UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4370 Fall 2023

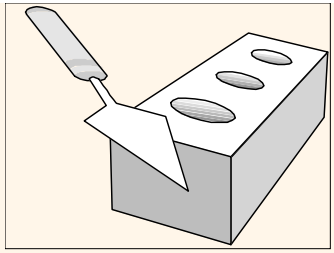
Interactive Computer Graphics

M & W 5:30 to 7:00 PM

Prof. Victoria Hilford

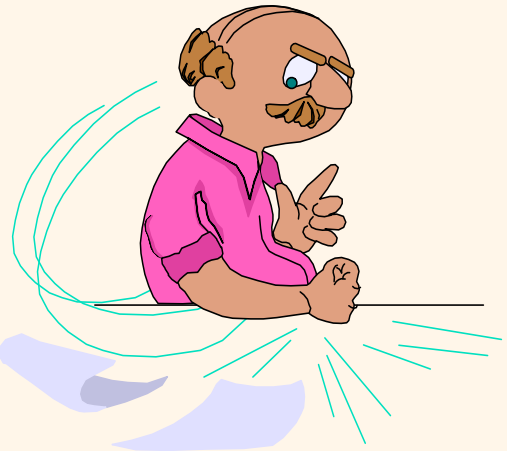
PLEASE TURN your webcam ON

NO CHATTING during LECTURE



COSC 4370

5:30 to 7



PLEASE

LOG IN

CANVAS

Please close all other windows.

From 5:30 to 6:45 PM – 75 minutes.

09.06.2023 (W 5:30 to 7) (5)	Homework 2	Lecture 3
09.11.2023 (M 5:30 to 7) (6)		Math Review 2
09.13.2023 (W 5:30 to 7) (7)	Homework 3	Lecture 4
09.18.2023 (M 5:30 to 7) (8)		PROJECT 1
09.20.2023 (W 5:30 to 7) (9)		EXAM 1 REVIEW
09.25.2023 (M 5:30 to 7) (10)		EXAM 1

COSC 4370 – Computer Graphics

Lecture 3

Chapter 3

SUMMARY and NOTES

The **interactive** aspects make the field of computer graphics **exciting and fun**. Although our **API**, **OpenGL**, is independent of any operating or window system, we recognize that any program must have at least minimal interaction with the rest of the computer system. We handled simple interactions by using a simple toolkit, **GLUT**, whose **API** provides the necessary additional functionality, **without being dependent on a particular operating or window system**.

From the **application programmers** perspective, various characteristics of interactive graphics are shared by most systems. We see the graphics part of the system as a server, consisting of a raster display, a keyboard, and a pointing device. In almost all workstations, we have to work within a multiprocessing, windowed environment. Most likely, many processes are executing concurrently with the execution of your graphics program. However, the window system **allows us to write programs for a specific window that act as though that window were the display device of a single-user system**.

The use of **logical devices within the application program frees the programmer from worrying about the details of particular hardware**. Within the environment that we have described, **event-mode input is the norm**. Although the other forms are available **request mode** is the normal method used for keyboard input event-mode input gives us far more flexibility in the design of **interactive programs**.

Interactive computer graphics is a powerful tool **with unlimited applications**. At this point, you should be **able to write fairly sophisticated interactive programs**.

Input and Interaction
Chapter 3

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

GLUT toolkit

OpenGL window

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

Graphical Input

- Devices can be described either by

Physical properties

- Mouse
- Keyboard
- Trackball

Logical Properties

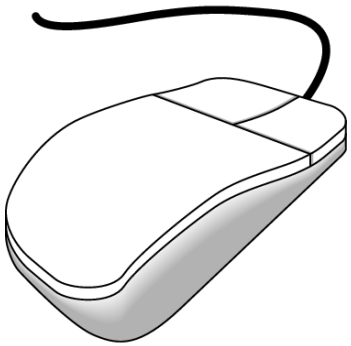
- What is returned to program via **API**
  - A position
  - An object identifier

Modes

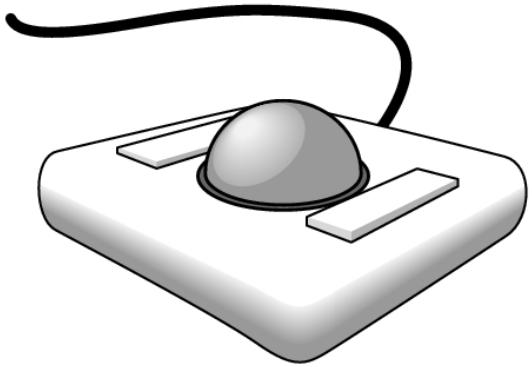
- How and when input is obtained
  - Request **or** event-driven

The University of New Mexico

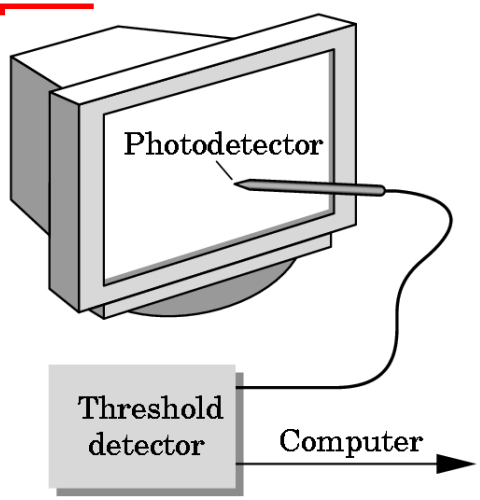
Physical Devices



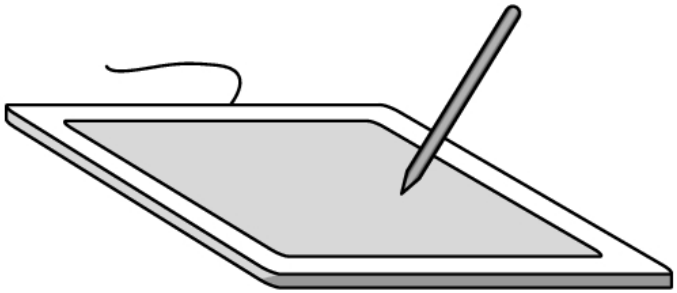
mouse



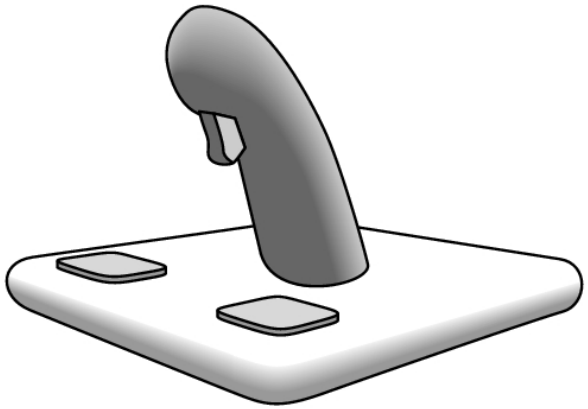
trackball



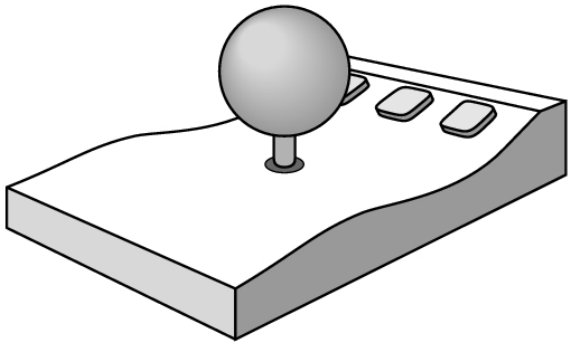
light pen



data tablet



joy stick



space ball

Treat input devices as **logical devices** whose properties are specified in terms of what they do from the perspective of the **application program.**

Incremental (Relative) Devices

- Devices such as the **data tablet** return a position directly to the operating system (**discrete**)

- Devices such as the **mouse**, **trackball**, and **joystick** return incremental inputs (or velocities) to the operating system (**analog**)
  - 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**

Logical Devices

- Consider the C++ and C code

  C++: `cin >> x;`

  C: `scanf (x);`

- What is the input device?

  Can't 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**

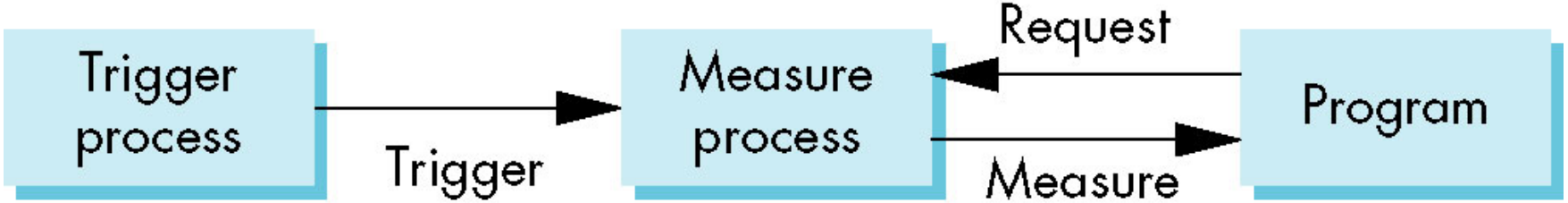
- Two older **API**s (**GKS**, **PHIGS**) defined six types of logical input

Locator: return a position	World X,Y
Pick: return ID of an **Object**	Selection
Keyboard: return strings of characters	String
Stroke: return array of positions	Transfer
Valuator: return floating point number	Analog
Choice: return one of n items	Integer

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

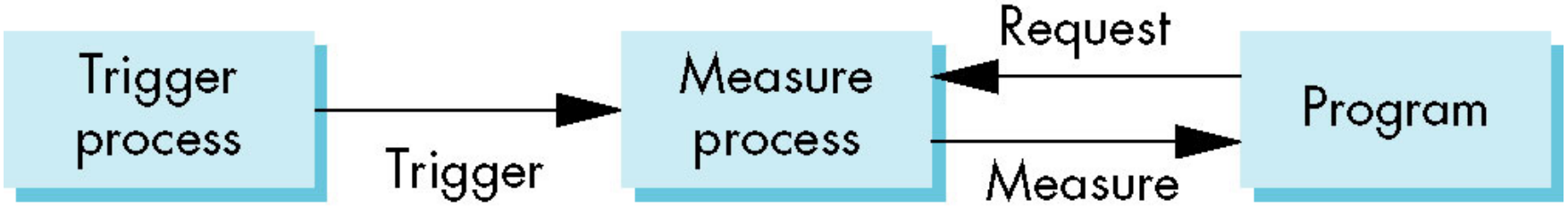


Request Mode

Input provided to program **only when user** triggers the device

Typical of **keyboard** input

- Can erase (backspace), edit, correct until **enter key** (the trigger) is depressed



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

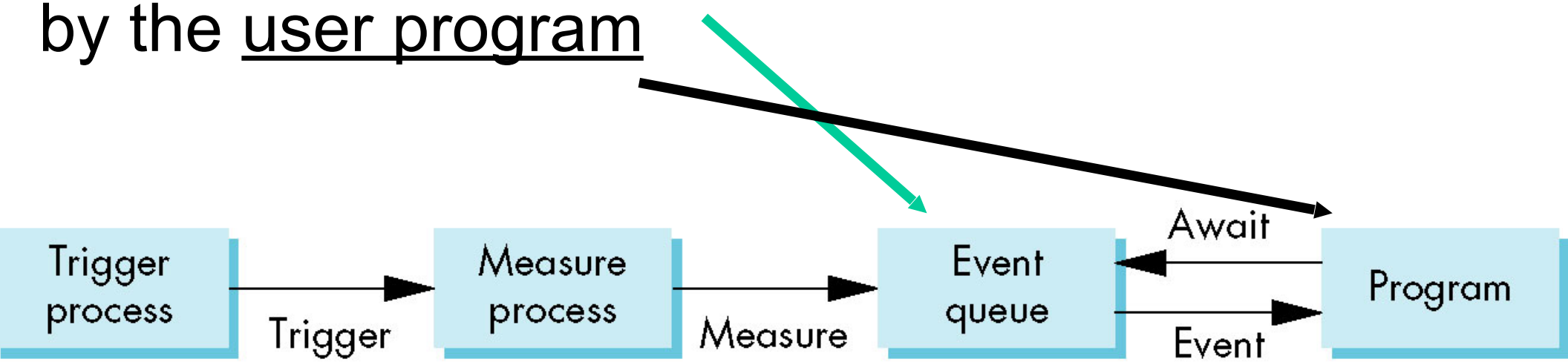
GLUT callbacks

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

- glutDisplayFunc
- glutMouseFunc
- glutReshapeFunc
- glutKeyboardFunc
- glutIdleFunc
- glutMotionFunc,
- glutPassiveMotionFunc

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

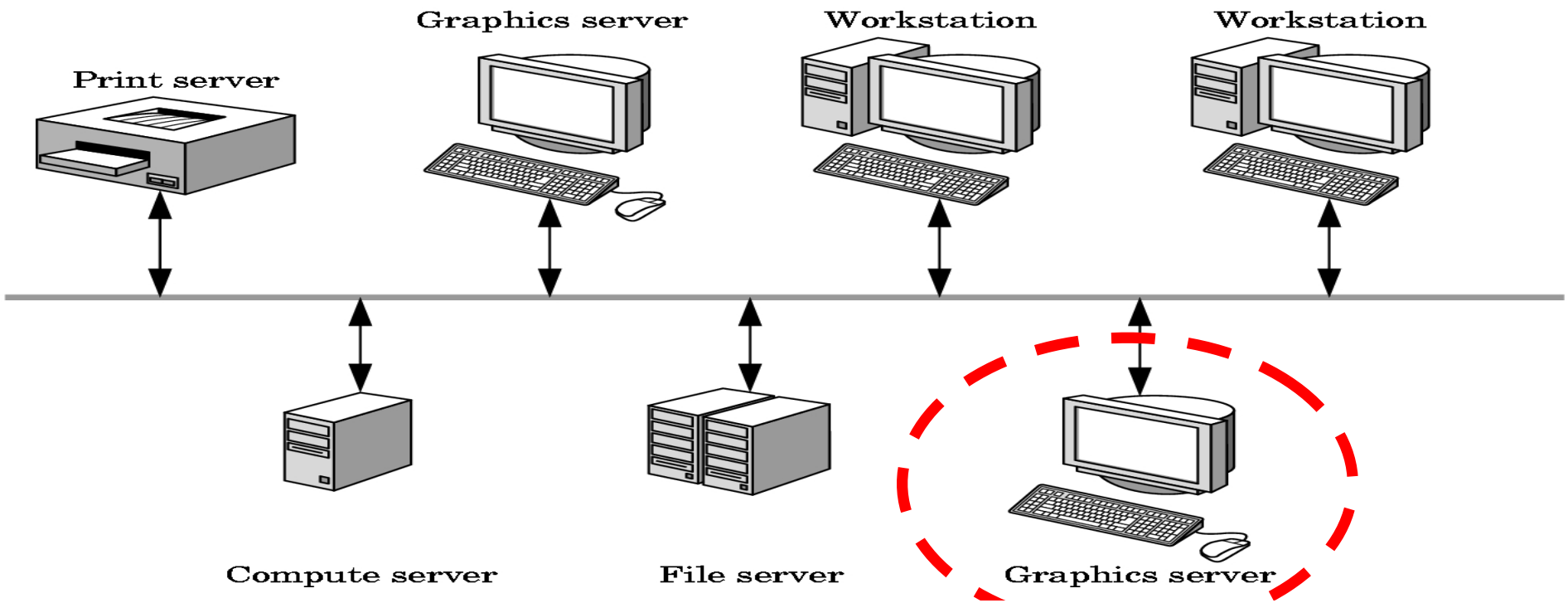


X Window Input

The X Window System introduced a client-server model for a network of workstations

