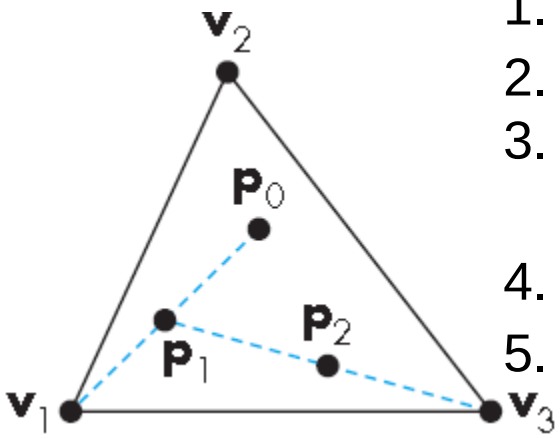
Today's Agenda

Programming with shader-based OpenGL: From 2D to 3D

Input and Interaction

Sierpinski Gasket

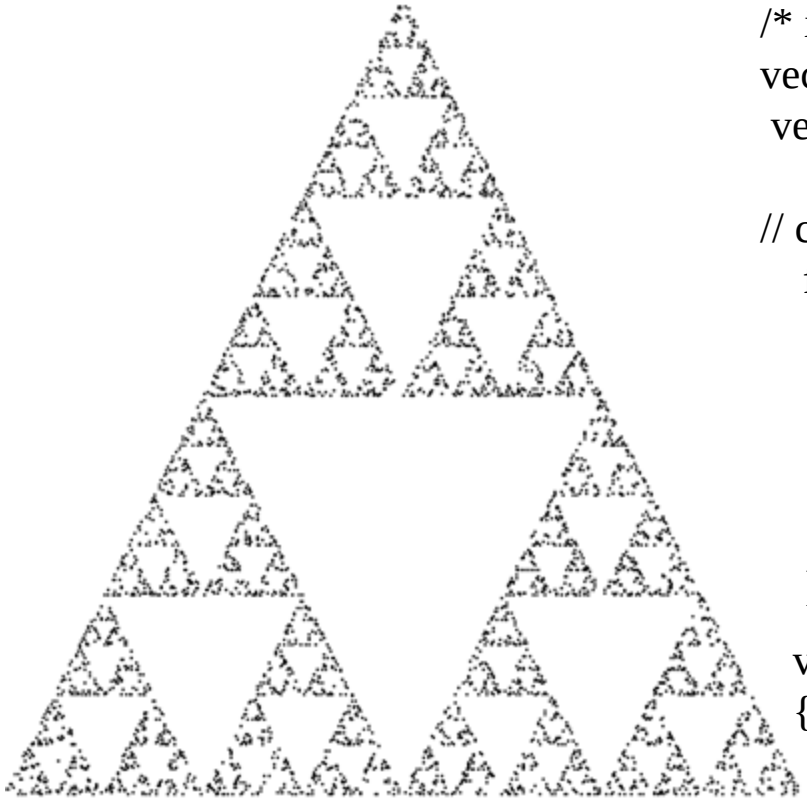
A fractal object defined recursively and randomly



1. Pick an initial point $\mathbf{p}_0$ randomly inside the triangle
2. Select one of the vertices randomly
3. Find a point $\mathbf{p}_1$ at the middle of the line segment $\mathbf{v}_1\mathbf{p}_0$
4. Replace $\mathbf{p}_0$ with $\mathbf{p}_1$
5. Go back to step 2.

