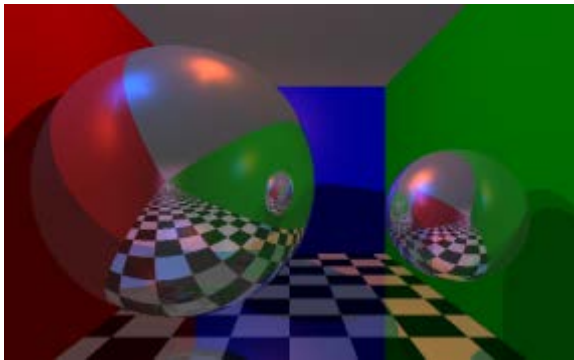
CSI 4105 Computer Graphics

Spring 2017

Lecture 3:
Programming with OpenGL (Part II)

Seon Joo Kim
Yonsei University



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

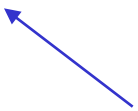
- Devices can be described either by
  - Physical properties
    - Mouse, Keyboard, Trackball, etc.
  - Logical Properties
    - What is returned to program via API
      - **A position**
      - **An object identifier**
- Modes
  - How and when input is obtained
    - Request or event

Event Types in GLUT

- **Window:** resize, expose, iconify (cause “display”)
- **Mouse:** click one or more buttons
  - (both **up** and **down** are an event – i.e. one click technically creates 2 events)
- **Motion:** move mouse
- **Keyboard:** press or release a key
- **Idle:** non-event
  - Define what should be done if no other event is in queue
  - We saw this in the last lecture

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, Mac)
  - **`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

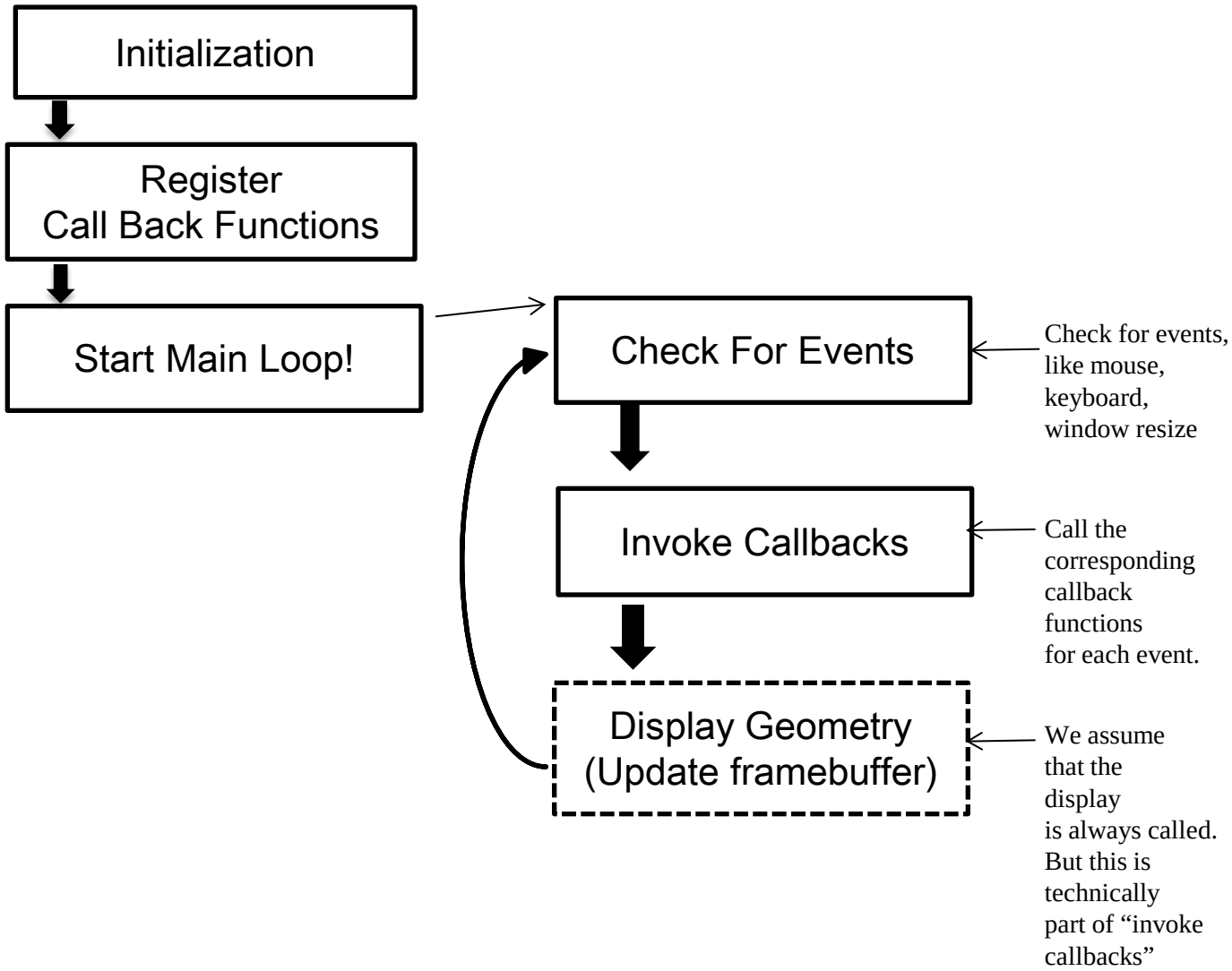
- In each pass through the event loop, GLUT
  - looks at the events in the queue
  - for each event in the queue, GLUT executes the appropriate callback function if one is defined
  - if no callback is defined for the event, the event is ignored