Client: OpenGL program

Graphics Server: bitmap display with a pointing device and a keyboard



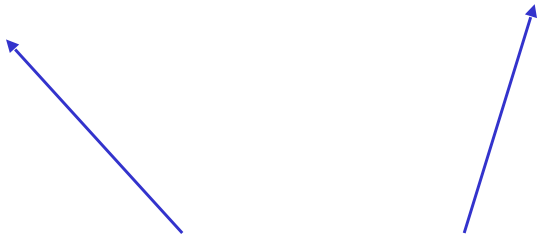
Event Types

- **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 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 function to be executed

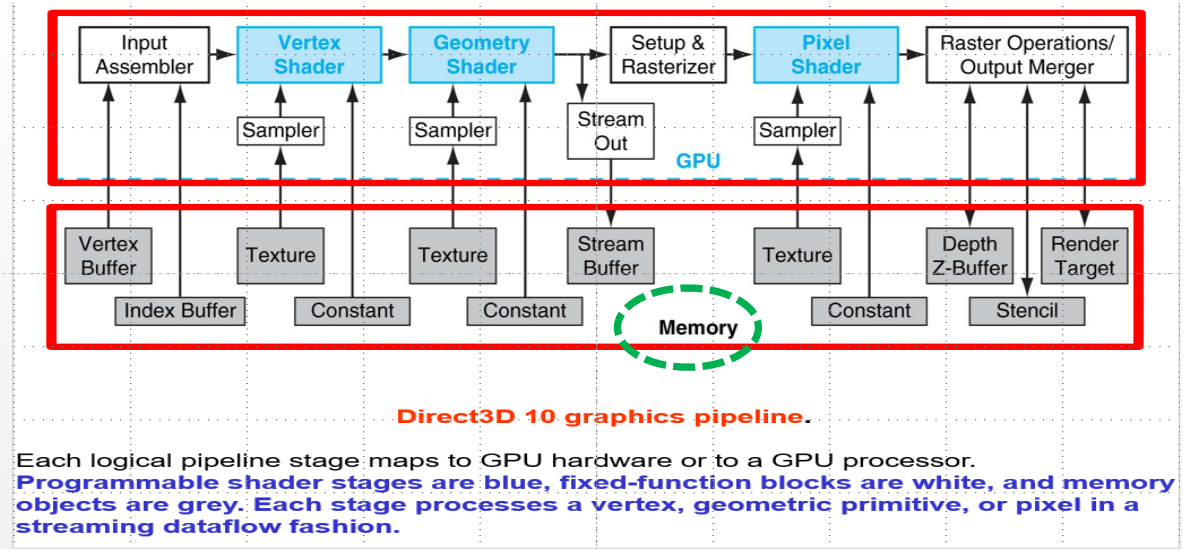
Every **GLUT** program must have a **display** callback

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

Objectives

- Introduce the basic **input devices**
  - Physical Devices
  - Logical Devices
  - Input Modes
- Event-driven input



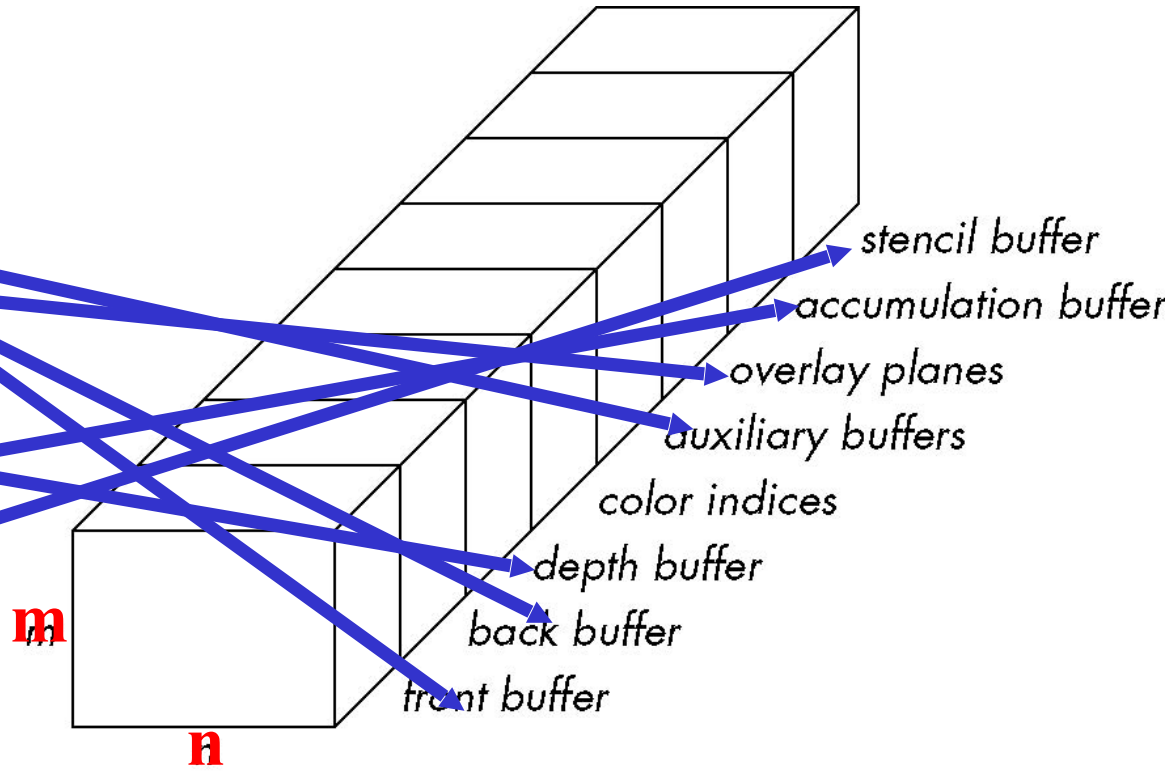
Direct3D 10 graphics pipeline.

Each logical pipeline stage maps to GPU hardware or to a GPU processor. Programmable shader stages are blue, fixed-function blocks are white, and memory objects are grey. Each stage processes a vertex, geometric primitive, or pixel in a streaming dataflow fashion.

- Introduce **double buffering** for smooth animations
- Programming event input with **GLUT**

OpenGL Buffers

- Color buffers can be displayed
  - Front
  - Back
  - Auxiliary
  - Overlay
- Depth
- Accumulation
  - High resolution buffer
- Stencil
  - Holds masks



Animating a Display

When we redraw the display through the **display** callback, we usually start by clearing the window

`glClear()`

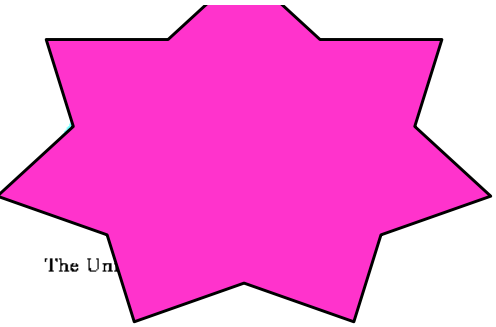
then draw the altered display

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 `single_double.c` for an example with a rotating cube



LECTURE 3 CLASS PARTICIPATION

VH, publish

Class Participation on Lecture 3

VH, publish

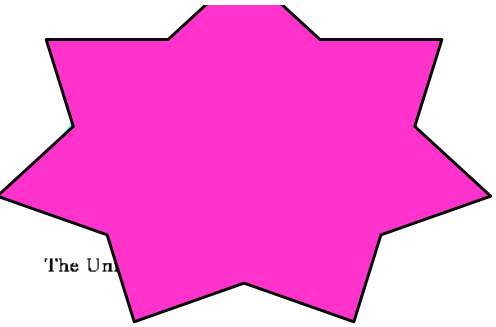
CLASS PARTICIPATION - 15%

15% of Total

Class PARTICIPATION on Lecture 3

Not available until Sep 6 at 6:00pm | Due Sep 6 at 7pm | 100 pts

VH, publish



The Un

Class PARTICIPATION on Lecture 3

Publish | Edit | ⋮

Download and complete this word document.

Class Participation on Lecture 3.doc

single_double.c

square.c

pick.c

You will be prompted when to Upload completed document to CANVAS as **score.doc** (example 100.doc).

Warning:

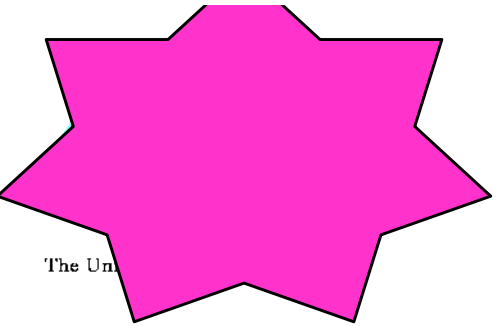
TA, at random, will inspect the Uploaded document.

If you score is not honestly entered you will get a zero.

Points 100

Submitting Nothing

Due	For	Available from	Until
Sep 6 at 7pm	Everyone	Sep 6 at 6pm	Sep 6 at 7pm



The Un

single_double.c **VH, publish**

Name: _____

Total score: ☐

Class PARTICIPATION on Lecture 3.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

Build a Lecture3 C++ Empty Project

1. (33 points) Download single_double.c from CANVAS and add it to Project.

Self Graded - correctly

a. Build and run the project. Use the middle button to stop spinning.

b. Slow down the spinning. Rebuild and run the project.

(print screen)

Animating a Display

Hence we can see partially drawn display

Program `single_double.c` rotating cube

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

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

    glutMainLoop();
}
```

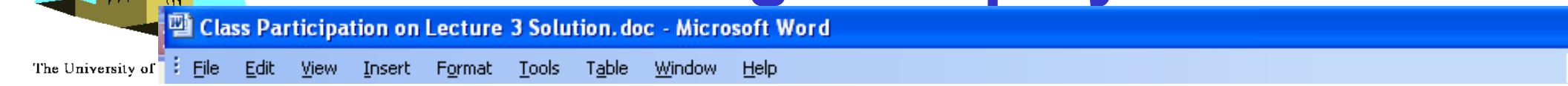
single buffered

double buffered

single_double 1

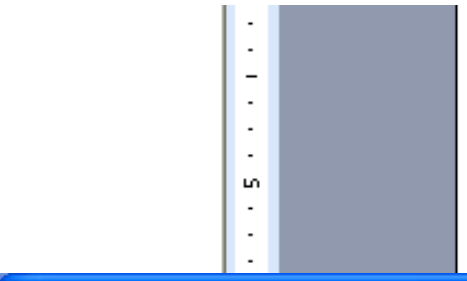
Lecture 3 Class Participation 1

Animating a Display

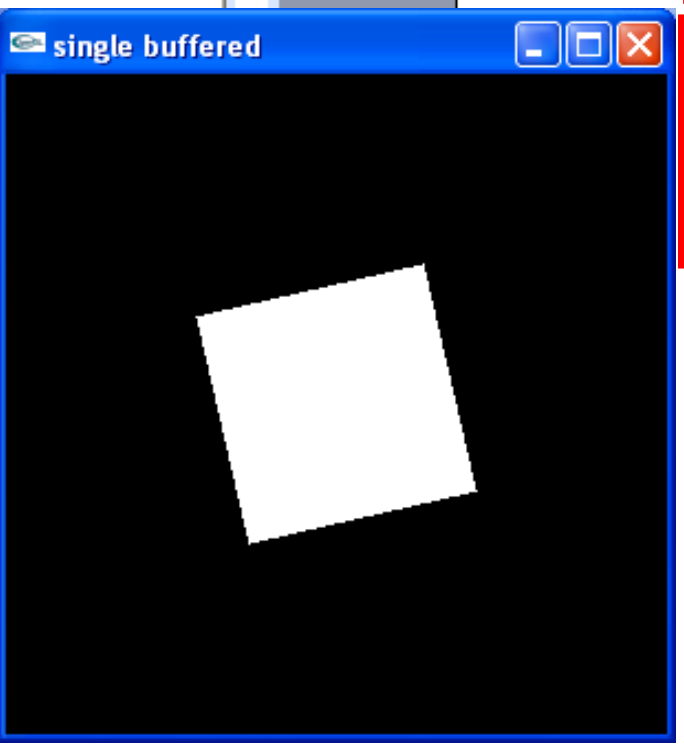


Hence we can see partially drawn display

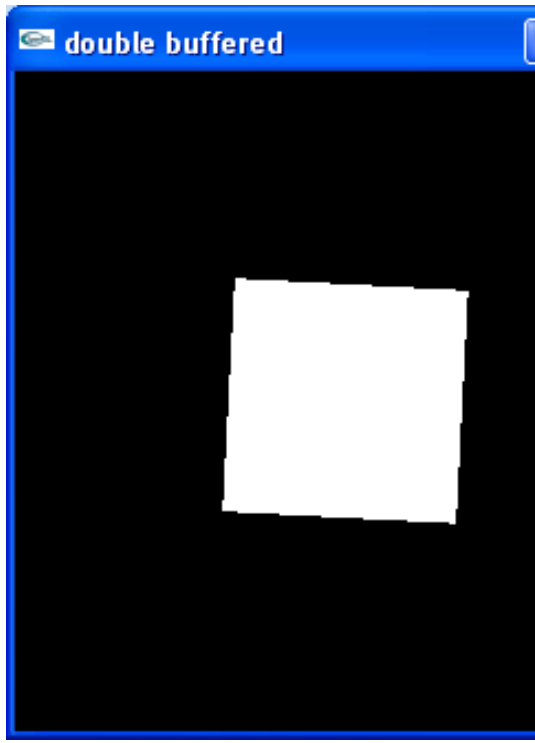
Program `single_double.c` rotating cube



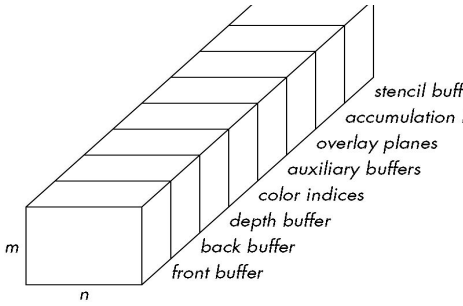
```
void displayd()
{
    glClear (GL_COLOR_BUFFER_BIT);
    square();

    glutSwapBuffers ();
}
```




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

    glFlush();
}

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();
}
```



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()
}
```

```
33
void displayd()
{
    glClear (GL_COLOR_BUFFER_BI
    square();
    glutSwapBuffers ();
}

void displays()
{
    glClear (GL_COLOR_BUFFER_BI
47
```

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**

```
GLfloat t; /*global */

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

```
19
20  static GLfloat spin = 0.0;
21  GLfloat x, y;
22  int singleb, doubleb;
23
24  void square()
25  {
26      glBegin(GL_QUADS);
27          glVertex2f( x, y);
28          glVertex2f(-y, x);
29          glVertex2f(-x,-y);
30          glVertex2f( y,-x);
31      glEnd();
32  }
33
34
35  void displayd()
```

Working with Callbacks
Chapter 3.6

Objectives

Learn to build interactive programs using **GLUT callbacks**

- **Mouse**
- **Keyboard**
- **Reshape**

Introduce **menus** in **GLUT**

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 coordinates** (**x**, **y**)

The Mouse callback

cture3 (Running) - Microsoft Visual Studio

Edit View Project Build Debug Tools Visual Assert Test Window Help

Debug Win32

Hex

Thread: Stack Frame:

n Explorer - Solution 'L...

