EBU5405
3D Graphics Programming Tools

OpenGL: Events and Animation

Dr. Marie-Luce Bourguet
(marie-luce.bourguet@qmul.ac.uk)

1

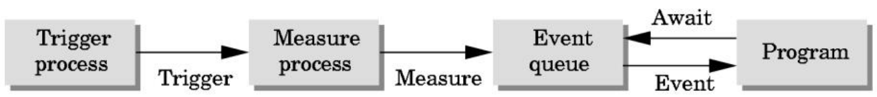
Objectives

- Event-driven input
- Introduce double buffering for smooth animations
- Programming event input with GLUT
- Learn to build interactive programs using GLUT callbacks
  - Mouse
  - Keyboard
  - Reshape
- Introduce menus in GLUT

2

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



3

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 the queue

4

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

5

GLUT callbacks

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

- **`glutDisplayFunc`**
- **`glutIdleFunc`**
- **`glutMouseFunc`**
- **`glutMotionFunc, glutPassiveMotionFunc`**
- **`glutKeyboardFunc, glutKeyboardUpFunc`**
  `glutSpecialFunc, glutSpecialUpFunc`
- **`glutReshapeFunc`**

6

GLUT Event Loop

- Recall that the last line in **`main`** 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

Queen Mary
University of London

7

main

```
int main (int argc, char** argv) {
  glutInit (&argc, argv);
  glutInitWindowSize (ww, wh);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutCreateWindow (“interactive");
  myinit ();
  glutReshapeFunc (myreshape);
  glutMouseFunc (mymouse);
  glutDisplayFunc (mydisplay);
  glutMainLoop ();
}
```

Queen Mary
University of London

8

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 (i.e. draw)
- In **`main`**
  - **`glutDisplayFunc(mydisplay)`** identifies the function to be executed
  - Every GLUT program must have a display callback

Queen Mary
University of London

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

Queen Mary
University of London

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
  - Hence we can see partially drawn displays

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 must request a double buffer in main
  - `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();
}
```

13

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 */
}
```

14

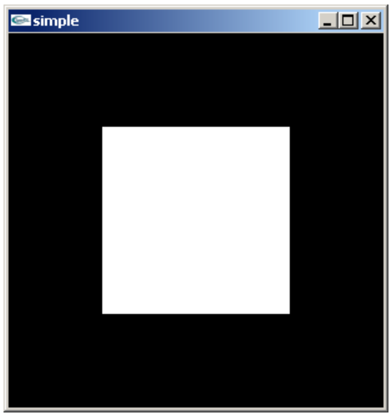
Using globals

```
float   t;
GLfloat pos[3];

void myidle() {
      /* change something */
      t += dt;
      pos[0] = 0; pos[1] = 0; pos[3] = t;
      glutPostRedisplay();
}
void mydisplay() {
      glClear();
      /* draw something at position pos */
      glutSwapBuffers();
}
```

