RV College of Engineering®

Go, change the world

COMPUTER GRAPHICS & VIRTUAL REALITY (18CS72)

UNIT-II

Introduction to Computer Graphics and Virtual Reality

Input and Output Devices:

Input and Output Devices:

Input and Interaction: Input Devices. Physical Input Devices, Logical Devices. Measure and trigger. Input Modes. Event-Driven Input: Using the pointing device, Window events, and Keyboard events. Menus.

VR related Input Devices: Trackers, Navigation, and Gesture Interfaces;

VR related Output Devices: sound displays & haptic feedback

Introduction to Computer Graphics and Virtual Reality

Input and Output Devices : (7 Hrs)

- ✓ Input and Interaction: Input Devices. Physical Input Devices, Logical Devices.
- ✓ Measure and trigger. Input Modes.
- ✓ Event-Driven Input: Using the pointing device, Window events, and Keyboard events.
- ✓ Menus.

VR related Input Devices: Trackers, Navigation, and Gesture Interfaces;

VR related Output Devices: sound displays & haptic feedback

Input & Interaction

Till now – no interaction

1. Introduction of devices for interaction
2. how devices "appear" in your program
3. client-server network & client-server graphics
4. development of a painting program

Different approach will be taken – we will use API, but OpenGL does not support it directly – due to portability of the renderer, interaction with OS etc.

Input Devices

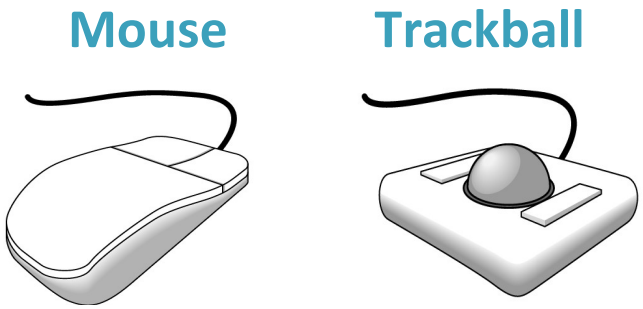
Two possible ways to see input devices:

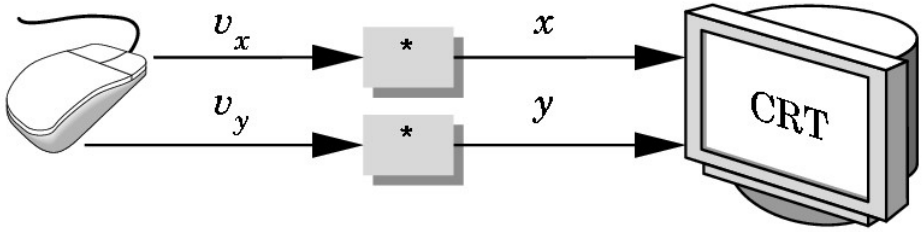
- as a **physical device** – keyboard, mouse, trackball, etc.
- as a ***logical device*** – from a programmer perspective – with specified functionality, in graphics more complex
- the ***separation of physical and logical levels*** enable us to make programs more flexible, independent from the actual physical device

Physical Input Devices

Physical input devices:

- ***pointing device*** – allows to indicate position & send signals/interrupts to the computer – relative/absolute positioning
- keyboard device – almost physical keyboard – returns character codes to a program

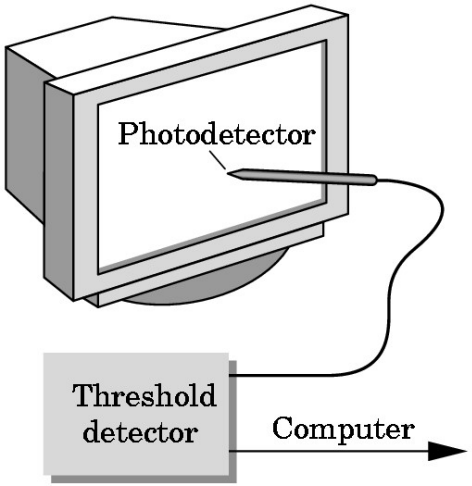




Physical Input Devices

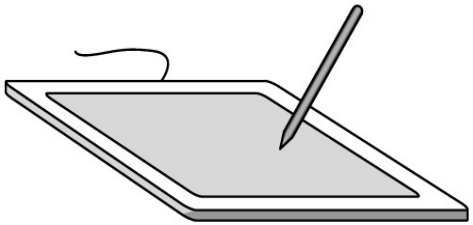
Absolute positioning:

