

---

# CIS 454 Computer Graphics

## Lecture 7, 09/26/2006

---

Li Shen  
Computer and Information Science  
UMass Dartmouth

---

## Notes

- Thursday: HW1 due
-

---

## 3D OpenGL Programming

- Develop a more sophisticated three-dimensional example
    - Sierpinski gasket: a fractal
  - Introduce hidden-surface removal
- 

---

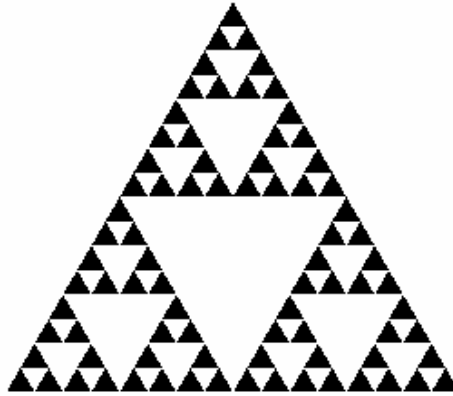
## Three-dimensional Applications

- In OpenGL, two-dimensional applications are a special case of three-dimensional graphics
  - Going to 3D
    - Not much changes
    - Use `glVertex3*` ( )
    - Have to worry about the order in which polygons are drawn or use hidden-surface removal
    - Polygons should be simple, convex, flat
-

---

## Sierpinski Gasket (2D)

- Five subdivisions



---

## Gasket Program

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

/* initial triangle */

GLfloat v[3][2]={{-1.0, -0.58},
                 {1.0, -0.58}, {0.0, 1.15}};

int n; /* number of recursive steps */
```

OpenGL <http://www.rush3d.com/reference/opengl-bluebook-1.0/>  
GLUT <http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

---

---

## Draw one triangle

```
void triangle( GLfloat *a, GLfloat *b, GLfloat
             *c)

/* display one triangle */
{
    glVertex2fv(a);
    glVertex2fv(b);
    glVertex2fv(c);
}
```

---

---

## Triangle Subdivision

```
void divide_triangle(GLfloat *a, GLfloat *b, GLfloat *c, int
                    m)
{
    /* triangle subdivision using vertex numbers */
    GLfloat v0[2], v1[2], v2[2];
    int j;
    if(m>0)
    {
        for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
        for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
        for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
        divide_triangle(a, v0, v1, m-1);
        divide_triangle(c, v1, v2, m-1);
        divide_triangle(b, v2, v0, m-1);
    }
    else(triangle(a,b,c));
    /* draw triangle at end of recursion */
}
```

---

---

## display and init Functions

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        divide_triangle(v[0], v[1], v[2], n);
    glEnd();
    glFlush();
}

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
    glMatrixMode(GL_MODELVIEW);
    glClearColor (1.0, 1.0, 1.0,1.0)
    glColor3f(0.0,0.0,0.0);
}
```