Event Loop (recall from last lecture)

Initialize user variables, load geometry, set window size, Open window with attributes (e.g. RGB, double-buffer, z-buffer, etc. .)

Register user "callback" function, e.g. Reshape, Keyboard, Mouse, Idle . . .

`glutMainLoop()`
Begins infinite loop for Glut



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
- This often creates a strange and unpleasant effect

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();
}
```

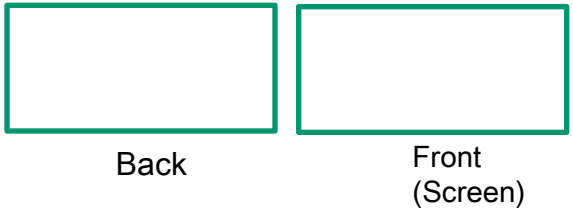
Do this before you create your window.

Call swap buffer.
What if you forget this? No picture on the screen!

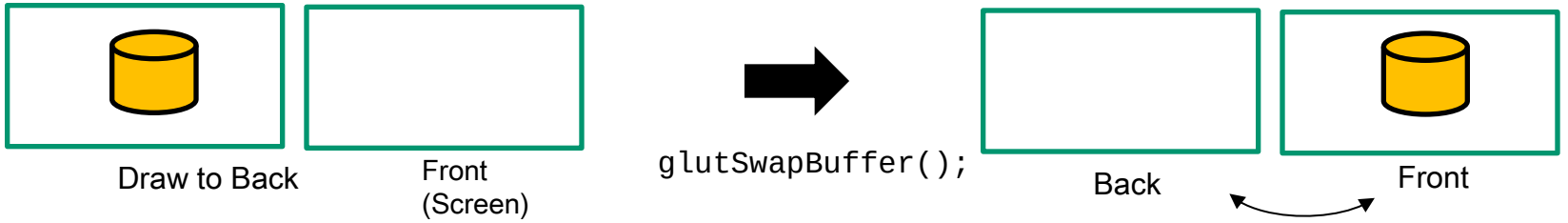
*The truth is we **always** use double-buffering, single buffer graphics apps are rare!

Double Buffering (use two buffers)

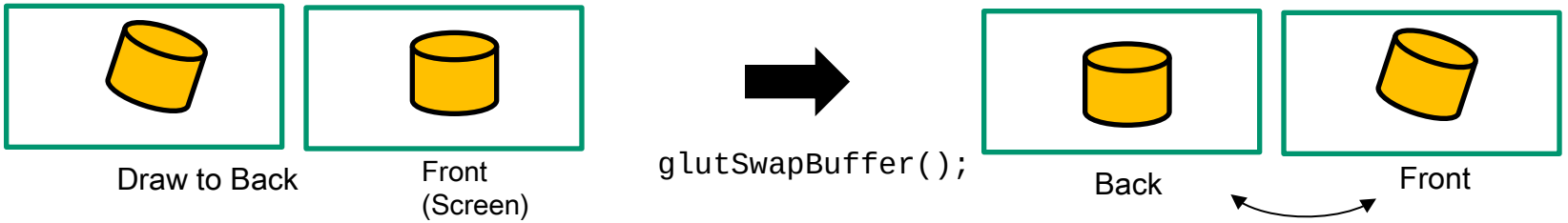
1- App opens, both buffers are empty



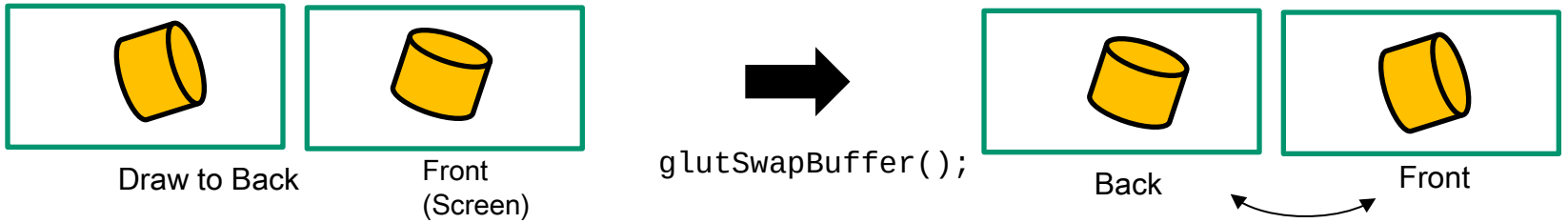
2- Draw some awesome graphics in the BACK, then swap buffers



3- Draw some more awesome graphics in the BACK, then swap buffers



4- Draw even more awesome graphics in the BACK, then swap buffers [REPEAT]



Using the Idle Callback (see previous lecture)

- 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 the glutTimers

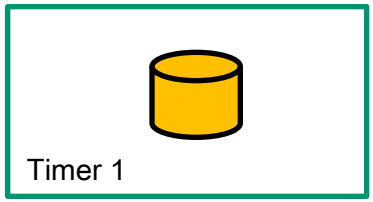
- One problem with idle is it is hard to control the “speed” of redraw.
- Another option is using the GlutTimer Func