Solution 'Lecture3' (1 project)
Lecture3
Header Files
Resource Files
Source Files
single_double.c

single_double.c*

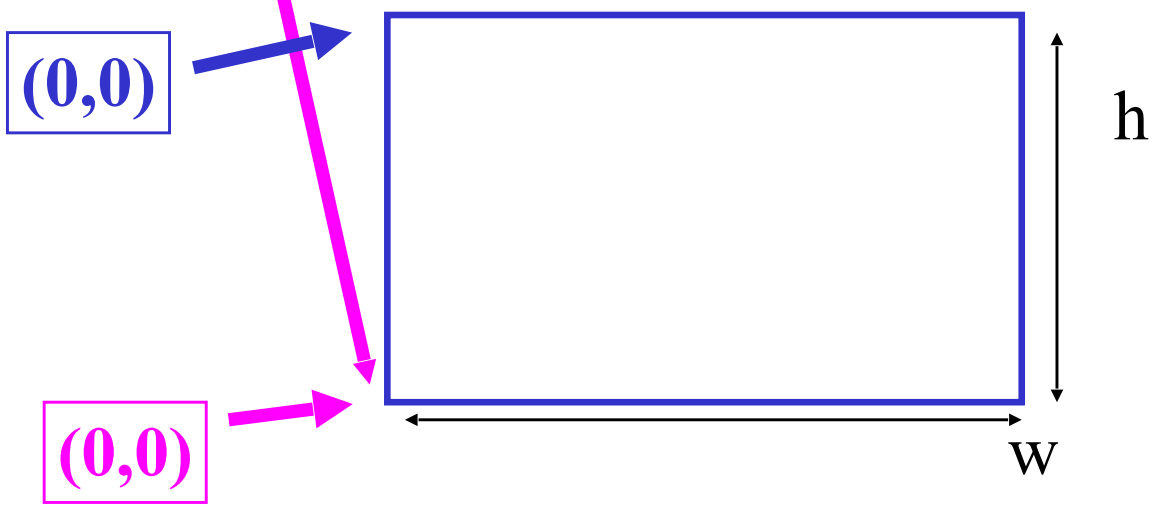
(Global Scope) mouse(int btn, int state, int x, int y)

```
60
61          glutSetWindow(singleb);
62          glutPostRedisplay();
63          glutSetWindow(doubleb);
64          glutPostRedisplay();
65      }
66
67  void myinit ()
68  {
69          glClearColor (0.0, 0.0, 0.0, 1.0);
70          glColor3f (1.0, 1.0, 1.0);
71          glShadeModel (GL_FLAT);
72  }
73
74  void mouse(int btn, int state, int x, int y)
75  {
76  if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)      glutIdleFunc(spinDisplay);
77  if(btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)    glutIdleFunc(NULL);
78  }
79
80  void myReshape(int w, int h)
81  {
82          glViewport(0, 0, w, h);
83          glMatrixMode(GL_PROJECTION);
84          glLoadIdentity();
```

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`

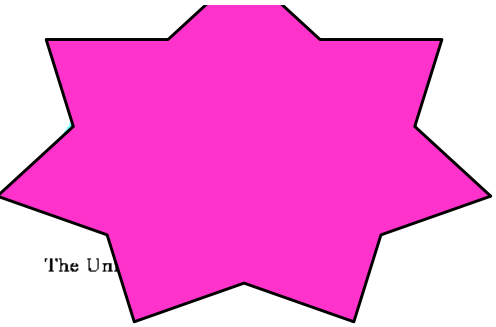
to obtain any value that is **part of the state**

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);
                                }
```



square.c

VH, publish

The Un

Name: _____

Total score: ☐

Class PARTICIPATION on Lecture 3.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

2. (33 points) Download square.c from CANVAS and add it to Project.

a. Build and run the project. Use the left button to draw and the middle button to stop the program.

Self Graded - correctly

b. Make squares larger. Rebuild and run the project.

(print screen)

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(){}
```

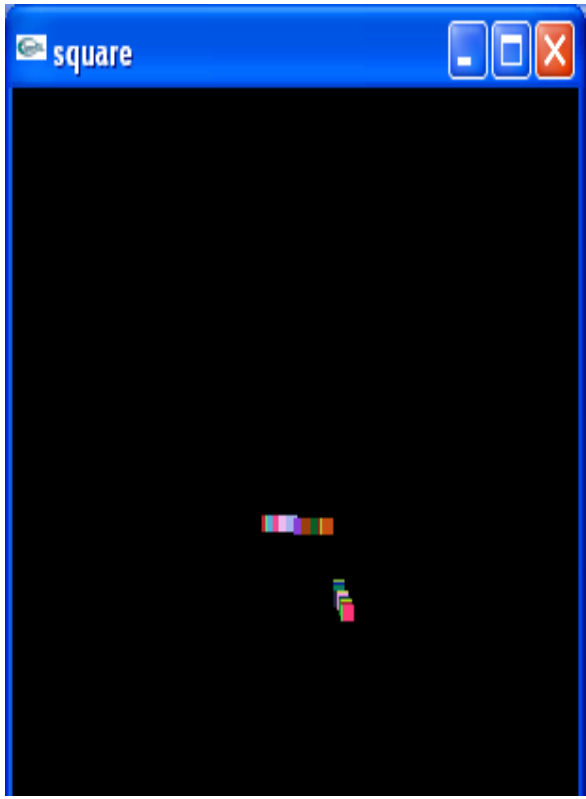
square.c 2

Lecture 3 Class Participation 2

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

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

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();
}
```



Drawing squares at cursor location

The

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

```
void drawSquare(int x, int y)
{
    y = wh-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 continuously as long as a mouse button is depressed by using the **Motion** callback (The **motion** callback for a window is called when the **mouse moves within the window while one or more mouse buttons are pressed**)

```
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' .....`

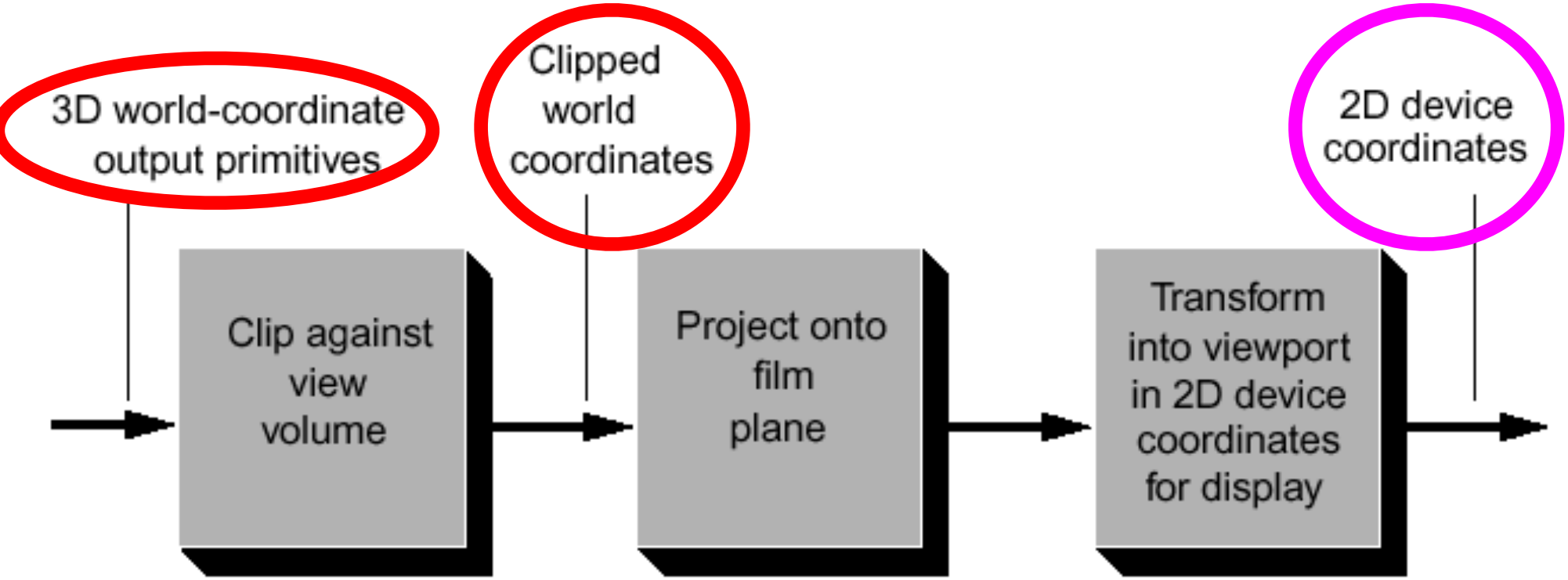
- Can also check of one of the **modifiers**
  - `GLUT_ACTIVE_SHIFT`
  - `GLUT_ACTIVE_CTRL`
  - `GLUT_ACTIVE_ALT`

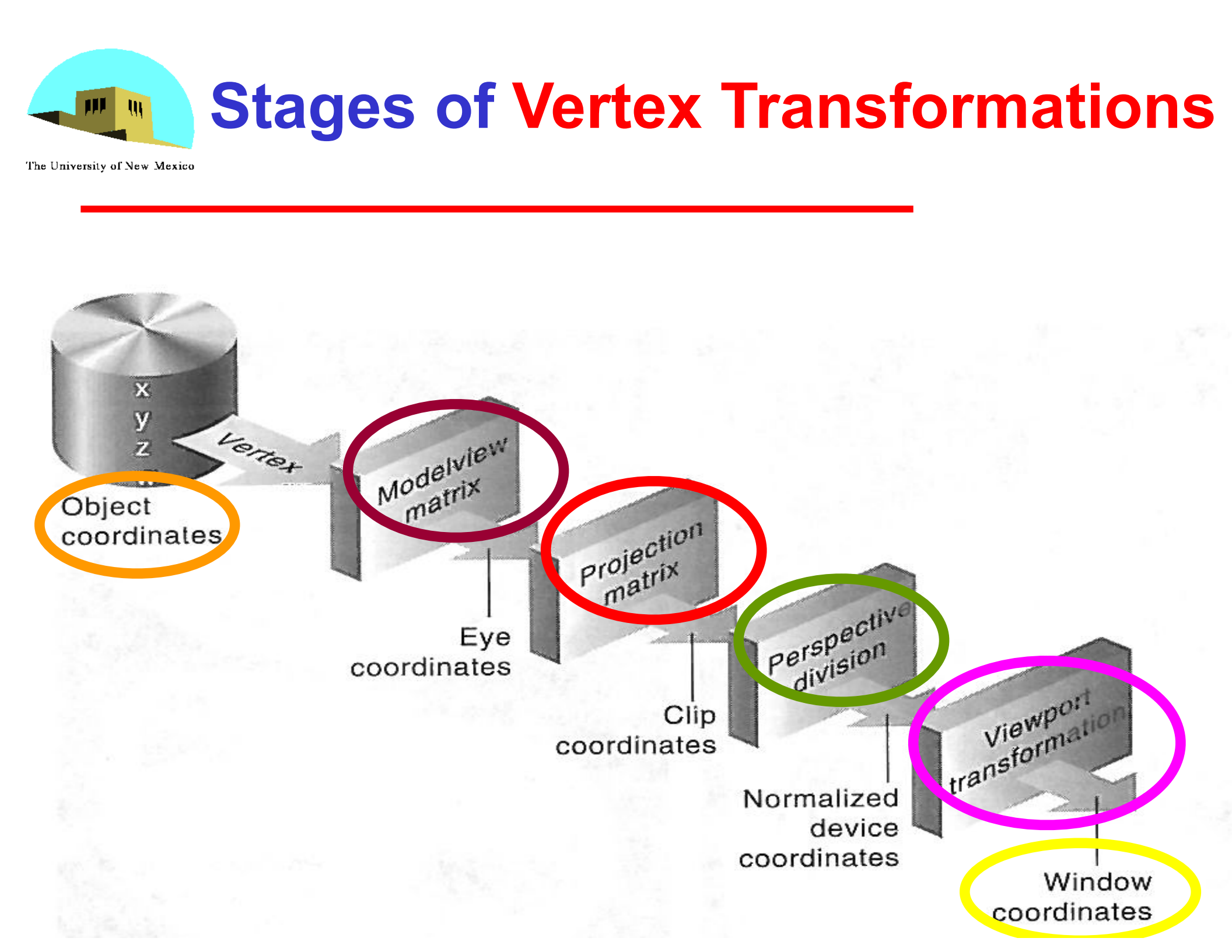
  is depressed by

  `glutGetModifiers()`

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

Viewing transformations




The University of New Mexico
Stages of Vertex Transformations
x
y
z
Vertex
Object coordinates
Modelview matrix
Eye coordinates
Projection matrix
Clip coordinates
Perspective division
Normalized device coordinates
Viewport transformation
Window coordinates

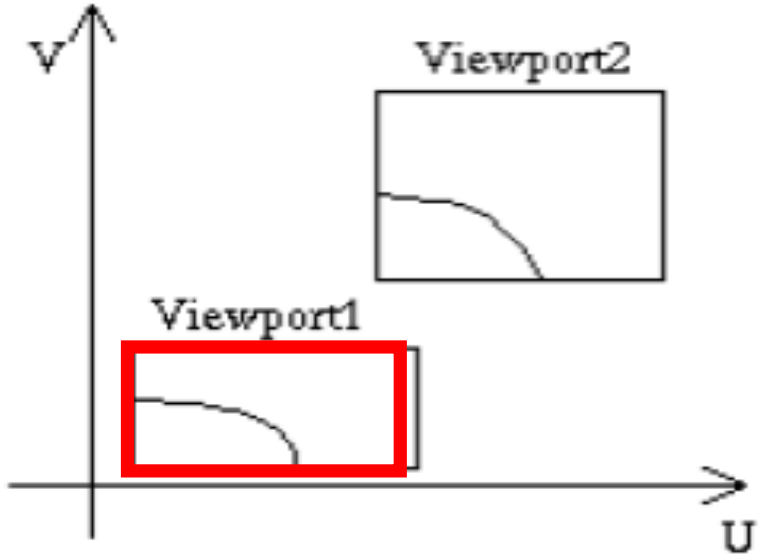
Coordinate Systems

Screen Coordinates: The coordinate system used to address the screen (device coordinates) **u v**

World Coordinates: A user-defined application specific coordinate system having its own units of measure, axis, origin, etc. **x y**

Window: The rectangular region of the world that is visible.

Viewport: The rectangular region of the screen space that is used to display the window.



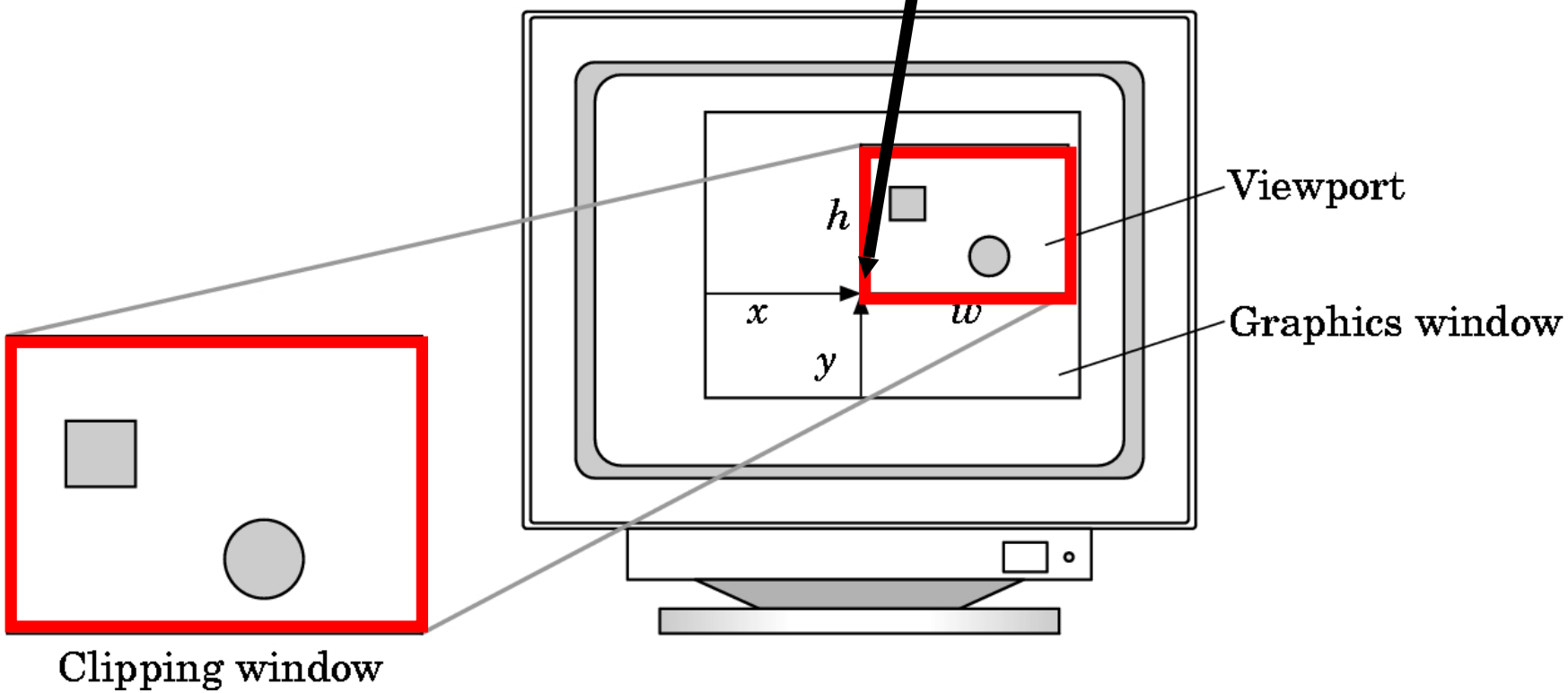
Screen

Viewports

- Do not have use the entire window for the image:

