Comp 410/510
Computer Graphics
Spring 2019

Input & Interaction
Objectives

• Introduce the basic input devices
  - Physical Devices
  - Logical Devices
• Input Modes (*Request* vs *Event*)
• Event-driven programming with GLUT
Basic interaction

- 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, etc.)
  - Object changes (moves, rotates, morphs)
  - Repeat
Graphical Input

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

- Graphical input is more varied than input to standard programs (which is usually numbers, characters, or bits)
- Examples to types of logical input:
  - **Locator**: return a position
  - **Pick**: return ID of an object
  - **Keyboard**: return strings of characters
  - **Stroke**: return array of positions
  - **Valuator**: return floating point number (such as slidebars)
  - **Choice**: return one of $n$ items (such as menus)
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 on the keyboard
- When triggered, input devices return information (their *measure*) to the system
  - Mouse returns position information
  - Keyboard returns ASCII code

*Request or Event Mode*
Request Mode

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

- Program waits for inputs that user may possibly provide
- Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user
- Each trigger generates an event whose measure is put in an event queue which can be examined by the user program
Event Types

- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: non-event
  - 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, Mac OS, etc.)

- `glutDisplayFunc(myDisplay)`
- `glutMouseFunc(myMouse)`
- `glutReshapeFunc(myReshape)`
- `glutKeyboardFunc(myKeyboard)`
- `glutIdleFunc(myIdle)`
- `glutMotionFunc(myMotion)`, `glutPassiveMotionFunc`
GLUT Event Loop

- Remember 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 to change the display
    - e.g., using `glutPostRedisplay()`

- In `main.c`
  - `glutDisplayFunc(mydisplay)` identifies the display callback function to be executed
  - Every GLUT program must have a display callback
#include <GL/glut.h>

void myinit(){
    ....
}
void myDisplay(){
    ...
}
void myMouse(int button, int state, int x, int y){
    ....
}
void myKey(unsigned char key, int x, int y){
    ....
}
void myIdle(){
    ...
}
void myMotion(int x, int y){
    ...
}
void myReshape(int w, int h){
    ...
}

int main(int argc, char** argv)
{
    /* window intializations*/
    glutDisplayFunc(mydisplay);
    glutMouseFunc(myMouse);
    glutReshapeFunc(myReshape);
    glutKeyboardFunc(myKey);
    glutIdleFunc(myIdle);
    glutMotionFunc(myMotion);

    myinit();
    glutMainLoop();
}
The mouse callback

- `glutMouseFunc(mymouse)`
- `void mymouse(GLint button, GLint state, GLint x, GLint y)` is returned
  - which button (`GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`, `GLUT_RIGHT_BUTTON`) causes the event
  - state of that button (`GL_UP`, `GLUT_DOWN`)
  - cursor position in window
Terminating a program

We can use a simple mouse callback function to terminate the program execution through OpenGL

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

• The form of all GLUT callbacks is fixed
  - void mydisplay()
  - void mymouse(GLint button, GLint state, GLint x, GLint y)

• Must use global variables to pass information to callbacks:

```c
float t; /*global */

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

```c
void drawSquare(int x, int y)
{
    glUniform2f( Cursor, x, y );
    glUniform3f( RandomColor, ((float) rand()) / RAND_MAX,
                ((float) rand())/RAND_MAX, ((float) rand())/RAND_MAX );
    glutPostRedisplay();
}