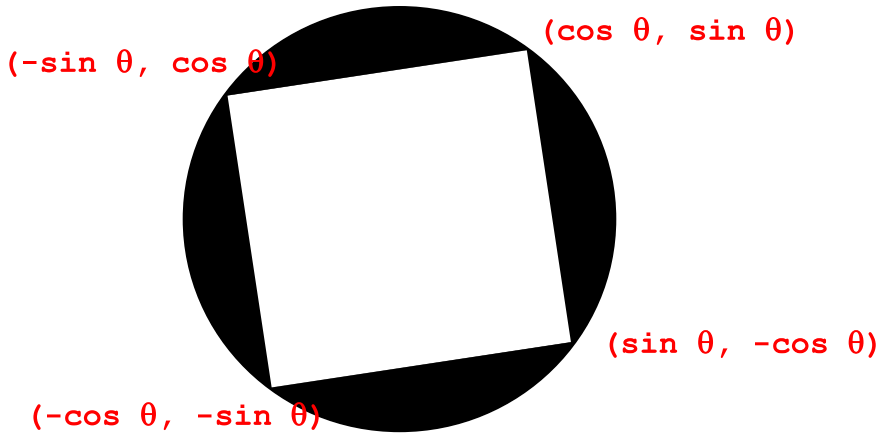
15

E.g. Rotating a square



16

E.g. Rotating a square



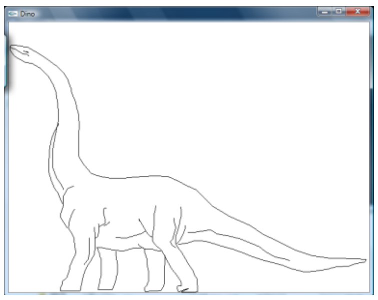
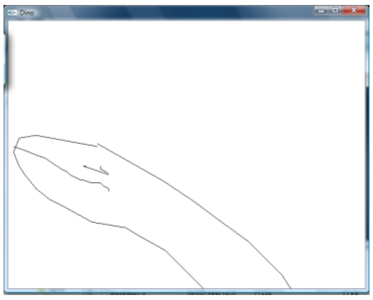
```
#include <math.h>
#define DEGREES_TO_RADIANS 3.14159/180.0
GLfloat theta = 0.0;
GLfloat a, b;

void myidle() {
       theta = theta + 5.0;
       if (theta > 360.0) theta = theta - 360.0;
       a = 0.5 * cos(DEGREES_TO_RADIANS * theta);
       b = 0.5 * sin(DEGREES_TO_RADIANS * theta);
       glutPostRedisplay();
}

void square(){
    glBegin(GL_QUADS);
          glVertex2f(a , b);
          glVertex2f(-b , a);
         glVertex2f(-a , -b);
          glVertex2f(b , -a);
    glEnd();
}

void mydisplay() {
       glClear (GL_COLOR_BUFFER_BIT);
       square();
       glutSwapBuffers ();
}
```

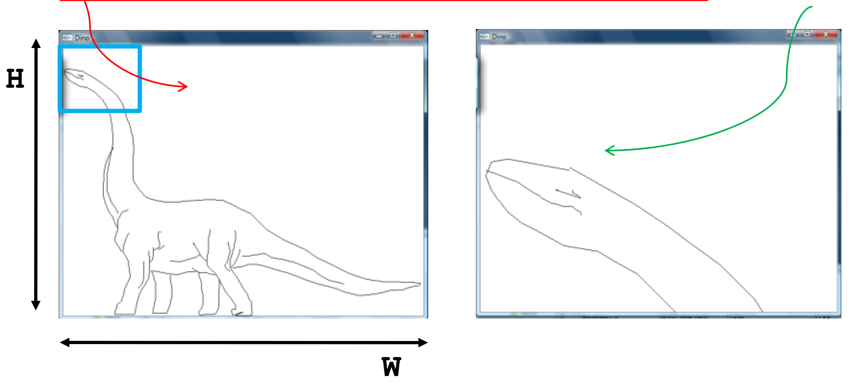
E.g. Zooming and animation

19

E.g. Zooming and animation

```
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
    gluOrtho2D(0.0, (640.0)/5, (4*480.0)/5, 480.0);
```



```
gluOrtho2D (0, W*zoom, H*(1-zoom), H);
0.2 <= zoom <= 1.0
```

20

E.g. Zooming and animation

```
GLdouble zoom = 1.0;

void myidle() {
    if (zoom > 0.2) zoom -= 0.0005;
    glutPostRedisplay();
}

void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
    gluOrtho2D (0, W*zoom, H*(1-zoom), H);
    drawDinosaur …
    glutSwapBuffers();
}
```

21

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

22

Terminating a program

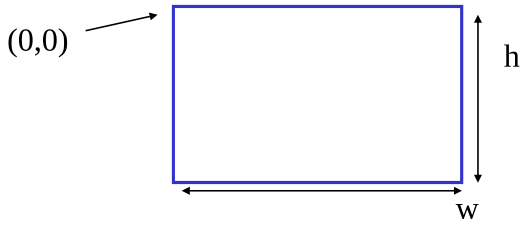
- In our original programs, there was no way to terminate them through OpenGL
- We can use a simple mouse callback

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

23

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;*



24

Obtaining the window size

- To invert the *y* position we need the window height
  - The height can change during program execution
  - It must be tracked using a global variable
  - The new height is returned by the reshape callback (that we will look at in detail soon)
  - We can also use query functions
    - **`glGetIntegerv`**
    - **`glGetFloatv`**

  to obtain any value that is part of the state

25

Obtaining the window size

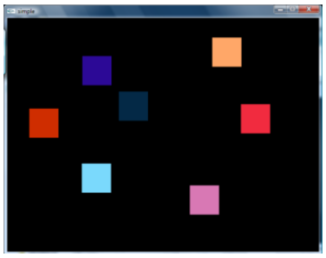
```
glInt viewport[4];
glGetIntegerv (GL_VIEWPORT, viewport);
```

- viewport[0] = x
- viewport[1] = y
- viewport[2] = width
- viewport[3] = height

26

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.