- data tablets
- light pen
- joystick – variable-sensitivity device & haptic device
- spaceball – up-down, left-right, front-back & 3 independent twists

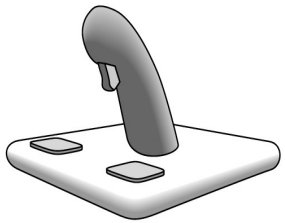


Light pen

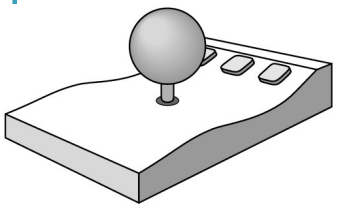
Data Tablet



Joystick



Spaceball



Logical Input Devices

- Some APIs (PHIGS, GKS, Direct xx) supports ***6 classes of logical input devices*** – OpenGL does not take this approach
- **String** – logical device providing ASCII strings – keyboard
- **Locator** – provides a position in world coordinates – usually implemented via pointing device – mouse, trackball. OpenGL provides similar but *conversion from screen coordinates to world coordinates must be made by a user*
- **Pick** – returns identifier of an object – in OpenGL process called *selection* can be used to accomplish picking

Logical Input Devices

- **Choice** – allows the user to select one of a discrete number of options – in OpenGL various widgets provided by the window system can be used; widget is a graphical interactive device provided by window system or a toolkit (menus, scroll bars, graphical buttons menu with n selections etc.)
- **Valuators/Dial** – provides analog input to the user program – slidebars, radio buttons etc.
- **Stroke** – device returns an array of locations – different implementations – usually: mouse button down, transfer data to an array with different positions, release button – ends the transfer

Input Devices & Modes

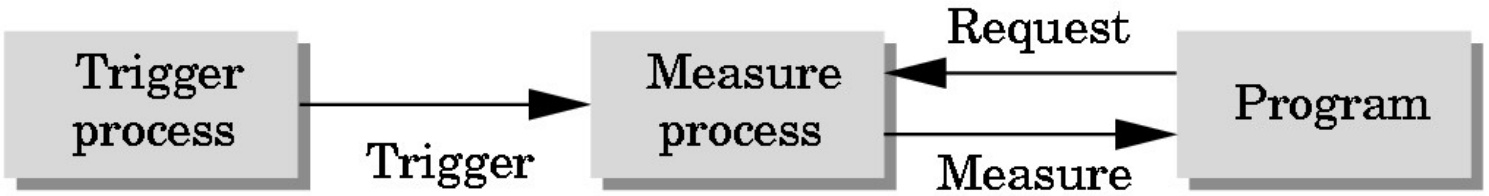
Two entities:

- a ***measure process*** – is what the device returns to the user program (string from a keyboard)
- a ***device trigger*** – is a physical input on the device which user can signal the computer (return – end of the process)

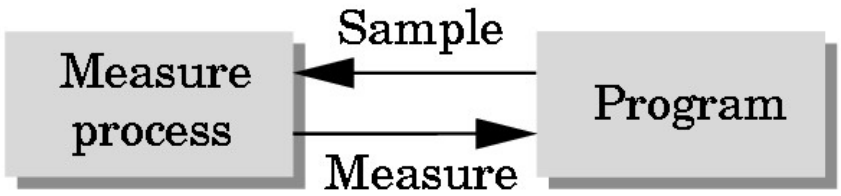
Measure of a device in ***3 distinct Modes***:

- **request** – value is returned on a request
- **sample mode** – actual value is given (no buffering)
  - request_locator (device_id, &measure); /* usual form */
  - sample_locator (device_id, &measure);