void mouse(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);
    }
}
```

See `drawSmallSquares` example
Shaders

in vec4 vPosition;
uniform vec2 cursor;
void
main()
{
    gl_Position =
        vec4(vPosition.x + 2.0*cursor.x/500.0, vPosition.y +
             2.0*(500.0-cursor.y)/500.0, 0, 1);
}

uniform vec3 randomColor;
out vec4 fragColor;
void
main()
{
    vec4 color = vec4( randomColor.r, randomColor.g,
                      randomColor.b, 1.0);
    fragColor = color;
}
Positioning

• The cursor position in the screen window is usually measured in pixels with the origin at the top-left corner
  • That is because the refresh is done from top to bottom
• OpenGL uses a default world coordinate system with origin located at the center of the window
  • Must invert $y$ coordinate (returned by callback) using the height $h$ of window:
    • $y = h - y$;
Obtaining the window size

• To invert the $y$ position we need to know the window height
  - Height $h$ can change during program execution
  - Can track with a global variable
  - New height is returned to reshape callback that we will look at in detail soon
  - Can also use enquiry functions

  • `glutGet(GLUT_WINDOW_Y)`
  • `glGetIntv(...)`
  • `glGetFloatv(...)`

  to obtain any value that is part of the state
Using the motion callback

- We could draw squares (or anything else) continuously as long as a mouse button is depressed by using the motion callback:
  - `glutMotionFunc(drawSquare)`
- We could draw squares without depressing a button using the passive motion callback:
  - `glutPassiveMotionFunc(drawSquare)`
- The system returns to the callback function the $x$ and $y$ positions of the cursor.
Using the keyboard

- **glutKeyboardFunc** (mykey)
- **void mykey(unsigned char key, int x, int y)**
  - Is returned the ASCII code of the key depressed and the mouse location
  - Note that GLUT does not recognize the key release as an event

```c
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 modifiers
  - GLUT_ACTIVE_SHIFT
  - GLUT_ACTIVE_CTRL
  - GLUT_ACTIVE_ALT

are depressed by
  glutGetModifiers()

- which allows emulation of three-button mouse with one- or two-button mice
We can reshape and resize the OpenGL display window by pulling the corner of the window.

What happens to the display then?
- Must redraw the scene from application
- Three possibilities
  - Display part of the world
  - Display the whole world but force to fit in new window
    - Can alter aspect ratio
  - Display with modified view volume (or area)
Reshape possibilities

- Display part of the world
- Display the whole world but force to fit in new window
- Display with modified view volume (aspect ratio is not changed)
The Reshape callback

- `glutReshapeFunc(myreshape)`
- `void myreshape(int w, int h)`
  - Is returned width and height of the new resized window (in pixels)
  - A redisplay is posted automatically at the end of execution of the callback
  - GLUT has a default reshape callback but you probably want to define your own
- The reshape callback is a good place to put camera functions because it is also invoked when the window is first opened
Example Reshape

The reshape callback below preserves shapes by making the viewport and the view volume have the same aspect ratio

```c
void reshape( int w, int h )
{
  glViewport( 0, 0, w, h );

  mat4 projection;
  if (w <= h)
    projection = Ortho(-1.0, 1.0, -1.0 * (GLfloat) h / (GLfloat) w, 1.0 * (GLfloat) h / (GLfloat) w, -1.0, 1.0);
  else projection = Ortho(-1.0* (GLfloat) w / (GLfloat) h, 1.0 * (GLfloat) w / (GLfloat) h, 1.0 * (GLfloat) w / (GLfloat) h, -1.0, 1.0, -1.0, 1.0);

  glUniformMatrix4fv( Projection, 1, GL_TRUE, projection );
}
```

See `spinCubeRevisited` example
OpenGL/GLUT program structure

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

void myinit()
{
    ....
}
void myDisplay()
{
    ....
}
void myMouse(int button, int state, int x, int y)
{
    ....
}
void myKey(unsigned char key, int x, int y)
{
    ....
}
void myIdle()
{
    ...
}
void myMotion(int x, int y)
{
    ...
}
void myReshape(int w, int h)
{
    ...
}

int main(int argc, char** argv)
{
    /* window intializations*/
    glutDisplayFunc(mydisplay);
    glutMouseFunc(myMouse);
    glutReshapeFunc(myReshape);
    glutKeyboardFunc(myKey);
    glutIdleFunc(myIdle);
    glutMotionFunc(myMotion);
    myinit();
    glutMainLoop();
}
```
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 each event loop
- If set then the display callback function is executed

```c
void idle( void )
{
    Theta[Axis] += 4.0;
    if ( Theta[Axis] > 360.0 ) {
        Theta[Axis] -= 360.0;
    }

    glutPostRedisplay();
}
```
Using glutTimerFunc

- `void glutTimerFunc(unsigned int msecs, void (*func)(int value), value);`

- `glutTimerFunc` registers the timer callback `func` to be triggered in at least `msecs` milliseconds.