```
glViewport(x,y,w,h)
```

- Values in pixels (screen coordinates)



Window to Viewport Transformation

Want to find the transformation matrix that maps the window in world coordinates to the viewport in screen coordinates.

Viewport: (u, v space) denoted by: $u_{min}, v_{min}, u_{max}, v_{max}$

Window: (x, y space) denoted by: $x_{min}, y_{min}, x_{max}, y_{max}$

The transformation:

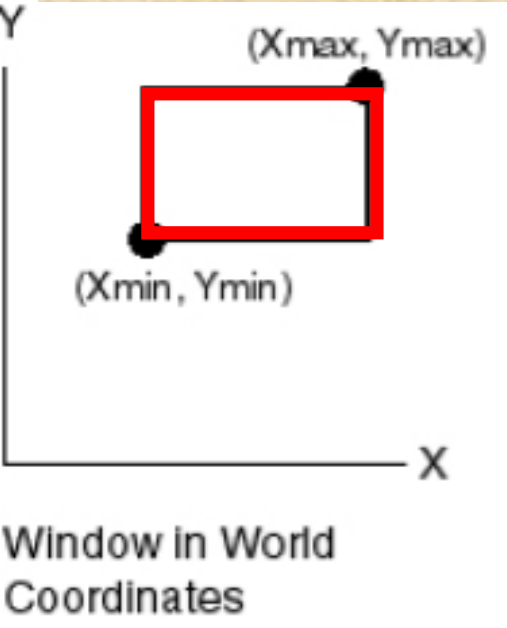
- Translate the window to the origin
- Scale it to the size of the viewport
- Translate it to the viewport location

$$M_{WV} = T(U_{min}, V_{min}) \; S(S_x, S_y) \; T(-x_{min}, -y_{min})$$

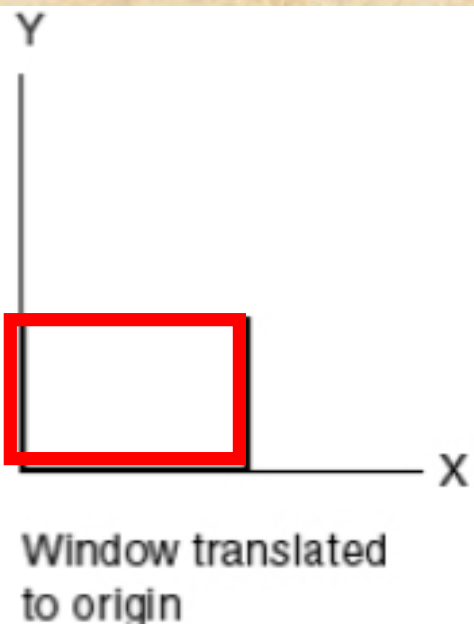
$$S_x = (U_{max} - U_{min}) / (x_{max} - x_{min});$$

$$S_y = (V_{max} - V_{min}) / (y_{max} - y_{min});$$

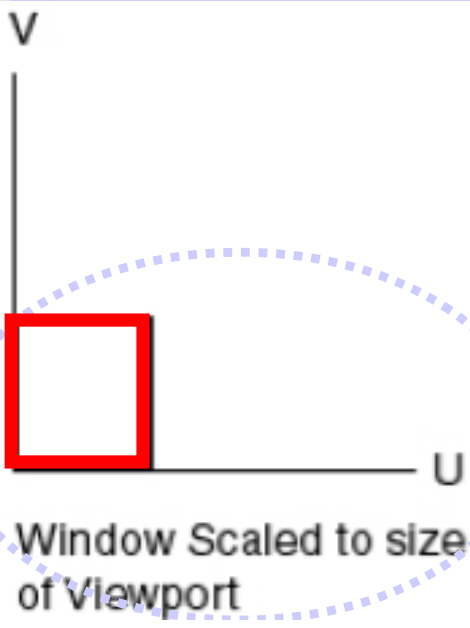
$$M_{WV} = \begin{bmatrix} S_x & 0 & (-x_{min} * S_x + U_{min}) \\ 0 & S_y & (-y_{min} * S_y + V_{min}) \\ 0 & 0 & 1 \end{bmatrix}$$



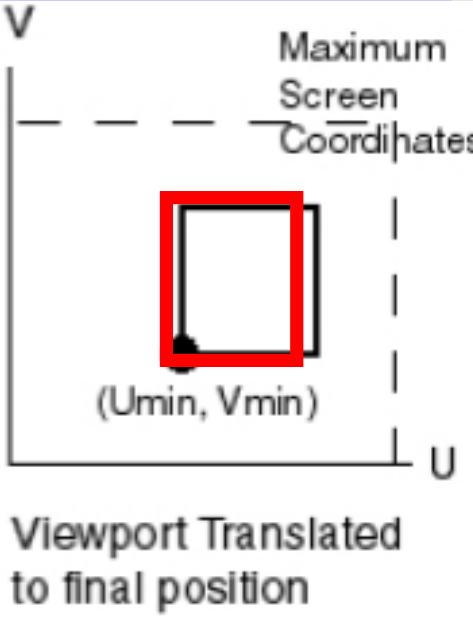
Window in World Coordinates



Window translated to origin



Window Scaled to size of Viewport



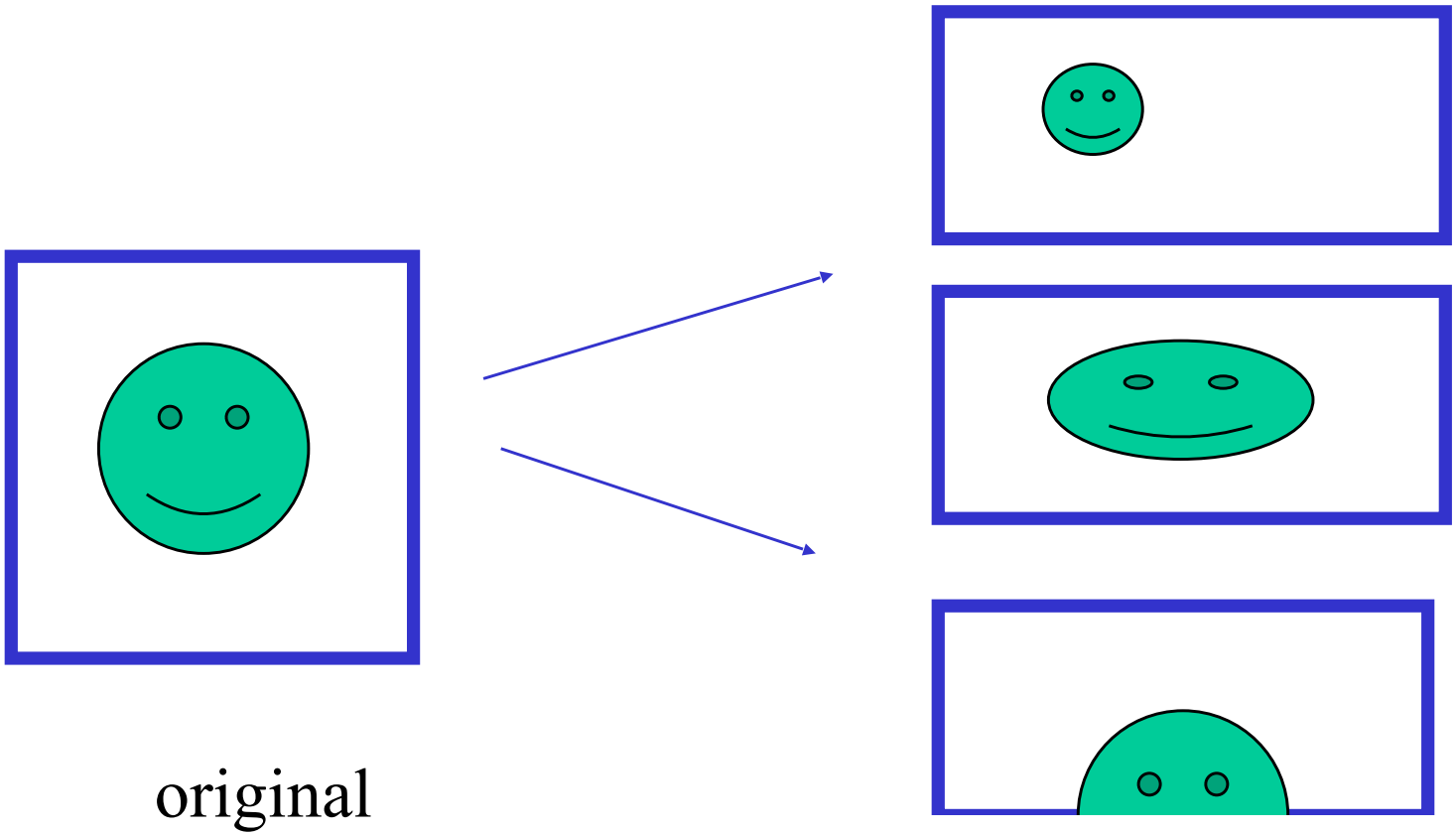
Viewport Translated to final position

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 possibilities



The Reshape callback