Input Devices & Modes

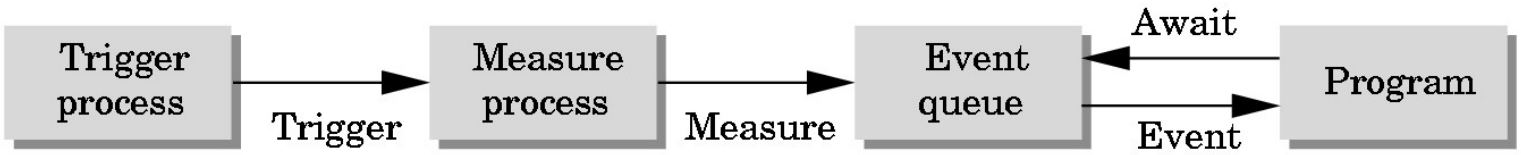


Request versus Sample modes



Generally sample & request modes are not sufficient for Human-Computer-Interface (HCI)

Input Devices & Modes



Event mode

- working in environment with multiple input devices – each has its own trigger and running measure process
- each time when the device is triggered – the event is generated, the measure with the process identifier is placed in an *event queue*.
- The program can examine the front event in the queue and decides what to do – this is used for GKS, PHIGS etc.

ISO standards for the Application Programmer Interface(API)

Programmer's Hierarchical Interactive Graphics System **(PHIGS)**

Graphical Kernel System (**GKS**)

Event-Driven Input

Using the Pointing Device

Callback

another approach – association of a *callback* function with a specific type of event – mostly used in windowing system and client/server environments

```
int main (...) { ....glutMouseFunc(mouse); glutDisplayFunc(...);..}
.....
void mouse_callback_func(int button, int state, int x, int y)
{ if button ==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
      drawSquare(x,y); /*user's function*/
if button ==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
        exit ();
} /* window reshape principle */
```

)

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)

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

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

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

Programming Event Driven Input

Pointing device: - 2 type of events

- *move event* – mouse moved with one of the buttons pressed.
- *passive move event* – mouse moved without pressing a button
- *mouse event* – occurs when one of the button pressed, or button released (some systems counts the pushing & releasing of a button as only a single event)

```
glutMouseFunc(mouse_callback_func)  /* registration */


void mouse_callback_func(int button, int state, int x, int y);
{     if (button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
exit ( );
} /* the right button pressed will be omitted
no action will be taken as no corresponding action is specified */
```

Programming Event Driven Input

```
int main (int argc, char **argv);
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutCreateWindow("square");     /* create a window with name square */
        myinit( );
        glutReshapeFunc(myReshape); /* generated if window size changed*/
        glutMouseFunc(mouse);
                                /* activated if status or position of a mouse changed */
        glutDisplayFunc(display);
                /* GLUT call back requires it strictly – in case of no action */
                /* void display ( ) { }  must be specified – empty function */
        glutMainLoop( );
}
```

Programming Event Driven Input

Keyboards events:

glutKeyBoardFunc – callback for events generated by pressing a key

glutKeyBoardUpFunc - callback for events generated by releasing a key

```
    glutKeyBoardFunc ( keyboard); /* registration */

void keyboard (unsigned char key, int x, int y);
{   if (key == 'q' || key == 'Q') exit ( ); /* exits the program */
}
```

glutGetModifiers – function to define actions using meta keys, such as Control and Alt Keys

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

Window Management

- GLUT supports multiple windows and subwindows of a given window

Multiple Window management:

- To Open a second top-level window
- id = glutCreateWindow("second window"); /* int id*/
- glutSetWindow (id) ; /* sets the window into which object will be rendered */

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

Animation

- For animation – double buffering

  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);

- Buffers:
  - front – content visible on the display
  - back – where the rendering is made