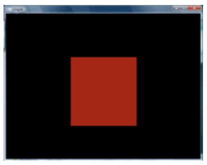
Drawing squares at cursor location

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

void drawSquare(int x, int y)
{
    y = h - y; /* invert y position */
    glColor3ub( (char)rand()%256, (char)rand()%256,
       (char)rand()%256); /* a random color */
    glBegin(GL_POLYGON);
        glVertex2f(x+size, y+size);
        glVertex2f(x-size, y+size);
        glVertex2f(x-size, y-size);
        glVertex2f(x+size, y-size);
     glEnd();
}
```

Using the motion callback

- We can draw squares (or anything else) continuously as long as a mouse button is depressed by using the motion callback
  - `glutMotionFunc(drawSquare)`



Queen Mary University of London

29

Drawing squares continuously

```
GLint xx = 0;
GLint yy = 0;

int main(int argc, char** argv) {
….
  glutDisplayFunc(mydisplay);
  glutMouseFunc(mymouse);
  glutMotionFunc(drawSquare);
….
}

 void mydisplay(){
 }
```

Queen Mary University of London

30

Drawing squares continuously

```
void mymouse(int btn, int state, int x, int y)
{
   GLint viewport[4];
   glGetIntegerv (GL_VIEWPORT, viewport);
   if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
    {
    glColor3ub((char)rand()%256,
     (char)rand()%256,
     (char)rand()%256);
    xx = x;
    yy = viewport[3] - y;
   }
}
```

31

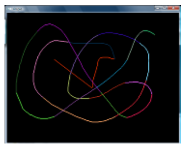
Drawing squares continuously

```
void drawSquare(int x, int y) {
    GLint viewport[4];
    glGetIntegerv (GL_VIEWPORT, viewport);
    y = viewport[3] - y; /* invert y position */
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2i(xx, yy);
        glVertex2i(x, yy);
        glVertex2i(x, y);
        glVertex2i(xx, y);
     glEnd();
     glFlush();
}
```

32

Using the passive motion callback

- We can draw without depressing a button using the passive motion callback
  - `glutPassiveMotionFunc(drawPath)`



33

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

34

Moving the squares with the arrow keys

```
void mykey(unsigned char key, int x, int y)
{
   switch (key) {
   case KEY_LEFTARROW:
               chpos[0] = 1; // set a flag that is used in the idle function to change the position
               break;
   case KEY_RIGHTARROW:
               chpos[0] = -1;
               break;
   case KEY_Q:
               exit(0);
   default:
               printf('Key %d is not defined\n', key);
   }
}

int main(){
               glutKeyboardFunc(mykey);
               glutKeyboardUpFunc(mykeyup);   // this is called when a key is depressed
}
```

35

Special and Modifier Keys

- GLUT defines special keys in **`glut.h`**
  - Function key 1: **`GLUT_KEY_F1`**
  - Up arrow key: **`GLUT_KEY_UP`**
    - **`if(key == 'GLUT_KEY_F1'`** ……
- Uses glutSpecialFunc and glutSpecialUpFunc for the callbacks
- Can also check if one of the modifiers
  - **`GLUT_ACTIVE_SHIFT`**
  - **`GLUT_ACTIVE_CTRL`**
  - **`GLUT_ACTIVE_ALT`**

  is depressed by
  `glutGetModifiers()`

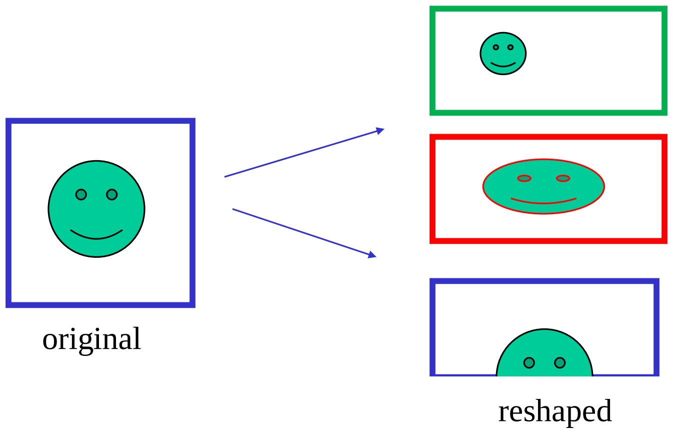
36

Reshaping the window

- We can reshape and resize the OpenGL display window by pulling the corner of the window
- What happens to the display?
  - It must be redrawn

37

Reshape possiblities



38

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 the end of the execution of the callback
- GLUT has a default reshape callback but you probably want to define your own …

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

Queen Mary University of London

39

Example Reshape

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

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

Queen Mary University of London

40

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 just a few widgets including menus

Queen Mary
University of London

41

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 is carried out if the entry is selected
  - Attach menu to a mouse button

Queen Mary
University of London

42

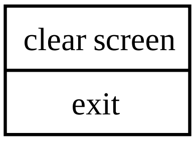
Defining a simple menu

- In **`main.c`**

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

glutAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



entries that appear when right button depressed

identifiers

43

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

44

Other functions in GLUT

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

Queen Mary
University of London