```
glutReshapeFunc(myreshape)
```

```
void myreshape(int w, int h)
```

Returns **w**idth and **h**eight of new window (in **pixels**)

A Redisplay is posted automatically at end of execution of the callback

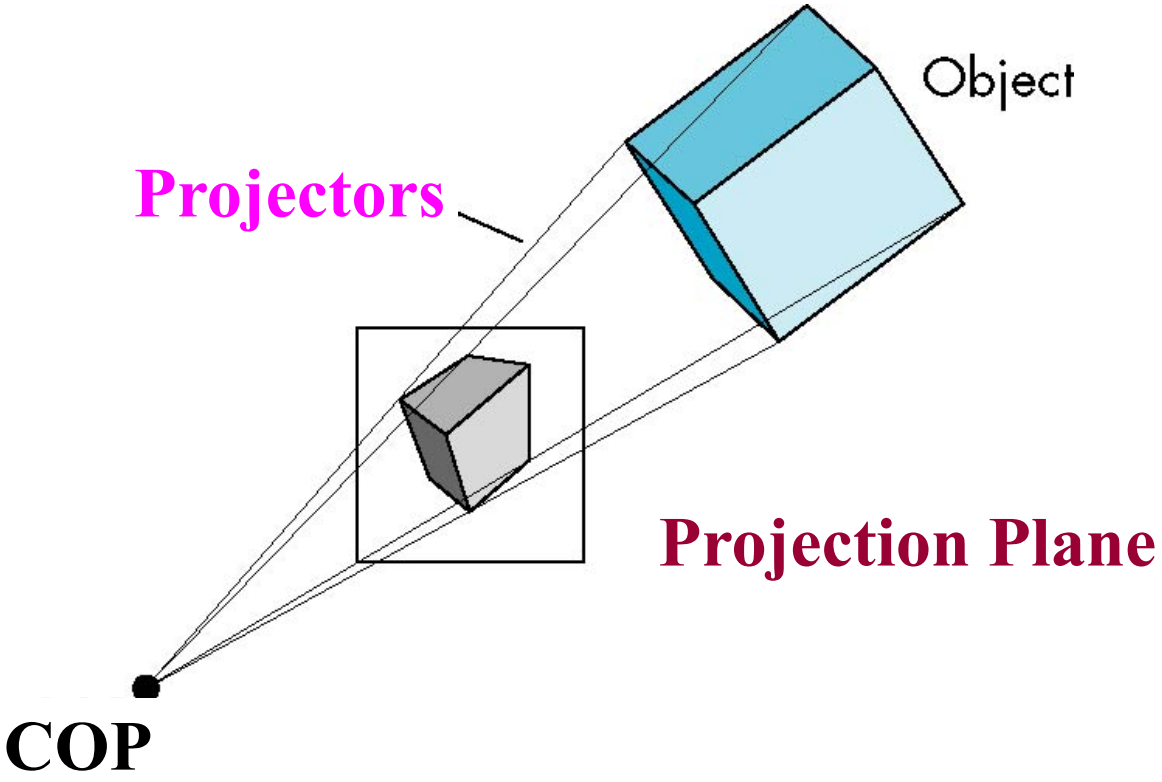
GLUT has a **default reshape callback** but you probably want to define your own

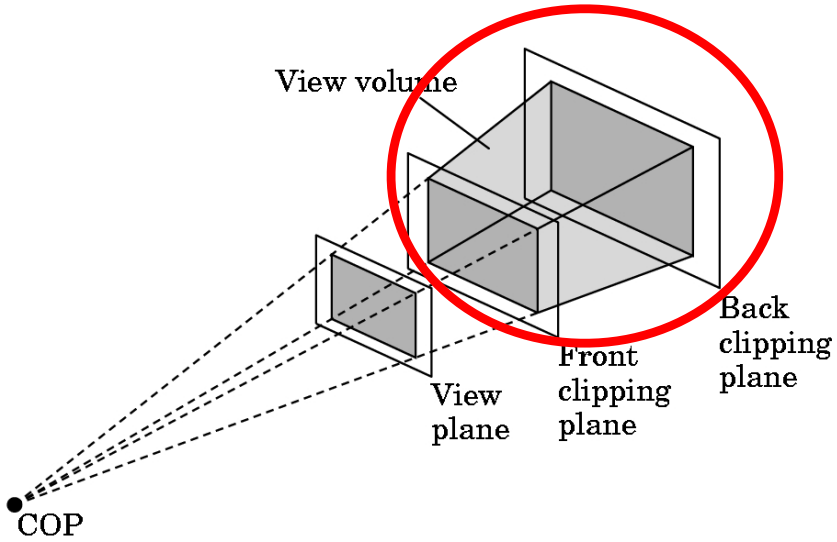
- The Reshape callback is good place to put viewing functions because it is invoked when the window is first opened

OpenGL functions for setting up transformations

modelling transformation *modelview matrix*	glTranslatef() glRotatef() glScalef()
viewing transformation *modelview matrix*	gluLookAt()
projection transformation ***projection matrix***	glFrustum() gluPerspective() glOrtho() gluOrtho2D()
viewing transformation	glViewport()

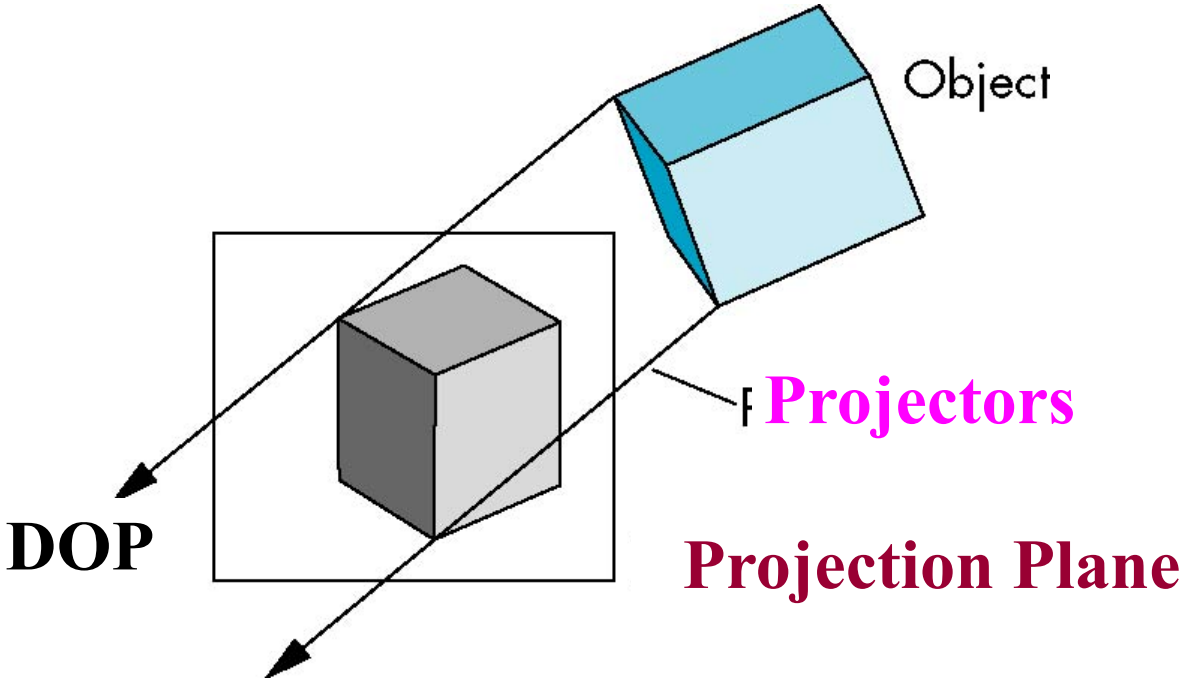
Perspective Projection

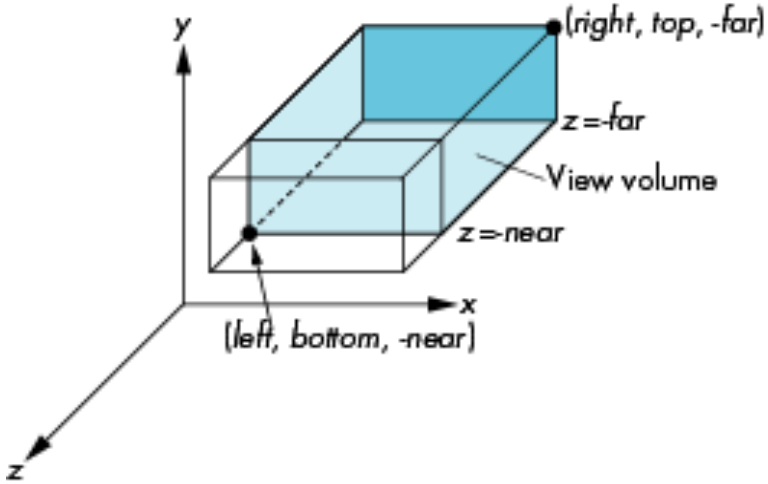




glPerspective(field_of_view, aspect_ratio, near, far);

Parallel (Orthographic) Projection



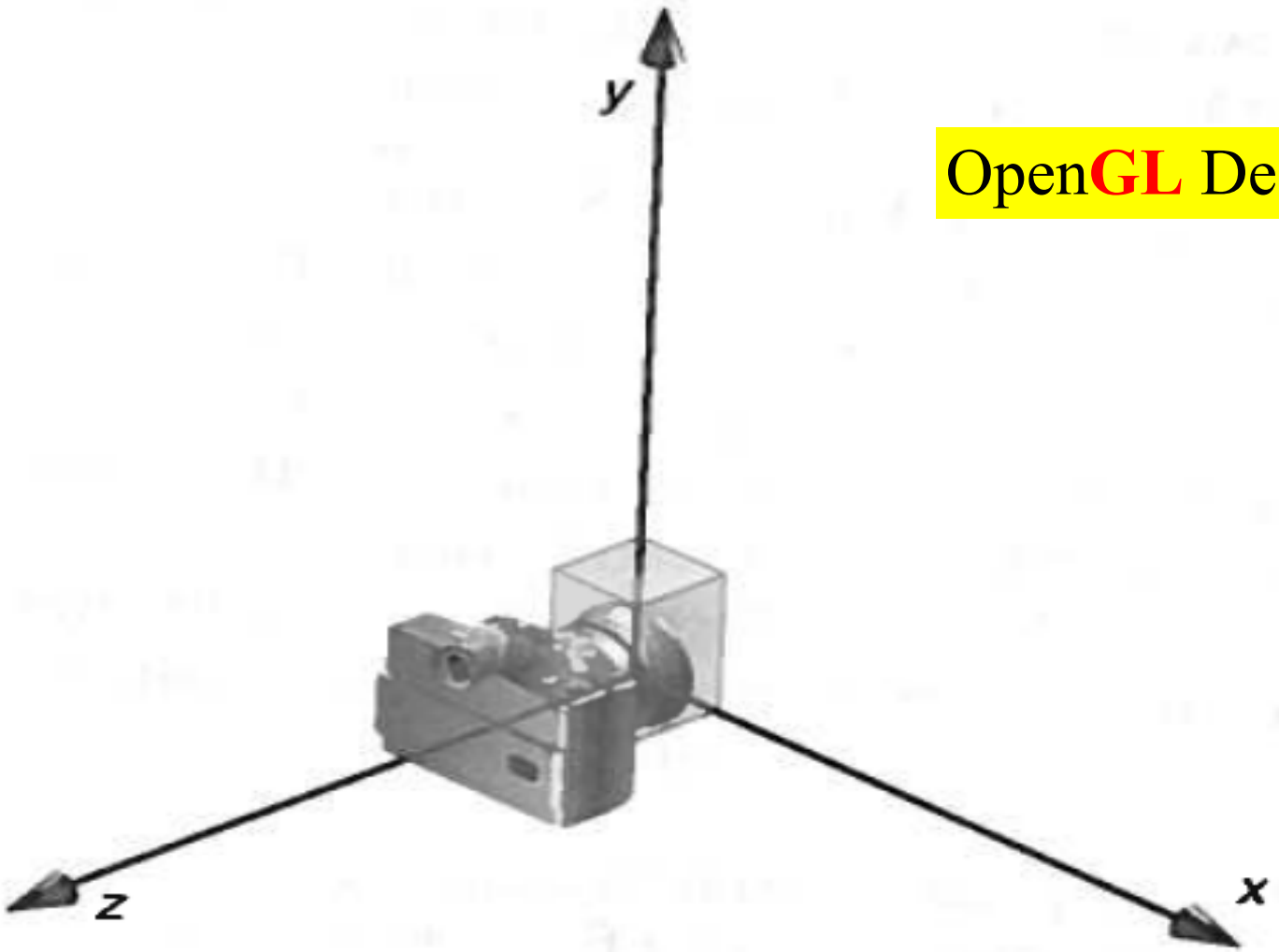


`near` and `far` **measured from camera**

```
glOrtho(left,right,bottom,top,near,far)
```

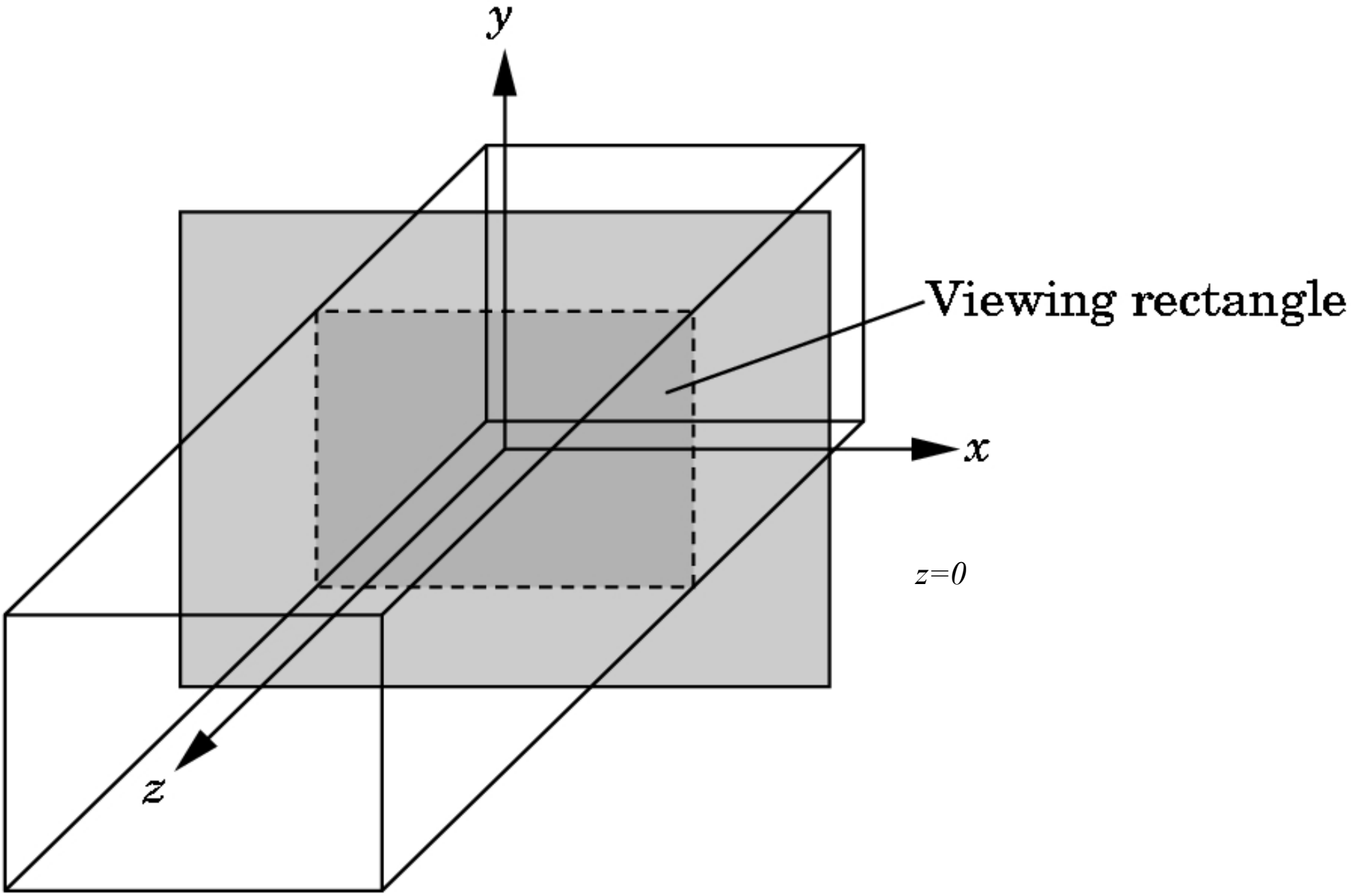
Default Camera Position

OpenGL Default



Object and Viewpoint at the Origin

gluOrtho2D(left,right,bottom,top)



Example Reshape

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

gluOrtho2D(left,right,bottom,top)