- to the display function must be added
  - **glutSwapBuffers ( );**

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
}
```

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 for using Menus:**
  - Define entries for the menu. Define action for each menu item
    - Action carried out if entry selected
  - Attach menu to a mouse button (link menu to a mouse button)
  - Register a callback function for each menu

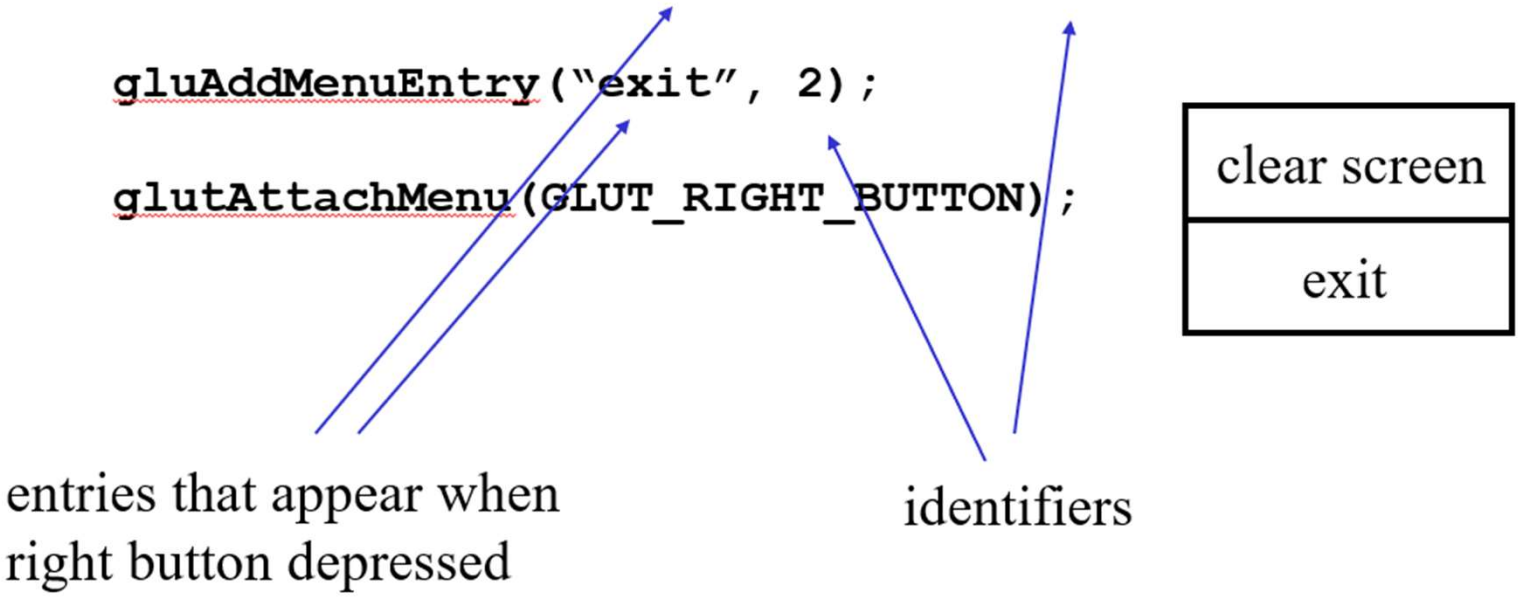
Menus

- In **main.c**

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

gluAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



Menus

- 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

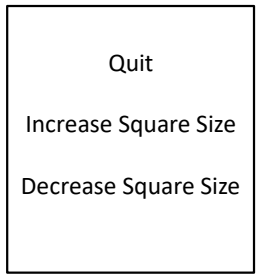
glutAddSubMenu(char *submenu_name, submenu id)

entry in parent menu

Menus

Pop-up menus