```c
void myTimerFunc () {
/* change something */
t += dt;
glutPostRedisplay();
glutTimerFunc(20,myTimerFunc,0);
}
```

```c
int main(int argc, char **argv)
{
    ..... 
    glutTimerFunc(20,myTimerFunc,0);
    ...
}
```

See `spinCubeRevisited` example
Hidden-Surface Removal

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

- The algorithm uses an extra buffer, the z-buffer, to store depth information as geometry travels down the pipeline
- It must be
  - Requested in `main()`
    - `glutInitDisplayMode`
    - `(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)`
  - Enabled in `init()`
    - `glEnable(GL_DEPTH_TEST)`
  - Cleared in the display callback
    - `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)"
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
  - Input boxes
• But toolkits tend to be platform dependent
• GLUT is platform-independent and provides a few widgets including pop-up menus
Menus

- GLUT supports pop-up menus
  - A menu can have submenus

- Three steps
  1. Define entries (items) for the menu
  2. Define action for each menu item
     - Action carried out if entry selected
  3. Attach menu to a mouse button
Defining a simple menu

- In `main.c`

```c
glutCreateMenu(mymenu);

glutAddMenuEntry("clear screen", 1);

glutAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```

<table>
<thead>
<tr>
<th>clear screen</th>
<th>exit</th>
</tr>
</thead>
</table>

Menu entries that appear when right button depressed

Identifiers returned to callback function

Callback function

In `main.c`
**Menu actions**

- Menu callback

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

- Note each menu item has an id that is returned when selected
- Add submenus by

```c
    glutAddSubMenu(char *submenu_name, submenu id)
```

entry in parent menu
Example

```c
    c_menu = glutCreateMenu(color_menu);
    glutAddMenuEntry("Red", 1);
    glutAddMenuEntry("Green", 2);
    ...
    glutAddMenuEntry("Black", 8);

    p_menu = glutCreateMenu(pixel_menu);
    glutAddMenuEntry("increase pixel size", 1);
    glutAddMenuEntry("decrease pixel size", 2);

    f_menu = glutCreateMenu(fill_menu);
    glutAddMenuEntry("fill on", 1);
    glutAddMenuEntry("fill off", 2);

    glutCreateMenu(middle_menu);
    glutAddSubMenu("Colors", c_menu);
    glutAddSubMenu("Pixel Size", p_menu);
    glutAddSubMenu("Fill", f_menu);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);

    glutCreateMenu(right_menu);
    glutAddMenuEntry("quit", 1);
    glutAddMenuEntry("clear", 2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
```
void right_menu(int id)
{
    if(id == 1) exit(0);
    else glutPostRedisplay();
}

void middle_menu(int id)
{
}

void color_menu(int id)
{
    if(id == 1) {r = 1.0; g = 0.0; b = 0.0;}
    else if(id == 2) {r = 0.0; g = 1.0; b = 0.0;}
    ....
    else if(id == 8) {r = 0.0; g = 0.0; b = 0.0;}
}

void pixel_menu(int id)
{
    if (id == 1) size = 2 * size;
    else if (size > 1) size = size/2;
}

void fill_menu(int id)
{
    if (id == 1) fill = 1;
    else fill = 0;
}
Other functions in GLUT

• Dynamic Windows
  - Create and destroy during execution
• Subwindows
• Multiple Windows
• Changing callbacks during execution
• Portable fonts
Picking

• Identifying 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 a position corresponds
• Practical difficulties
  - Pipeline architecture is fed forward, hard to go from screen back to world
  - Complicated by screen being 2D, world is 3D
  - How close should we come to object to say “we’ve selected it”?
Two Approaches

1. Use of some other buffer to store object ids as the objects are rendered
2. Use of rectangular maps
   • Easy to implement for many applications
Using of Rectangular Maps

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

Easier to look at mouse position and determine which region of screen it is in than using more sophisticated techniques
Using another buffer and color for picking

• We first assign a unique color to each object (not necessarily the true color of the object)
• We then render the scene to a color buffer other than the front buffer, so the results of the rendering are not visible
  - e.g., can use `glDrawBuffer()`
  - e.g., can render to back buffer
• We then get the mouse position and use `glReadPixels()` function to read the color in the buffer, which is at the position of the mouse
• The returned color gives the id of the object

Pick tree object if cursor pixel is red