```
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 */
}
```

Objectives

Learn to build interactive programs using **GLUT callbacks**

- **Mouse**
- **Keyboard**
- **Reshape**

Introduce **menus** in **GLUT**

Menus

GLUT supports pop-up menus

A menu can have submenus

Three steps

Define entries for the menu **glutAddmenuEntry**

Define action for each menu item

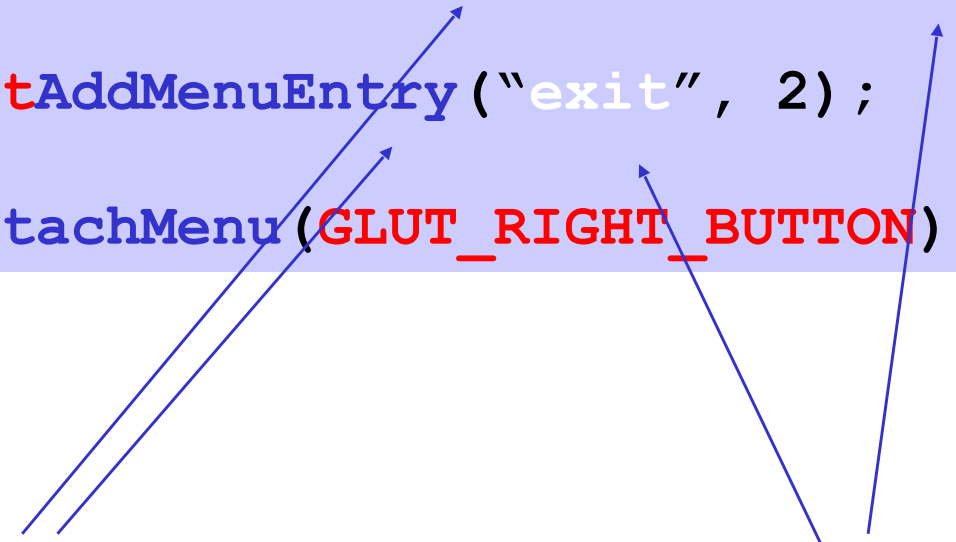
- Action carried out if entry selected **glClear();**

Attach menu to a mouse button **glutAttachMenu(GLUT_RIGHT_BUTTON);**

Defining a simple menu

- In **main.c**

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

    glutAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



clear screen
exit

entries that appear when **right button depressed**

identifiers

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

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
Chapter 3.12; 3.8; 3.4

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

Objectives

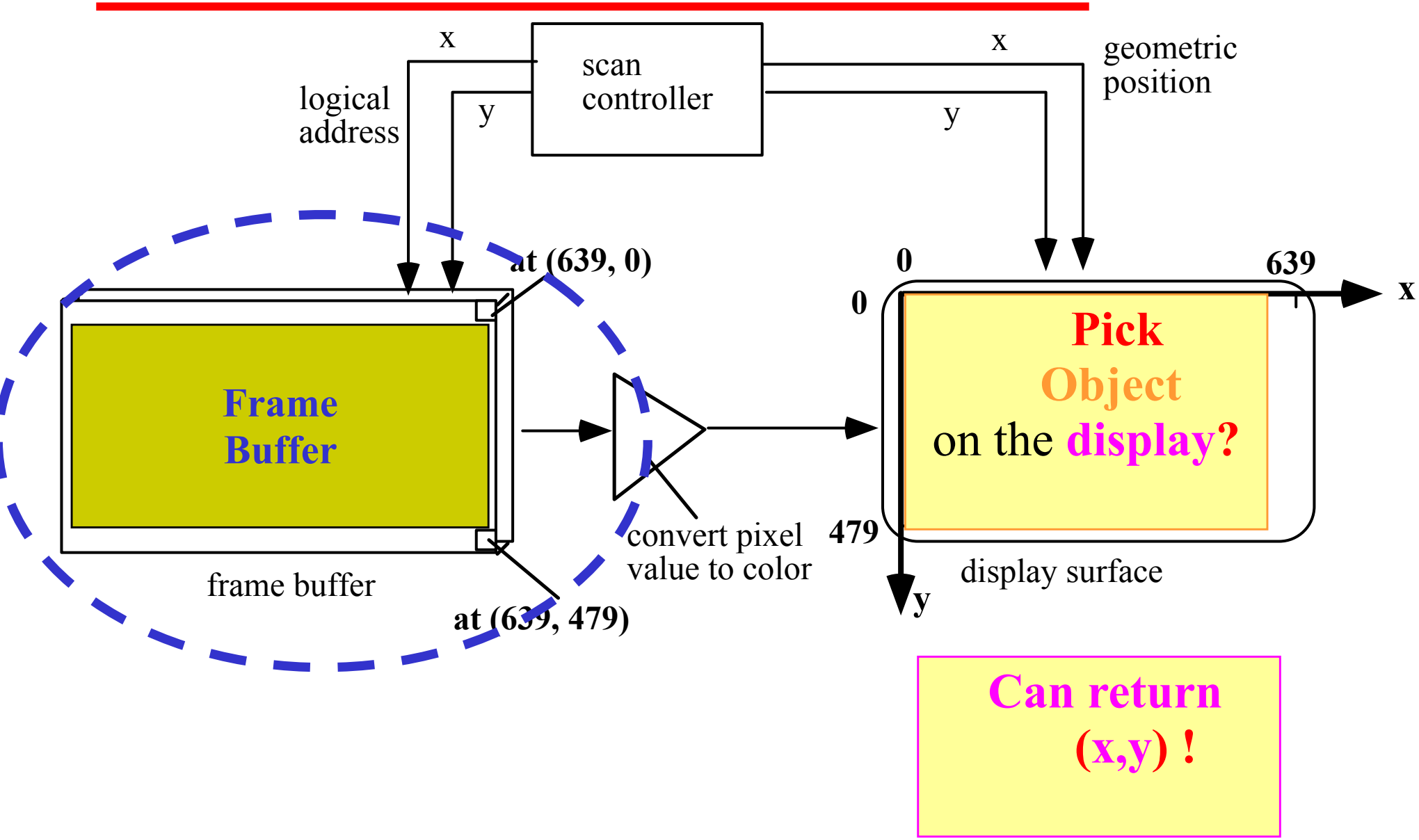
- Learn to build more sophisticated interactive programs using

  Picking
  - Select **Objects** from the display
  - Three methods

  Rubberbanding
  - Interactive drawing of lines and rectangles

  Display Lists
  - Retained mode graphics

Graphics Display Device Operation



The University of New Mexico

Picking

- 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

Identify a user-defined **Object** on the **display**

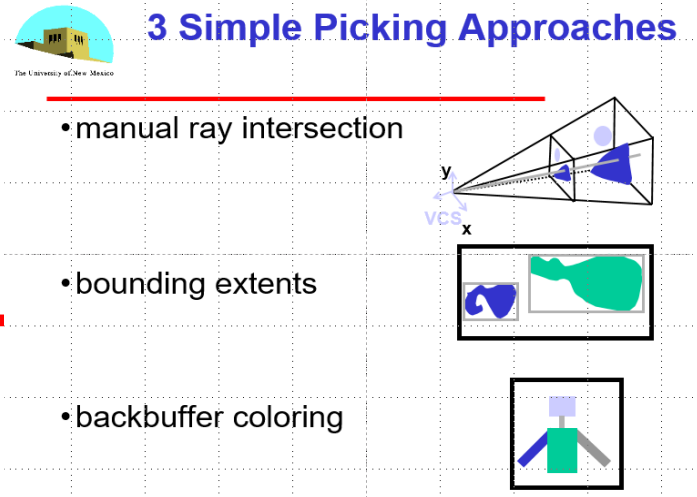
In principle, it should be simple because the mouse gives **the position** and we should be able to determine to which **object(s)** a position corresponds

Practical difficulties

- Pipeline architecture is feed forward, hard to go from screen back to **world**
- Complicated by screen being 2D, **world** is 3D
- How close do we have to come to **Object** to say we selected it?

Converting from a location on the display to the corresponding primitive is not a direct calculation.

Three Approaches



- Hit list - **keep track of which primitives in a small clipping region are rendered into a region near the cursor**

  Most general approach but most difficult to implement

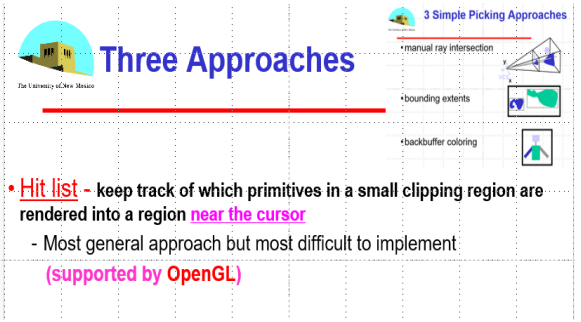
  (supported by OpenGL)

- Rectangular maps - **bounding boxes, or extents, for Objects of interest**

  Easy to implement for many applications

- Use back or some other buffer to store **Object ids** as the **Objects** are rendered

Rendering Modes



OpenGL can render in one of three modes selected by **`glRenderMode(mode)`**

`GL_RENDER`: normal rendering to the **frame buffer** (default)

`GL_FEEDBACK`: provides list of primitives rendered but no output to the **frame buffer**

`GL_SELECT`: Each primitive in the **view volume** generates a *hit record* that is placed in a ***name stack*** which can be examined later

GL SELECT Select Mode Functi

The University of New Mexico

`glSelectBuffer()`: specifies name buffer

`glInitNames()`: initializes name buffer

`glPushName(id)`: push id on name buffer

`glPopName()`: pop top of name buffer

`glLoadName(id)`: replace top name on buffer

id is set by application program to identify Objects

1. Using Select Mode

GL_SELECT Select Mode Functions

- glSelectBuffer(): specifies name buffer
- glInitNames(): initializes name buffer
- glPushName(id): push id on name buffer
- glPopName(): pop top of name buffer
- glLoadName(id): replace top name on buffer

id is set by application program to identify Objects

GL_SELECT: Each primitive in the **view volume** generates a *hit record* that is placed in a ***name stack*** which can be examined later

- Enter selection mode (using mouse) `glSelectBuffer();`
- Initialize name buffer `glInitNames();`
- Render scene with user-defined identifiers `glLoadName(id);`
- Reenter normal render mode

```
95    hits = glRenderMode (GL_RENDER);
```

  This operation returns number of hits

- Examine contents of **name buffer** (hit records)

  Hit records include **id** and depth information