```
glutCreateMenu(demo_menu);          //
glutAddmenuEntry("quit",1);
glutAddmenuEntry("increase square size",2);
glutAddmenuEntry("decrease square size",3);
glutAttachmenu(GLUT_RIGHT_BUTTON);
```

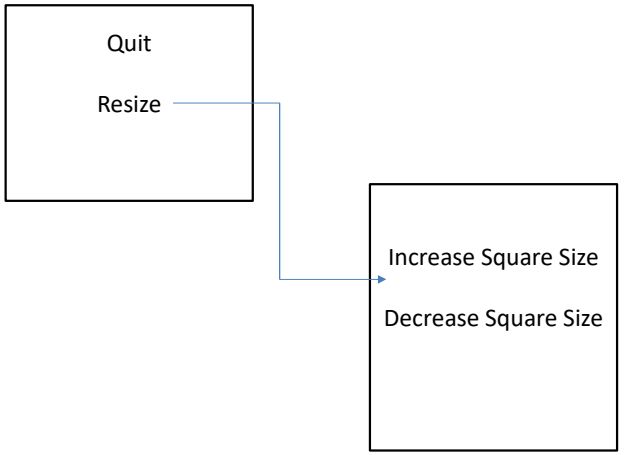



```
void demo_menu(int id)              // Call back function
{   if (id ==1) exit (0);
    else if (id ==2) size = size * 2;
    else if (id ==3) size = size / 2;
    glutPostRedisplay( );
    /* glutDisplayFunc is called-redisplay without menu */
}
```

Menus

Hierarchical Menus

```
sub_menu = glutCreateMenu(size_menu);
glutAddmenuEntry("increase square size",2);
glutAddmenuEntry("decrease square size",3);
glutCreateMenu(top_menu);
glutAddMenuEntry("Quit",1);
glutAddSubMenu("Resize", sub_menu);
glutAttachmenu(GLUT_RIGHT_BUTTON);
void size_menu(int id)            // Call back function
{     }
void top_menu( int id)
{ }
```



Control Functions

- Minimal interaction between the program and the particular OS must be used (X-Windows - UNIX, Windows)
- GLUT library provides minimal & simple interface between API and particular OS (full functionality for interaction between a program and OS is not a part of this course)
- Programs using GLUT interface should run under multiple window system.

Interaction with the Window System

- Window or screen window – rectangular area of a display
- Window system – refers to X-Windows or MS Windows
- The origin – not always lower-left corner – OpenGL style (some have orientation: top to bottom, left to right as TV sets or position information returned from input devices like mouse)

- glutInit(int *argcp, char *argv) – enables to pass command-line arguments as in the standard C *main* function
- glutCreateWindow(char *title) – creates a window with the name specified in the string title with default values (size, position etc.)

Interaction with the Window System

- BEFORE creating the window GLUT functions can be used to specify properties:

