$25\cos(\theta + \tfrac{\pi}{2}) = -25\sin\theta = -y$$
$$25\sin(\theta + \tfrac{\pi}{2}) = 25\cos\theta = x$$

A.6 simple-double.c (4/7)

```
void spinDisplay (void)
{
    spin = spin + 2.0;
    if (spin > 360.0) spin = spin - 360.0;
            x= 25.0*cos(DEGREES_TO_RADIANS * spin);
            y= 25.0*sin(DEGREES_TO_RADIANS * spin);
            glutSetWindow(singleb);
            glutPostRedisplay();
            glutSetWindow(doubleb);
            glutPostRedisplay();
}

void myinit ()
{
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glColor3f (1.0, 1.0, 1.0);
    glShadeModel (GL_FLAT);
}
```

A.6 simple-double.c (5/7)

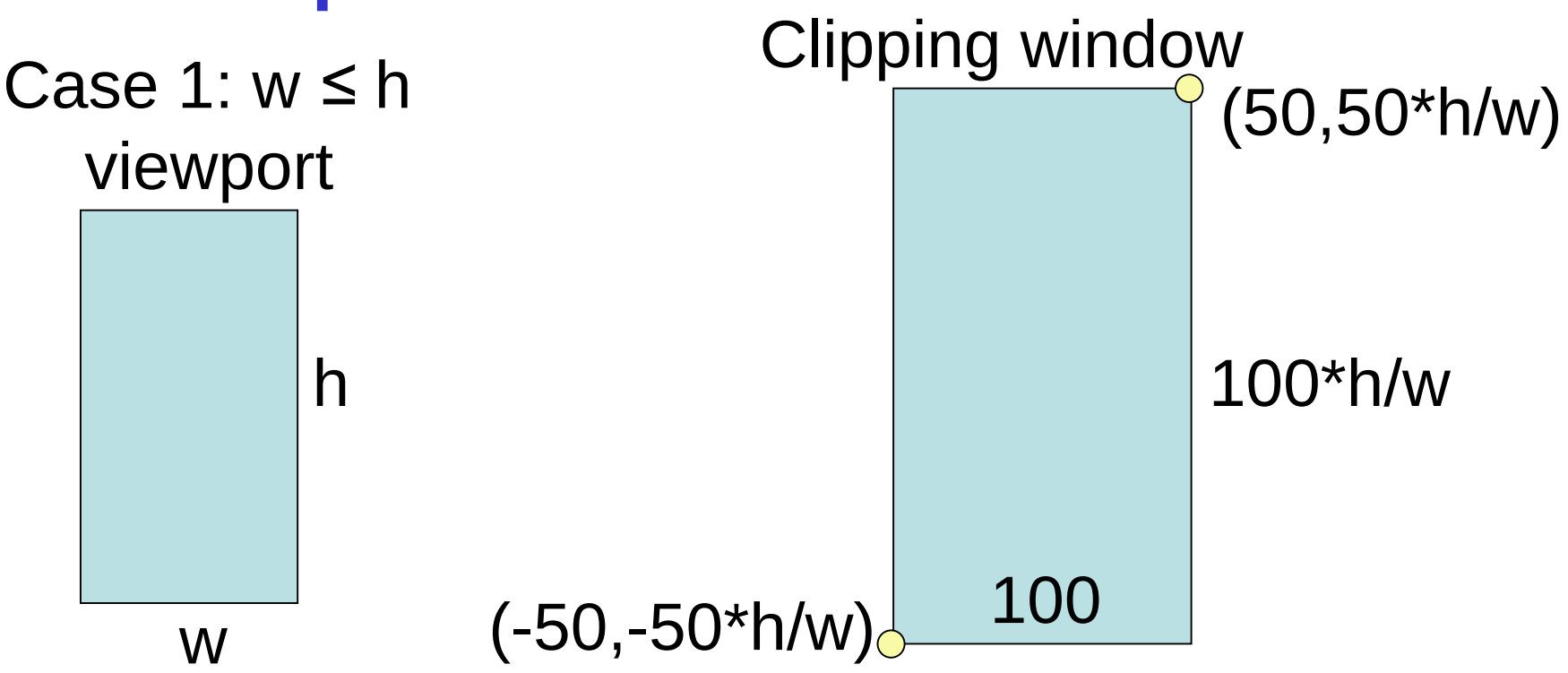
```
void mouse(int btn, int state, int x, int y)
{
If (btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        glutIdleFunc(spinDisplay);
If (btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
        glutIdleFunc(NULL);
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
            glOrtho (-50.0, 50.0, -50.0*(GLfloat)h/(GLfloat)w,
                50.0*(GLfloat)h/(GLfloat)w, -1.0, 1.0);
    else
            glOrtho (-50.0*(GLfloat)w/(GLfloat)h,
                50.0*(GLfloat)w/(GLfloat)h, -50.0, 50.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity ();

}
```

A.6 simple-double.c (6/7)

```
/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        singleb=glutCreateWindow("single buffered");
        myinit ();
        glutDisplayFunc(displays);
        glutReshapeFunc (myReshape);
        glutIdleFunc (spinDisplay);
        glutMouseFunc (mouse);

        glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowPosition(500,0);
        doubleb=glutCreateWindow("double buffered");
        myinit ();
        glutDisplayFunc(displayd);
        glutReshapeFunc (myReshape);
        glutIdleFunc (spinDisplay);
        glutMouseFunc (mouse);

        glutMainLoop();
}
```

Reshape Function

Case 1: w ≤ h



Case 2: w > h