```
void Timer(int param)
{
      glutPostRedisplay();
      glutTimerFunc(15, Timer, 0 ); // call in 15ms
}
...
int main()
{     ...
      glutTimerFunc(0,Timer,0);     // call in 0ms
      glutMainLoop();
}
```

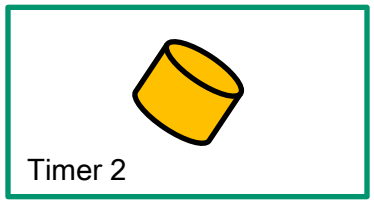
This is the value of “param” in the function. It can be used by the Timer func if desired (e.g. as an ID or something. Here I just use 0.)

See: BoxDropTimer.c

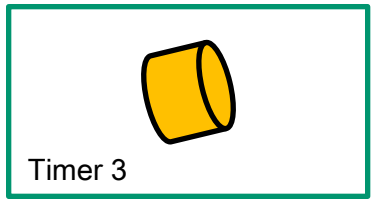
glutTimerFunc



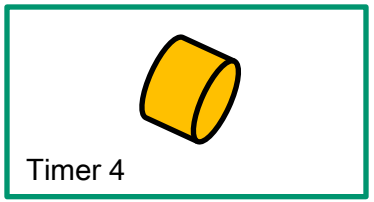

```
glutTimerFunc(0,Timer,0);
```



Post ReDisplay Event
FunctionTimer sets a new Timer for 15ms



Post ReDisplay Event
FunctionTimer sets a new Timer for 15ms



Post ReDisplay Event
FunctionTimer sets a new Timer for 15ms

NOTE: glutTimerFunc **is not** a call back function.

Instead it is a “one off” event.

Therefore, if you want it to work over an over, you need to set it again. Most often we just set the next `glutTimerFunc(..)` event in the Timer function itself.

The Need for Global Variables

- The form of all GLUT callbacks is fixed
  - **`void mydisplay()`**
  - **`void mymouse(GLint button, GLint state, GLint x, GLint y)`**
- We are forced to use globals to pass information to callbacks – its OK, this is graphics, can bend the rules