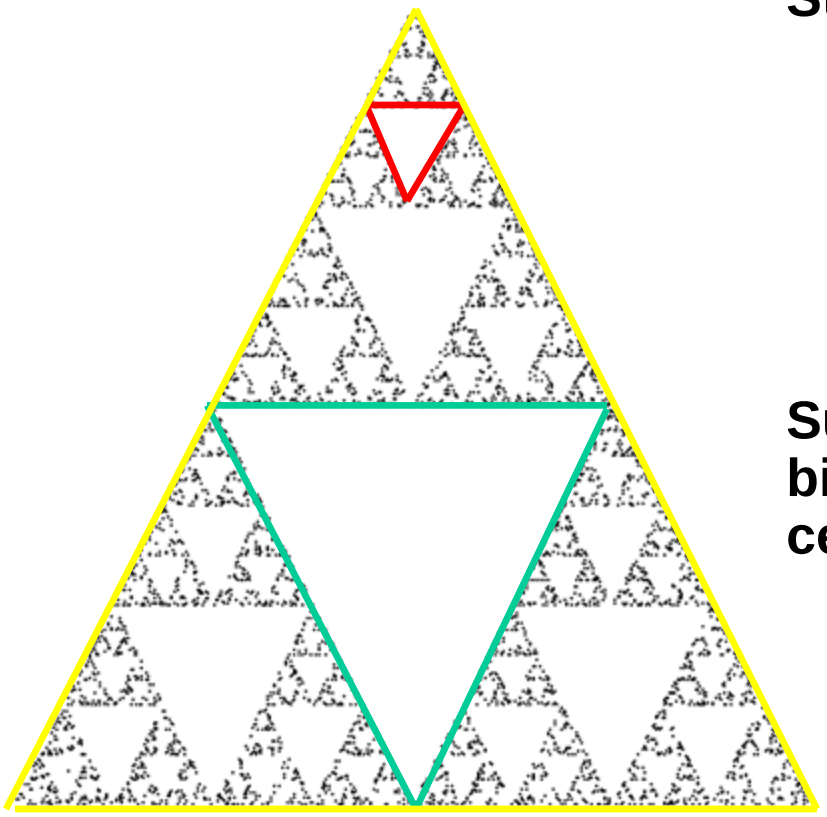
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Sierpinski Gasket – Isolated Points

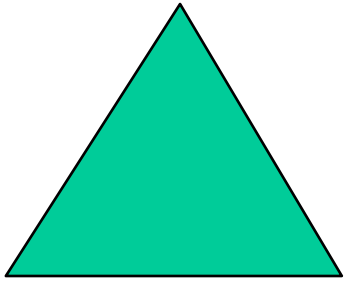


```
#include <GL/glut.h>
/* initial triangle */
vec2 vertices[3] ={vec2(-1.0, -0.58), vec2(1.0, -0.58),
 vec2(0.0, 1.15)};

// compute and store N-1 new points
    for ( int i = 1; i < NumPoints; ++i ) {
        int j = rand() % 3;   // pick a vertex at random

        // Compute the point halfway between the selected vertex
        //   and the previous point
        points[i] = ( points[i - 1] + vertices[j] ) / 2.0;
    }

  void display( void )
  {
      glClear( GL_COLOR_BUFFER_BIT ); // clear the window
      glDrawArrays( GL_POINTS, 0, NumPoints );// drawpoints
      glFlush();
  }
```

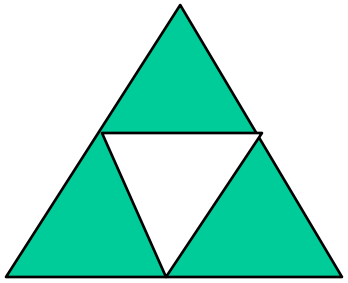
Sierpinski Gasket – Polygons



Start from a single triangle

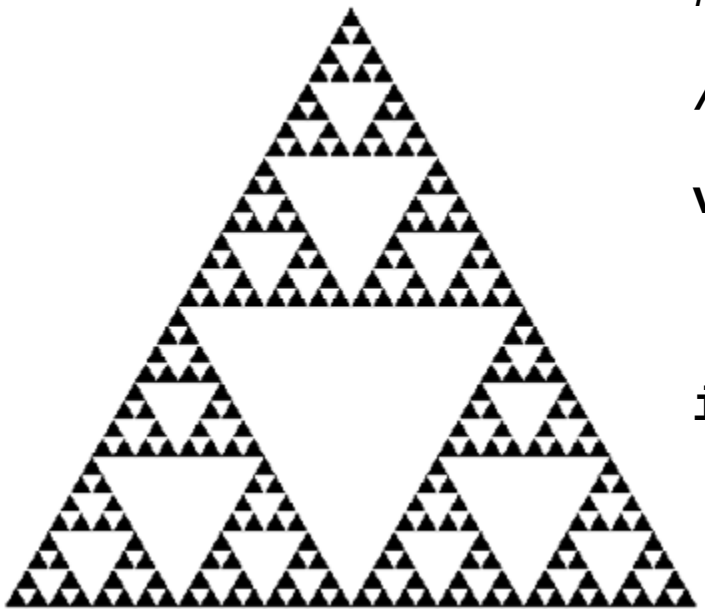


Subdivide it into 4 smaller ones by bisecting the sides and remove the central one



Repeat

Sierpinski Gasket – Polygons



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

/* initial triangle */

vec2 v[3] ={point2(-1.0, -0.58),
            point2(1.0, -0.58),
            point2 (0.0, 1.15)};