---

---

## main Function

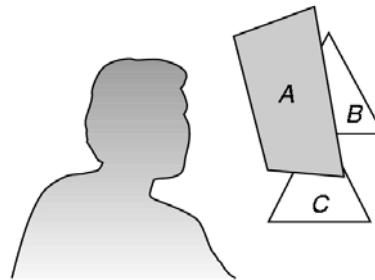
```
int main(int argc, char **argv)
{
    n=4;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("2D Gasket");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

---

---

## Hidden-Surface Removal

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



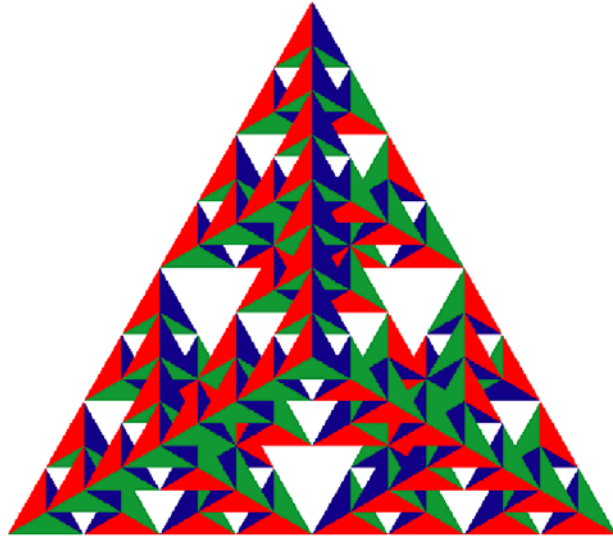
---

## 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.c`
    - `glutInitDisplayMode`  
(`GLUT_SINGLE` | `GLUT_RGB` | `GLUT_DEPTH`)
  - Enabled in `init.c`
    - `glEnable(GL_DEPTH_TEST)`
  - Cleared in the display callback
    - `glClear(GL_COLOR_BUFFER_BIT |`  
`GL_DEPTH_BUFFER_BIT)`

---

# Volume Subdivision



---

# Input and Interaction

---

## 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
-

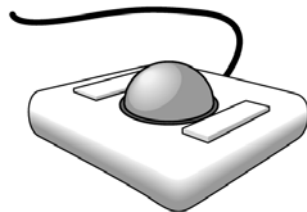
## Graphical Input

- 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

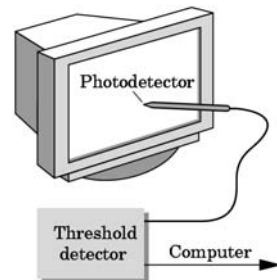
## Physical Devices



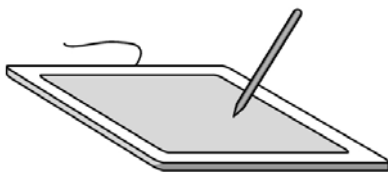
mouse



trackball



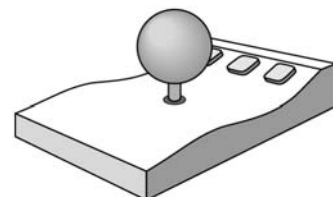
light pen



data tablet



joystick



space ball

---

## Incremental (Relative) Devices

- Devices such as the data tablet return a position directly to the operating system
  - Devices such as the mouse, trackball, and joy stick return incremental inputs (or velocities) to the operating system
    - 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

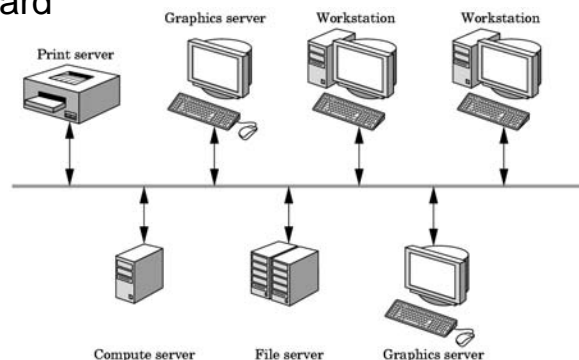
- Consider the C and C++ code
    - C++: `cin >> x;`
    - C: `scanf("%d", &x);`
  - What is the input device?
    - Can't tell from the code
    - Could be keyboard, file, output from another program
  - The code provides *logical input*
    - A number (an `int`) is returned to the program regardless 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 APIs (GKS, PHIGS) defined six 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
  - **Choice**: return one of n items

## 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

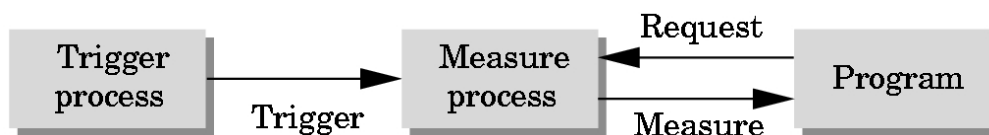


## Input Modes

- 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

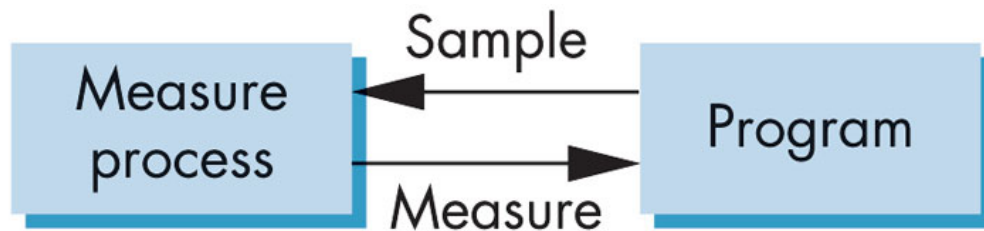
## Request Mode

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



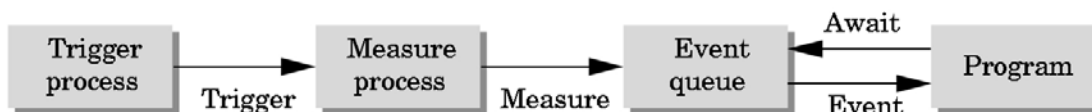
## Sample Mode

- Input is immediate
- No trigger needed
- Function call  $\leq$  measure



## 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



---

## 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)`

Example

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
-

---

## 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 redisplay

- 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
    - 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
- 

---

## 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  
}
```

---

---

## After Class

- Read Chapter 3
  - Work on HW1
-