```
float t; /*global */

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

Working with Callbacks

Mouse events

- Glut has three events for the mouse

1) `glutMouseFunc(..)`
   - Called when a mouse button is clicked down and up
     - Records (x,y) location, which button, and if it is down or up click
   - So, a single click will be two events!: 1) down and 2) up

2) `glutMotionFunc(..)`
   - Called when the mouse moves and a button is pressed down
   - Only reports (x,y) position, so we have to record which button it is using the call back above

3) `glutPassiveMotion(..)`
   - Reports anytime the mouse move and no button is pressed

See MouseMotion.c

1) glutMouseFunc(..)

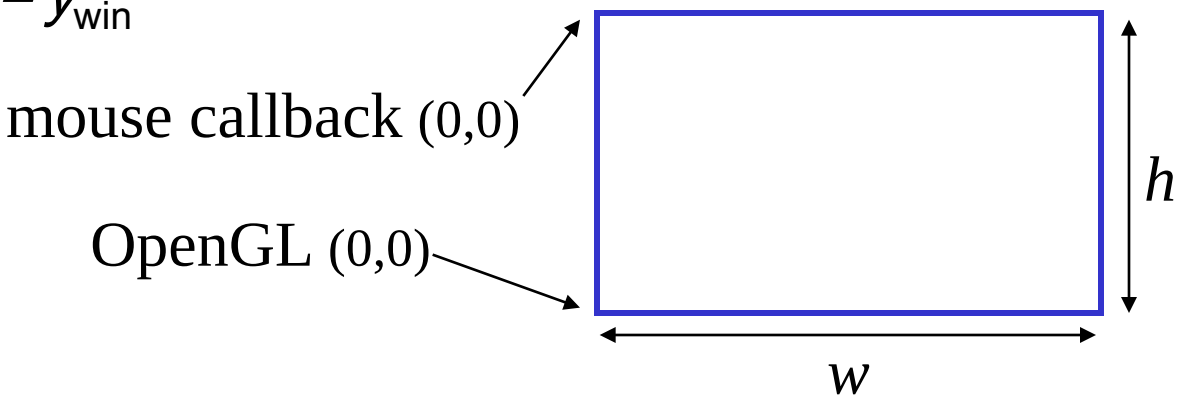
- `glutMouseFunc(mymouse);`
- `void mymouse(GLint button, GLint state, GLint x, GLint y)`

  Inputs

  - which button caused the event
    - **GLUT_LEFT_BUTTON**, **GLUT_MIDDLE_BUTTON** or **GLUT_RIGHT_BUTTON**
  - state of that button
    - **GLUT_UP** or **GLUT_DOWN**
  - mouse cursor position in window
    - top-left corner is (0,0),
      top-right corner is (winWidth−1,0),
      bottom-left corner is (0, winHeight−1),
      bottom-right corner is (winWidth−1, winHeight−1)

Mouse Position
(Pay ATTENTION, this is a little strange)

- To window system (and mouse & motion callback), position in window is measured in pixels with the origin at the top-left corner
  - Consequence of refresh done from top to bottom
- But to OpenGL, position in window is measured in pixels with the origin at the bottom-left corner
  - Must invert *y* coordinate returned by callback by height of window
  - $y_{opengl} = h - 1 - y_{win}$



Tracking 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
    - So, use globals in reshape to track window's width and height
  - Can also use query functions
    - **`glGetIntv`**
    - **`glGetFloatv`**

    to obtain any value that is part of the state

Tracking Window Size Example

```
int winH, winW;     // global

. . .
void reshape(int w, int h)
{
       winW = w; winH = h;
}

. . .

void mymouse(GLint button, GLint state,
             GLint x, GLint y)
{
       . . .
       y= winH – y;  // invert mouse
       . . .
}
```

Globals vars to track window.

Reshape is the callback to track the window change in. It will be called when the window is first opened.

In all the mouse call backs, use the global variable to adjust the mouse coordinates.