```
glutDisplayMode(GLUT_RGB|GLUT_DEPTH|GLUT_DOUBLE);
/* parameters or-ed and stored in the argument to glutInitDisplayMode */
glutWindowSize(480, 640);
glutWindowPosition(0 , 0);
```

specifies the windows 480 x 640 in the top-left corner of the display

GLUT_RGB x GLUT_INDEX – type of color system to be used

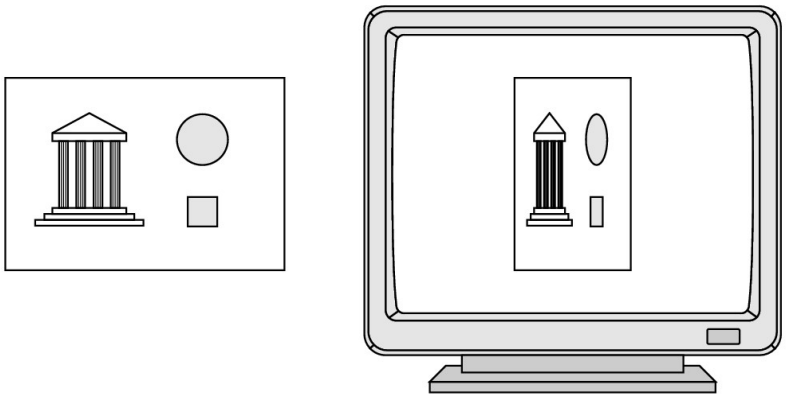
GLUT_DEPTH – a depth buffer for hidden-line removal

GLUT_DOUBLE x GLUT_SINGLE – double x single buffering

- Implicit options: RGB color, no hidden-line removal, single buffer

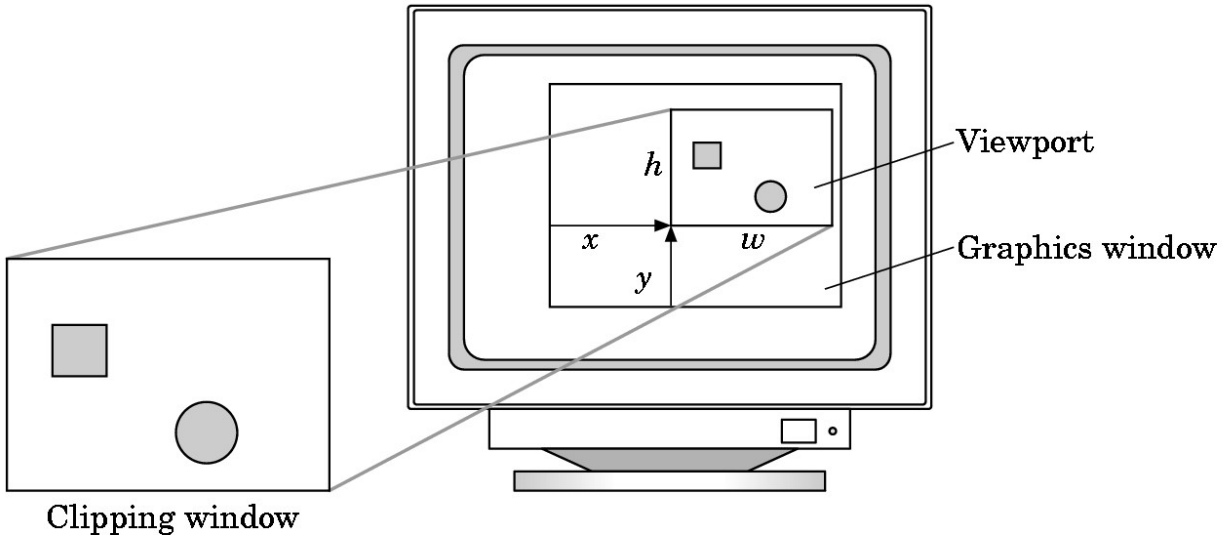
Aspect Ratio and Viewports

- Aspect ratio – the ratio of the rectangle's width and height
- If different in glOrtho and glutInitWindowSize – undesirable side effects
- Caused by the independence of object, viewing parameters and workstation window specifications
- Concept of a VIEWPORT



Aspect Ratio and Viewports

void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)



The viewport is part of the state – when changed between rendering objects or redisplay – different window-viewport transformations used to make the scene.

Traditional Input Device (1/4)

- Commonly used today
- Mouse-like devices
  - mouse
  - wheel mouse
  - trackball
- Keyboards

Traditional Input Device

- Joysticks
  - game pads
  - flightsticks
  - Touchscreens
- Microphones
  - wireless vs. wired
  - headset

3D Input Device

- Gloves
  - attach electromagnetic tracker to the hand
- Pinch gloves
  - contact between digits is a "pinch" gesture
  - in CAVE, extended Fakespace PINCH™ gloves with extra contacts

3D Input Device

- Gloves
  - attach electromagnetic tracker to the hand
- Pinch gloves
  - contact between digits is a "pinch" gesture
  - in CAVE, extended Fakespace PINCH™ gloves with extra contacts

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

simple.c revisited

- In this version, we shall see the same output but we have defined all the relevant state values through function calls using the default values
- In particular, we set
  - Colors
  - Viewing conditions
  - Window properties

86

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 gl.h

define window properties

display callback

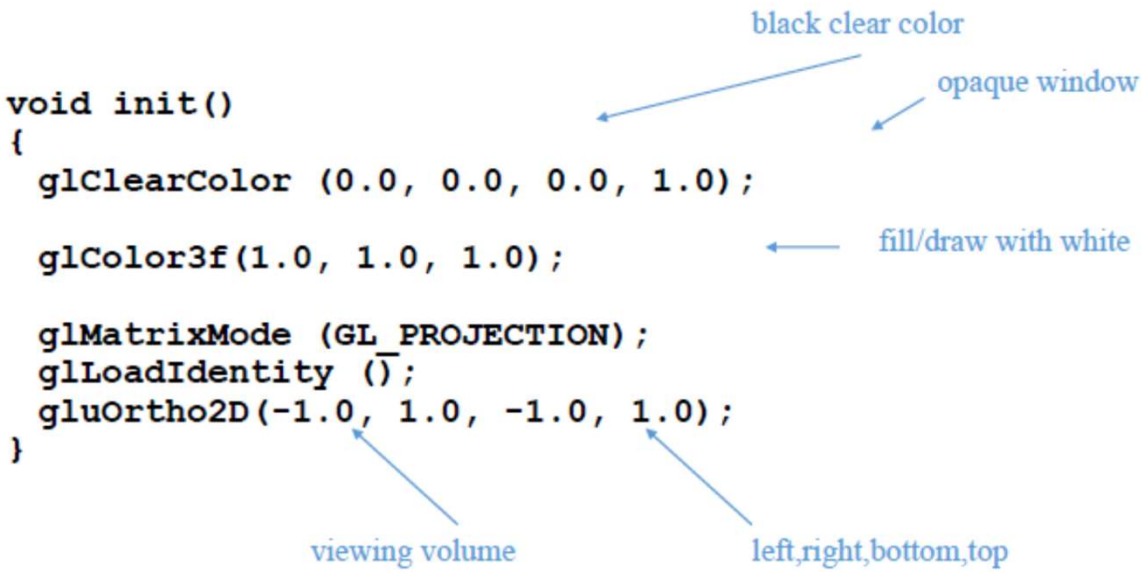
set OpenGL state

enter event loop

GLUT functions

- **`glutInit`** allows application to get command line arguments and initializes system
- **`gluInitDisplayMode`** requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically ORed together
- **`glutWindowSize`** in pixels
- **`glutWindowPosition`** from top-left corner of display
- **`glutCreateWindow`** create window with title "simple"
- **`glutDisplayFunc`** display callback
- **`glutMainLoop`** enter infinite event loop

init.c