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

Draw one triangle

```
void triangle( vec2 a, vec2 b, vec2 c)

/* display one triangle  */
{
      static int i =0;

      points[i] = a;
      points[i] = b;
      points[i] = c;
      i += 3;
}
```

Triangle Subdivision

```
void divide_triangle(vec2 a, vec2 b, vec2 c, int m)
{
/* triangle subdivision using vertex numbers */
    point2 ab, ac, bc;
    if(m>0)
    {
        ab = (a + b )/2;
        ac = (a + c)/2;
        bc = (b + c)/2;
        divide_triangle(a, ab, ac, m-1);
        divide_triangle(c, ac, bc, m-1);
        divide_triangle(b, bc, ac, m-1);
    }
    else(triangle(a,b,c));
 /* draw triangle at end of recursion */
}
```

display and init Functions

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);
    glFlush();
}

void myinit()
{
    vec2 v[3] = {point2(……
    .
    .
    divide_triangles(v[0], v[1], v[2], n);
    .
    .
    glBufferData( GL_ARRAY_BUFFER,sizeof(points),
points, GL_STATIC_DRAW );
    .
    .
}
```

main Function

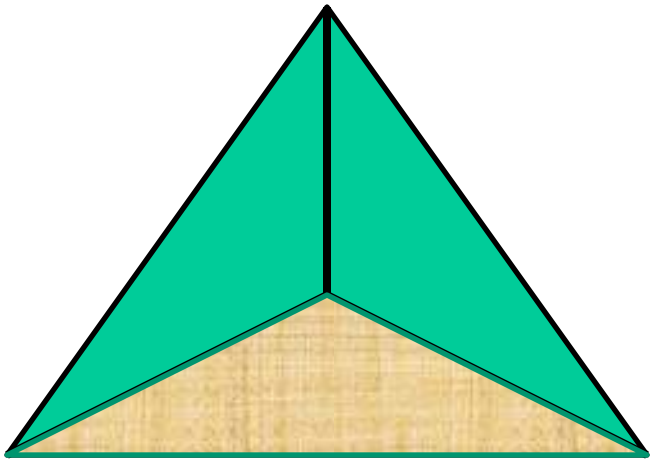
```
int main(int argc, char **argv)
{
   n=4;
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutCreateWindow("2D Gasket");
   glutDisplayFunc(display);