```
96    processHits (hits, selectBuf);
```

re3 - Microsoft Visual Studio

GL_SELECT Select Mode Functions

- glSelectBuffer(): specifies name buffer
- glInitNames(): initializes name buffer
- glPushName(id): push id on name buffer
- glPopName(): pop top of name buffer
- glLoadName(id): replace top name on buffer

id is set by application program to identify Objects

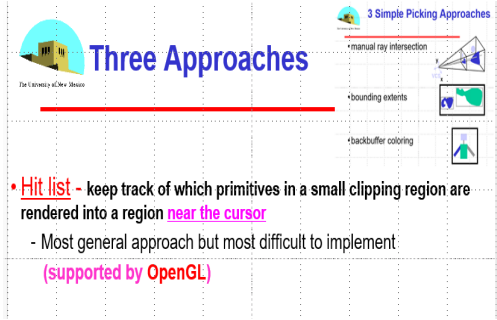
```
10  #endif
11
12
13  void init()
14  {
15      glClearColor (0.0, 0.0, 0.0, 0.0);
16  }
17
18  void drawObjects(GLenum mode)
19  {
20      if(mode == GL_SELECT)
21          glLoadName(1);
22      glColor3f(1.0, 0.0, 0.0);
23      glRectf(-0.5, -0.5, 1.0, 1.0)
24
25      if(mode == GL_SELECT)
26          glLoadName(2);
27      glColor3f(0.0, 0.0, 1.0);
28      glRectf(-1.0, -1.0, 0.5, 0.5)
29  }
30
31  void display()
32  {
33      glClear(GL_COLOR_BUFFER_BIT);
```

Ln 21 Col 23 Ch 17 INS

Ln 78 Col 22 Ch 19 INS

The University of New Mexico

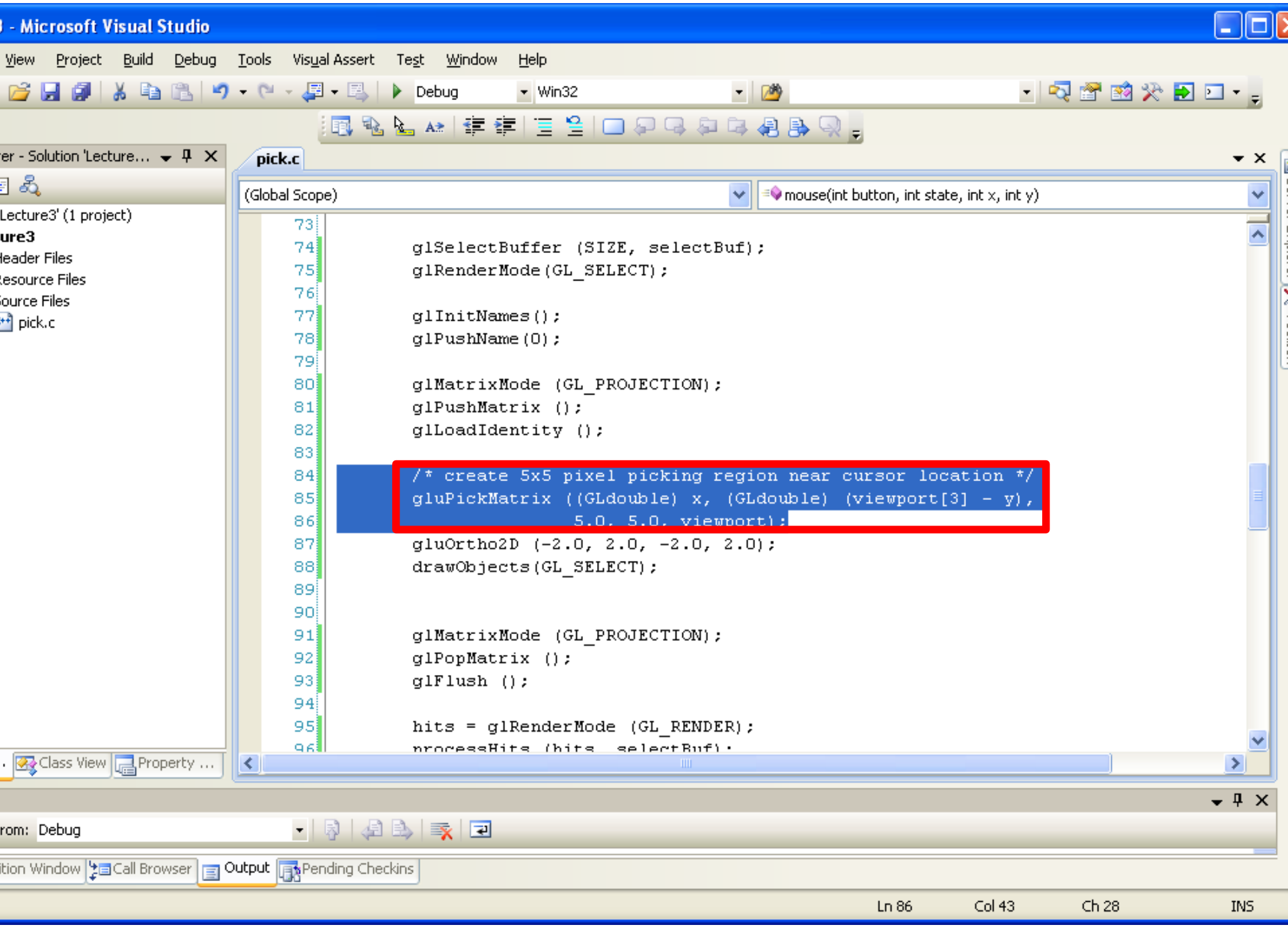
Select Mode and Picking

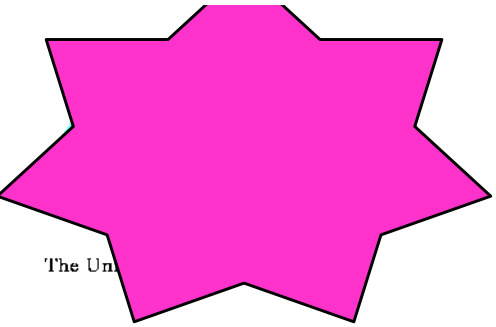


As we just described it, **select mode** won't work for picking because every primitive in the **view volume** will generate a hit

Change the viewing parameters so that only those primitives **near the cursor** are in the altered view volume

Use `gluPickMatrix`


- Microsoft Visual Studio
View Project Build Debug Tools Visual Assert Test Window Help
Debug
Win32
er - Solution 'Lecture...
Lecture3' (1 project)
ure3
Header Files
Resource Files
Source Files
pick.c
Class View
Property ...
pick.c
(Global Scope)
mouse(int button, int state, int x, int y)
73
74 glSelectBuffer (SIZE, selectBuf);
75 glRenderMode(GL_SELECT);
76
77 glInitNames();
78 glPushName(0);
79
80 glMatrixMode (GL_PROJECTION);
81 glPushMatrix ();
82 glLoadIdentity ();
83
84 /* create 5x5 pixel picking region near cursor location */
85 gluPickMatrix ((GLdouble) x, (GLdouble) (viewport[3] - y),
86 5.0, 5.0, viewport);
87 gluOrtho2D (-2.0, 2.0, -2.0, 2.0);
88 drawObjects(GL_SELECT);
89
90
91 glMatrixMode (GL_PROJECTION);
92 glPopMatrix ();
93 glFlush ();
94
95 hits = glRenderMode (GL_RENDER);
96 processHits (hits, selectBuf);
rom: Debug
ition Window
Call Browser
Output
Pending Checkins
Ln 86
Col 43
Ch 28
INS



pick.c

VH, publish

The Un

Name: _____

Total score: ☐

Class PARTICIPATION on Lecture 3.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

3. (34 points) Download the pick.c from CANVAS and add it to Project.

a. Build and run the project. Use the left button to pick red and blue squares. Watch the hits.

Self Graded - correctly

(print screen)

Using Regions of the Screen

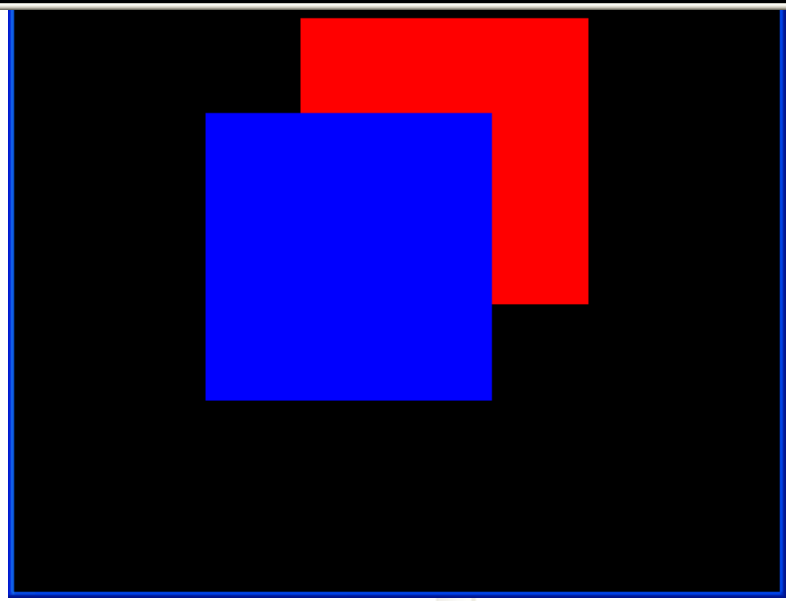
Easier to look at mouse position and determine which area of screen it is in than using **select mode picking**

GL_SELECT Select Mode Functions

- glSelectBuffer(): specifies name buffer
- glInitNames(): initializes name buffer
- glPushName(id): push id on name buffer
- glPopName(): pop top of name buffer
- glLoadName(id): replace top name on buffer

id is set by application program to identify Objects

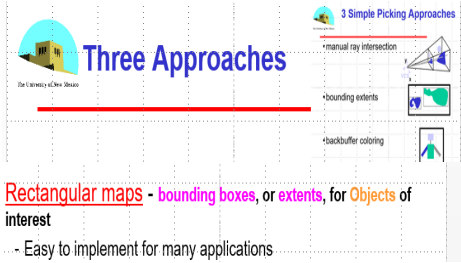
```
f:\COSC 4370 ---------- SUMMER 2010\LECTURES\CLASS PARTICIPATION ON LECTURE 3\...
blue rectangle