```
void init()
{
  glClearColor (0.0, 0.0, 0.0, 1.0);

  glColor3f(1.0, 1.0, 1.0);

  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}
```

Control Functions

Control functions enable the programmer to communicate with the window system, initialize the program etc.

1.glutInit(int*argcp, char **argv)

2.glutCreateWindow(char *title)

3.glutInitDisplayMode()

4.glutInitWindowSize()

5.GlutInitWindowposition()

6.glutDisplayFunc()

7.glutMainLoop()

Control Functions

Control functions enable the programmer to communicate with the window system, initialize the program etc.

1.glutInit(int*argcp, char **argv)

2.glutCreateWindow(char *title)

3.glutInitDisplayMode()

4.glutInitWindowSize()

5.GlutInitWindowposition()

6.glutDisplayFunc()

7.glutMainLoop()

Control Functions

1.glutInit(int*argcp, char **argv)

•It initiates an interaction between the windowing system and OpenGL. It is used before opening a window in the program.

•The two arguments allow the user to pass command-line arguments, as in the standard C main function, and are usually the same as in main.

2.glutCreateWindow(char *title)

•It opens an OpenGL window.

•The title string provides title at the top to the window displayed.

Control Functions

3.glutInitDisplayMode()

Used to initialize the display window created, by specifying colormodel ,buffering used etc.

4.glutInitWindowSize()

Used to specify the size of the window to be created.

5.glutInitWindowPosition()

Used to specify the position of the created window on
The monitor with respect to the top left corner.

Control Functions

6.glutDisplayFunc()

•Used to specify the display callback.
•This function executes when the window is created for the firsttime.
•It is also called when the window is moved from one location on the screen to another.

7.glutMainLoop()

•Causes the program to begin an event processing loop.
•If there are noevents to process, then the program would enter the wait state with the output on the screen.
•It is similar to getch() in C program