   myinit();
   glutMainLoop();
}
```

Moving to 3D

We can easily make the program three-dimensional by using

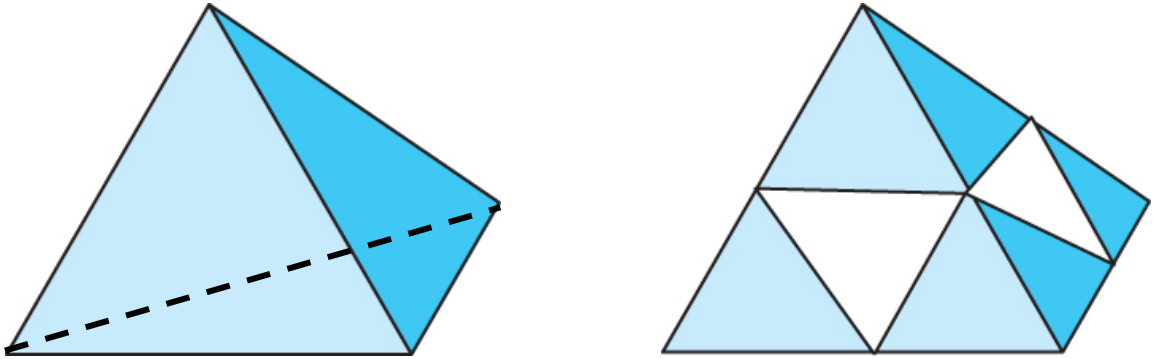
```
vec3 v[3]
```

and we start with a tetrahedron (a polyhedron with 4 vertices, 6 straight edges, and 4 triangle faces)



3D Gasket

We can subdivide each of the four faces

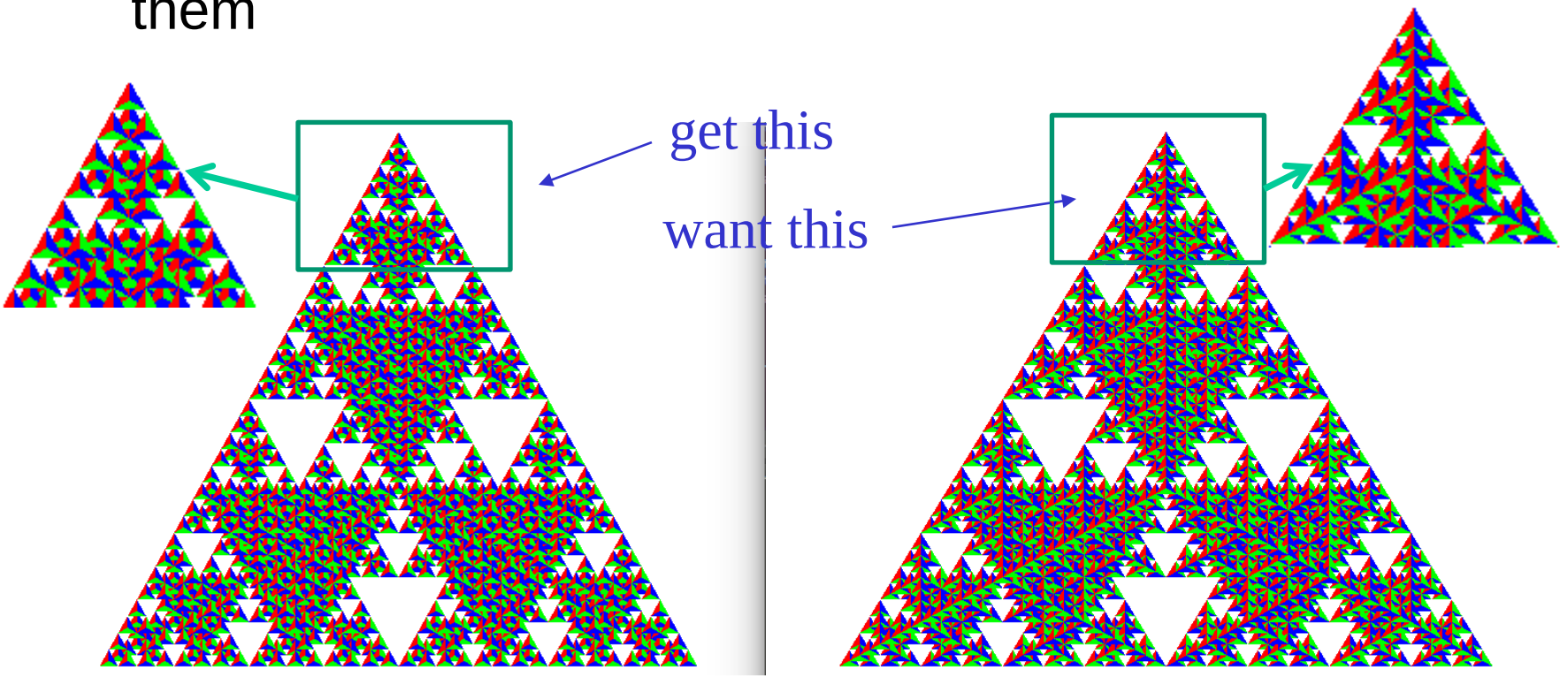


Appears as if we remove a solid tetrahedron from the center leaving four smaller tetrahedra

Code almost identical to 2D example

Almost Correct

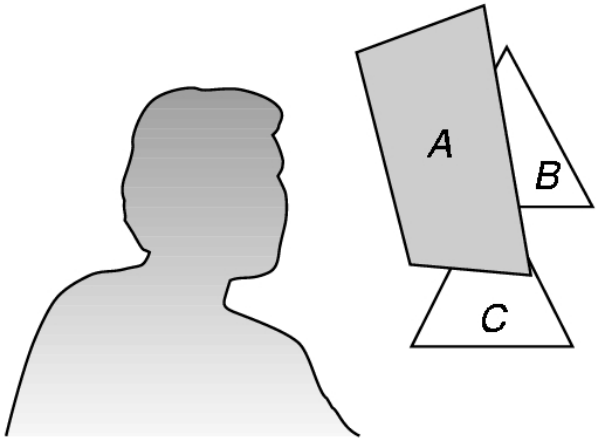
Because the triangles are drawn in the order they are generated/specified in the program, the front triangles are not always rendered in front of triangles behind them



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