See windowTracking.c

2) `glutMotionFunc(..)`

- Function is called as long as there is motion and a button is pressed
- Register call back :**`glutMotionFunc(mouseMotion);`**
- **`void mouseMotion(int x, int y)`**
  - x and y are the position of the mouse
  - We don’t know the button
  - Interesting thing – this will keep reporting a value even if the mouse is off the glut window (values will reflect correct position)

3) `glutPassiveMotionFunc(..)`

- Function is called as long as there is motion and no button pressed
- Register call back :**`glutMotionPassiveFunc(mousePassiveMotion);`**
- **`void mousePassiveMotion(int x, int y)`**
  - x and y are the position of the mouse
  - Will only report a value when the mouse is on the glut Window

Keyboard events

- **`glutKeyboardFunc(mykey)`**
- **`void mykey(unsigned char key,`**
  `int x, int y)`

  Returns

  - ASCII code of key depressed and
  - mouse location

```
void mykey(unsigned char key, int x, int y)
{
    if (key == 'Q' | key == 'q')
        exit(0);
}
```

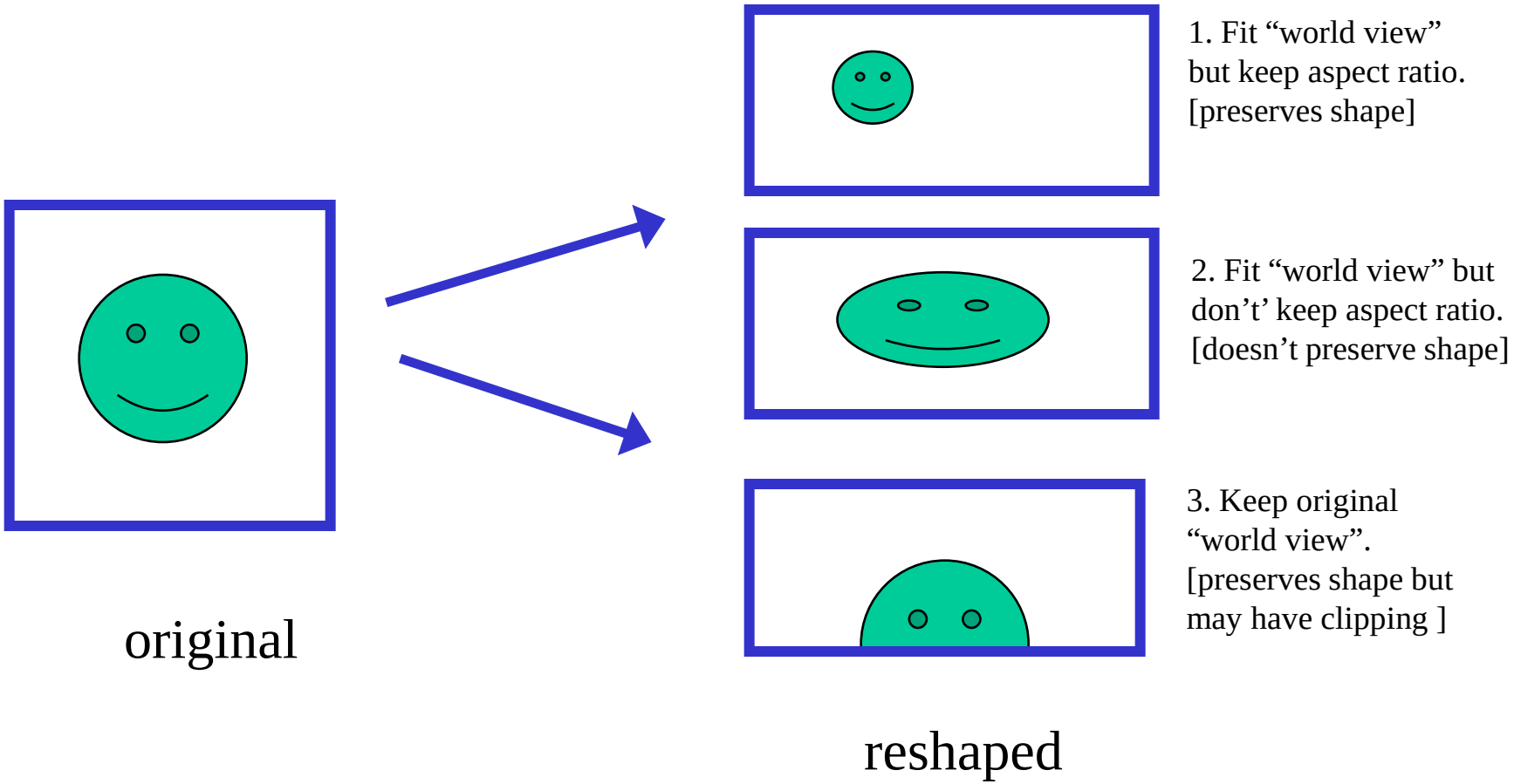
Special and Modifier Keys

- 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 whether any one of the modifiers is pressed
  - **`GLUT_ACTIVE_SHIFT`**, **`GLUT_ACTIVE_CTRL`**, **`GLUT_ACTIVE_ALT`** is depressed using **`glutGetModifiers()`**
    - **`if (glutGetModifiers() == GLUT_ACTIVE_CTRL)`**……
  - 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 Possibilities (typical)



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 (example 1 on slide 29)

- 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 / w,
                           2.0 * (GLfloat) h / w );
  else
    gluOrtho2D( -2.0 * (GLfloat) w / h,
                2.0 * (GLfloat) w / h, -2.0, 2.0 );

  glMatrixMode(GL_MODELVIEW); /* return to modelview mode */
}
```

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*
- Common 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

- See menuExample.c
- **Also see:**
  http://www.opengl.org/resources/libraries/glut/spec3/node35.html

Defining menu's is not that elegant in GLUT. Just keep close track to what you are doing and you'll be OK.

Example of a Glut Window

Basic Glut Menu Functions

- `glutCreateMenu(callBackFunc)`
  - Creates a menu, returns a menu_id