hits = 1
red rectangle

hits = 1
red rectangle

hits = 1
blue rectangle

hits = 1
red rectangle

hits = 1
red rectangle

hits = 1
red rectangle

hits = 1
red rectangle

hits = 0
```



```
void drawObjects(GLenum mode)
{
    if(mode == GL_SELECT) glLoadName(1);
    glColor3f(1.0, 0.0, 0.0);
    glRectf(-0.5, -0.5, 1.0, 1.0);
    if(mode == GL_SELECT) glLoadName(2);
    glColor3f(0.0, 0.0, 1.0);
    glRectf(-1.0, -1.0, 0.5, 0.5);
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    drawObjects(GL_RENDER);
    glFlush();
```

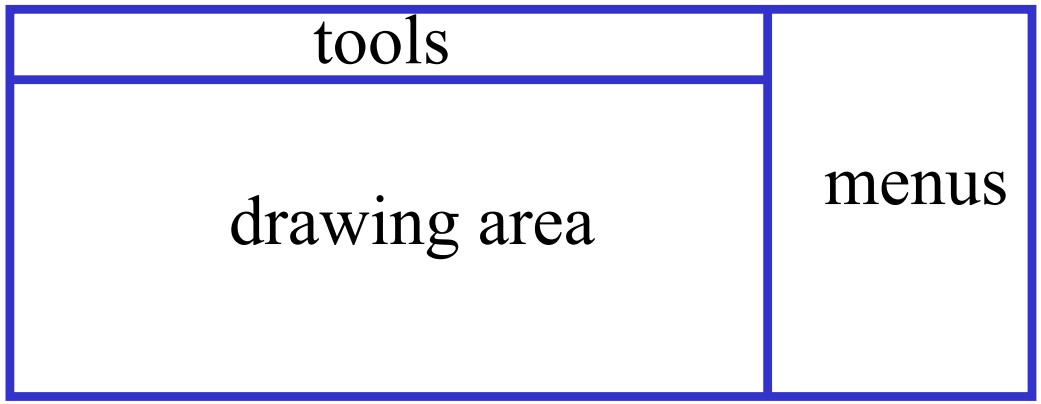
pick.c 3

2. Using Regions of the Screen



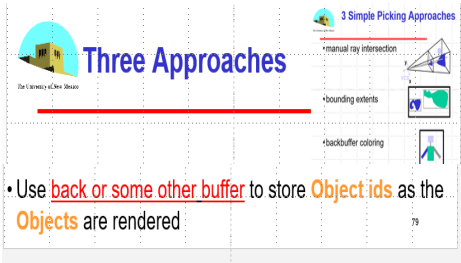
Many applications use a simple rectangular arrangement of the screen

Example: paint/CAD program



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

3. Using another buffer and colors for picking



For a small number of **Objects**, we can assign a unique color (often in color index mode) to each **Object**

We then **render the scene** to a color buffer other than the **front buffer** so the results of the rendering are not visible

We then get the mouse position and use **`glReadPixels()`** to read the color in the buffer we just wrote at the position of the mouse

The returned color gives the **id** of the **Object**

Objectives

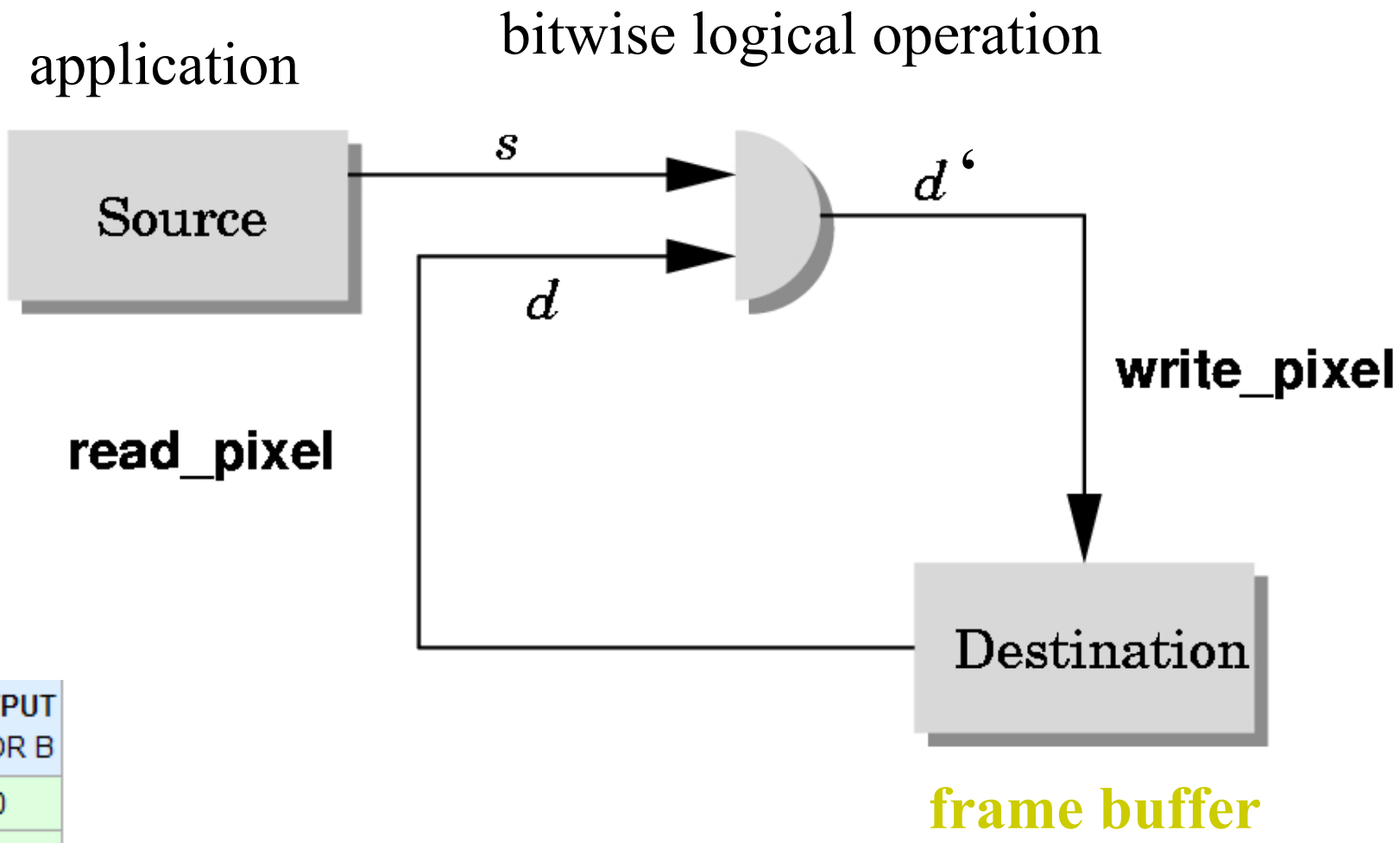
- Learn to build more sophisticated interactive programs using

  Picking
  - Select objects from the display
  - Three methods

  Rubberbanding
  - Interactive drawing of lines and rectangles

  Display Lists
  - Retained mode graphics

Writing Modes



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

If applied twice, it returns to the original state!

XOR write

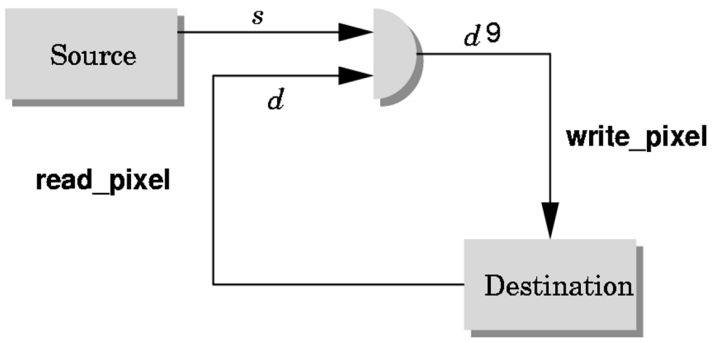
- Usual (default) mode: source replaces destination (d' = s)

  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 ⊕ s)

  x ⊕ y ⊕ x =y

  Hence, if we use XOR mode to write a line, we can draw it a second time **and line is erased!**



Rubberbanding

- 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

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**

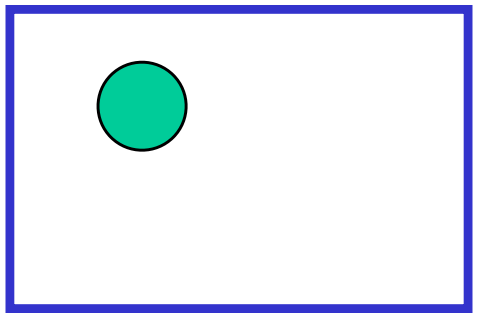
Each time mouse is moved, **redraw line which erases it** and then draw line from **fixed first position** to the **new second position**

At end, switch back to **normal drawing mode** and draw line

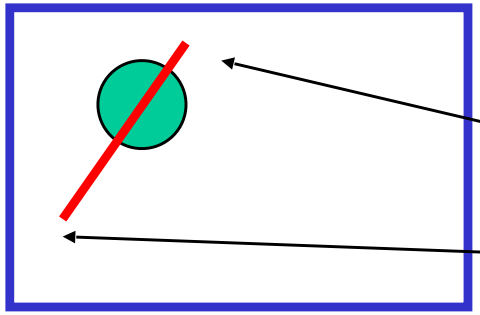
Works for other **Objects**: rectangles, circles

Rubberband Lines

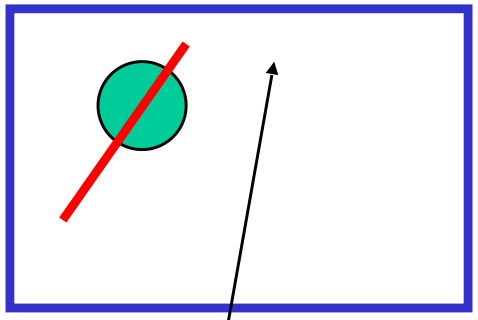
- Learn to build more sophisticated interactive programs using
  - Picking
    - Select objects from the display
    - Three methods
  - Rubberbanding
    - Interactive drawing of lines and rectangles
  - Display Lists
    - Retained mode graphics



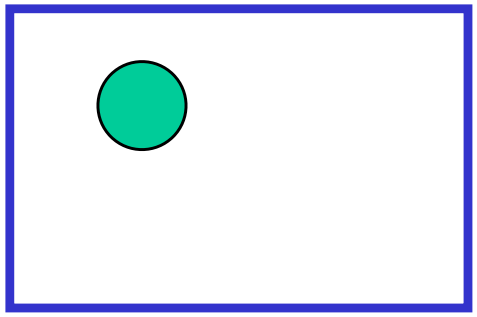
initial display



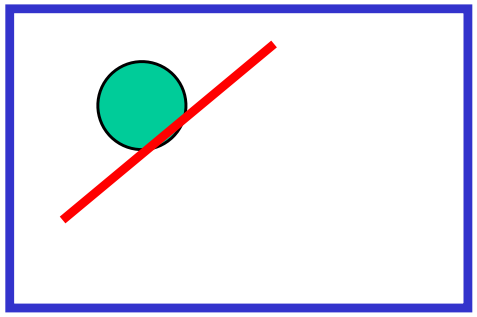
draw line with mouse
in XOR mode



mouse moved to
new position



original line redrawn
with XOR



new line drawn
with XOR

XOR in OpenGL

- 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

There are **16 possible logical operations** between **2 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)

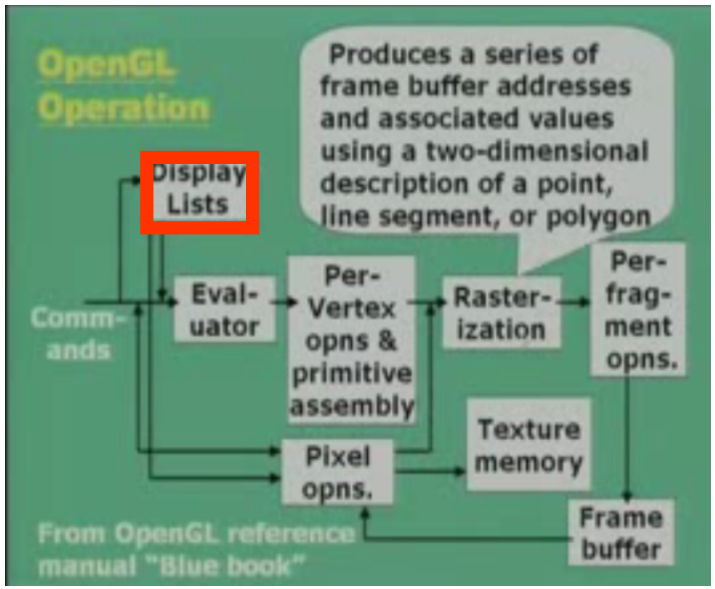
Objectives

- 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 (Scene Graph)
  - **Retained mode graphics**



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

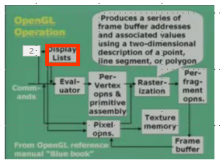
- Alternative is define **Objects** and keep them in some form that can be **redisplayed** easily

  Retained mode graphics

  Accomplished in OpenGL via ***display lists***

Display Lists

- 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 (Scene Graph)
    - **Retained mode graphics**



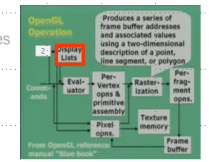
- Conceptually similar to a **graphics file**
  - Must define (name, create)
  - Add contents
  - Close

- In client-server environment, **Display List is placed on server**
  - Can be **redisplayed without sending primitives** over network each time

The University of New Mexico

Display List Functions

- 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 (Scene Graph)
    - Retained mode graphics

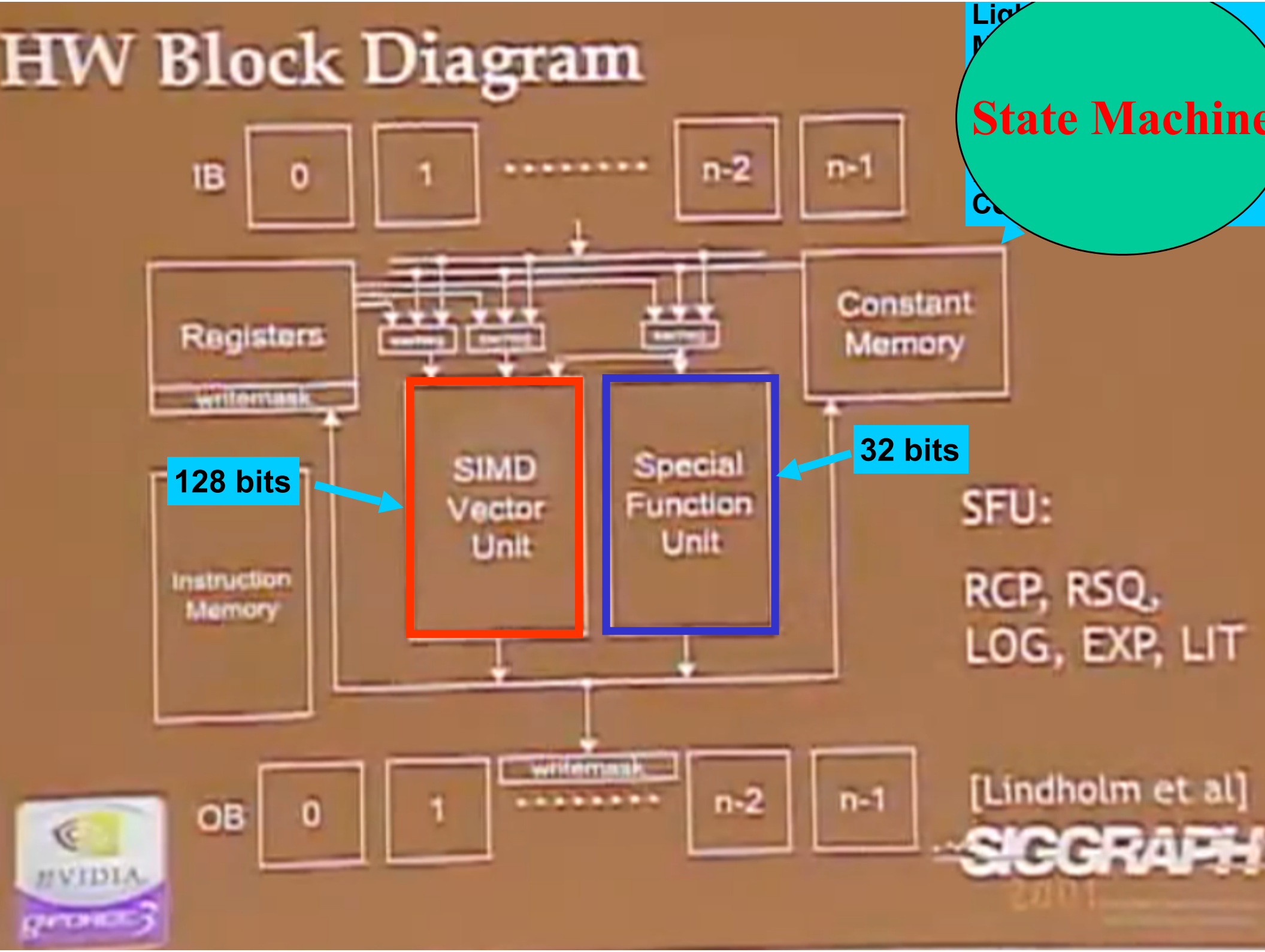


- Creating a Display List

```
GLuint id;

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

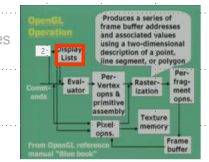
- Call a Created List

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


HW Block Diagram
State Machine
IB
0
1
n-2
n-1
Registers
writemask
Constant Memory
128 bits
SIMD Vector Unit
Special Function Unit
32 bits
Instruction Memory
SFU:
RCP, RSQ, LOG, EXP, LIT
OB
0
1
writemask
n-2
n-1
[Lindholm et al]
SIGGRAPH
NVIDIA

Display Lists and State

- 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 (Scene Graph)
    - Retained mode graphics



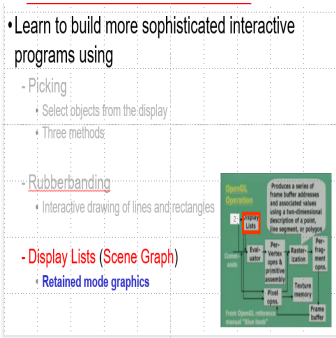
Most **OpenGL** functions can be put in **Display Lists**

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**

The University of New Mexico

Hierarchy and Display Lists



- Consider model of a car
  - Create Display List for chassis
  - Create Display List for wheel

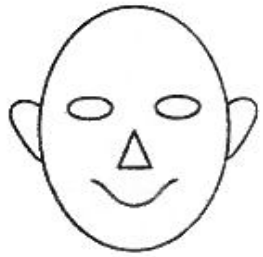
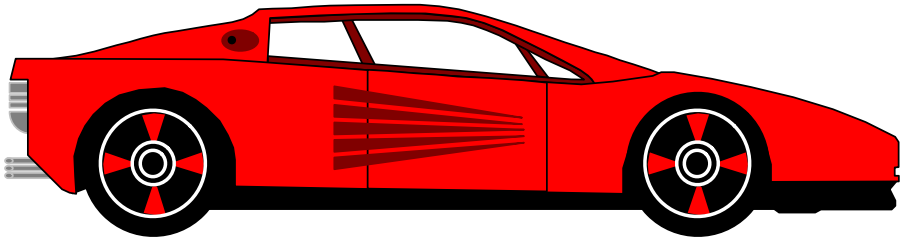


FIGURE 3.16 Simple animated face

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



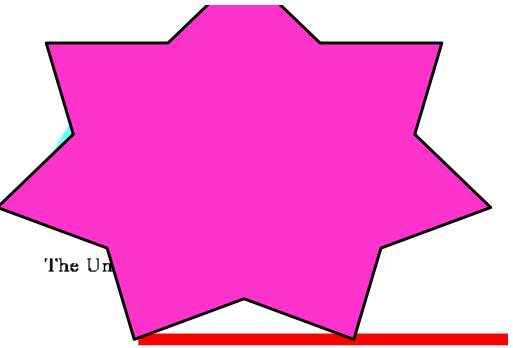
SUMMARY and NOTES

The **interactive** aspects make the field of computer graphics **exciting and fun**. Although our **API**, **OpenGL**, is independent of any operating or window system, we recognize that any program must have at least minimal interaction with the rest of the computer system. We handled simple interactions by using a simple toolkit, **GLUT**, whose **API** provides the necessary additional functionality, **without being dependent on a particular operating or window system**.

From the **application programmers** perspective, various characteristics of interactive graphics are shared by most systems. We see the graphics part of the system as a server, consisting of a raster display, a keyboard, and a pointing device. In almost all workstations, we have to work within a multiprocessing, windowed environment. Most likely, many processes are executing concurrently with the execution of your graphics program. However, the window system **allows us to write programs for a specific window that act as though that window were the display device of a single-user system**.

The use of **logical devices within the application program frees the programmer from worrying about the details of particular hardware**. Within the environment that we have described, **event-mode input is the norm**. Although the other forms are available **request mode** is the normal method used for keyboard input event-mode input gives us far more flexibility in the design of **interactive programs**.

Interactive computer graphics is a powerful tool **with unlimited applications**. At this point, you should be **able to write fairly sophisticated interactive programs**.



You will be prompted when to Upload completed document to CANVAS as **score.doc** (example 100.doc).

Warning:

TA, at random, will inspect the Uploaded document.

If you score is not honestly entered you will get a zero.

Please rename document to score.doc (example 100.doc)
Warning: if you score is not honestly honest you will get a zero.

Name: _____

Total score: ☐

Class PARTICIPATION on Lecture 3.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

VH, it closes at 7:00 PM

NEXT.

The University of New Mexico

09.11.2023 (M 5:30 to 7) (6)		Math Review 2

At 6:45 PM.

End Class 5

VH, Download Attendance Report
Rename it:
9.06.2023 Attendance Report FINAL

VH, upload Lecture 3 to CANVAS.