  - This is used for both creating the root menu and sub window
- `glutAddMenuEntry("name", item_id)`
  - Adds a menu item to the "current selected" menu
  - The name entry is "name", when it is clicked it passes the value "item_id" to the associated call back
- `glutAddSubMenu("name", menu_id)`
  - Adds a submenu to the menu whose id is menu_id
  - The name of the submenu is "name"
- `glutSetMenu(menu_id)`
  - Sets the "current selected" menu

Menu Example (1/2)

```
#define MenuItemBig        1
#define MenuItemSmall      2
#define SubMenuColorRed    3
#define SubMenuColorBlue   4
#define MenuItemExit       5

int main(..)
{
   ...
   rootMenu=glutCreateMenu(menuCallback);
   glutAddMenuEntry("Big Point", MenuItemBig);
   glutAddMenuEntry("Small Point", MenuItemSmall);

   subMenu=glutCreateMenu(subMenuCallback);

   … (con't next slide)
```

User defined ids for menu items

Create a new menu. Since it is the first, it is the root. Note that this is now the "current menu". Any add items will appear in this menu.

Notice this menu has a different call back than the root.

Create a new menu. It isn't attached to the root (we need to do that later). It is now the "current menu".

See: menuExample.c

Menu example (2/2)

```
…
glutAddMenuEntry("(Red)",SubMenuColorRed);
glutAddMenuEntry("(Blue)",SubMenuColorBlue);


glutSetMenu(rootMenu);
glutAddSubMenu("Point Colour",subMenu);
glutAddMenuEntry("Exit",MenuItemExit);

glutAttachMenu(GLUT_RIGHT_BUTTON);


glutMainLoop();
}
```

These entries are added to the current menu.

Set the current menu back to rootMenu. Add subMenu, and another entry.

Attach the menu to the right button.

Is it necessary to have two different call back func for each menu?
No. See example: menuExample2.c

Other Functions in GLUT

- Here are some other topics that we don't have time to cover, but you should be aware about. They are reasonably easy to learn on your own.
- Dynamic Windows
  - Create and destroy during execution
- Create Multiple Windows
- Create Subwindows
- Glut Picking/Selection (a bit advanced for beginners)
  - A method for selecting drawn geometry
- Changing callbacks during execution
- Drawing Text
  - **`glutBitmapCharacter`**
  - **`glutStrokeCharacter`**

End of Lecture 3
next -- transformations