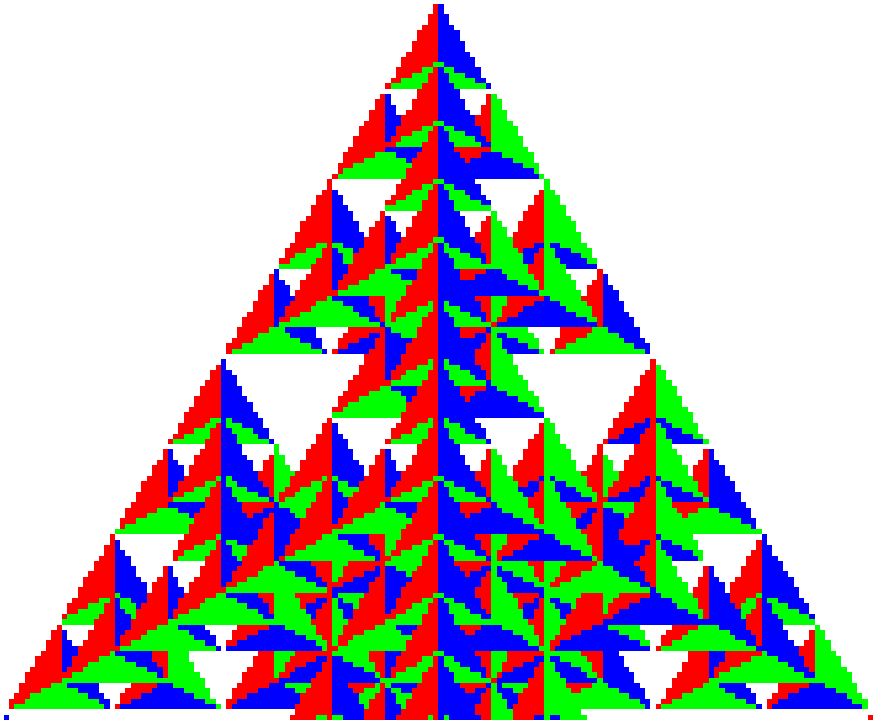
Using the *z*-buffer algorithm

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)`**

Using the *z*-buffer algorithm



Input and Interaction

Introduce the basic input devices

Event-driven input

Programming event input with GLUT

Graphical Input

Devices can be described either by

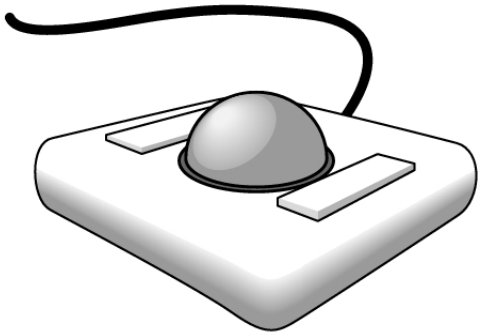
- Physical properties
  - Mouse
  - Keyboard
  - Trackball
- Logical Properties
  - What is returned to program via API
    - A position
    - An object identifier

Modes

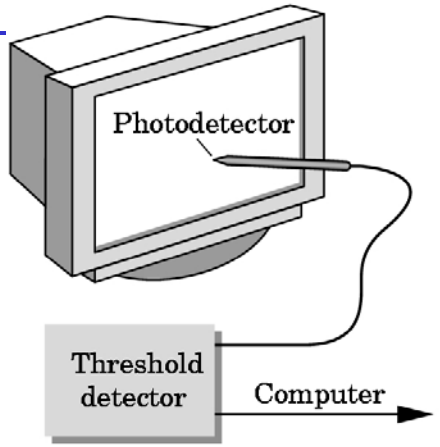
- How and when input is obtained
  - Request mode
  - Event mode

Physical Devices

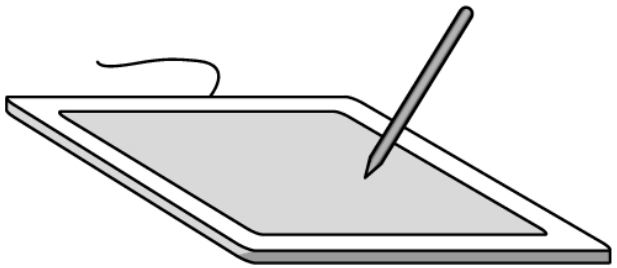
mouse



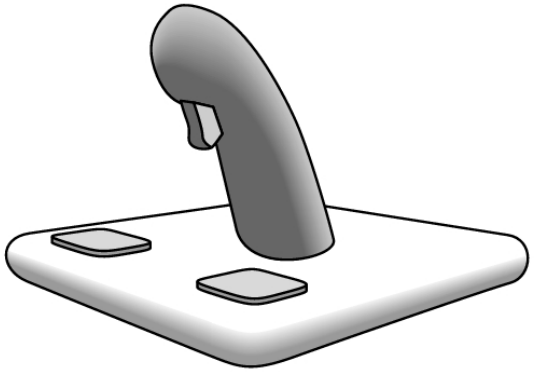
trackball



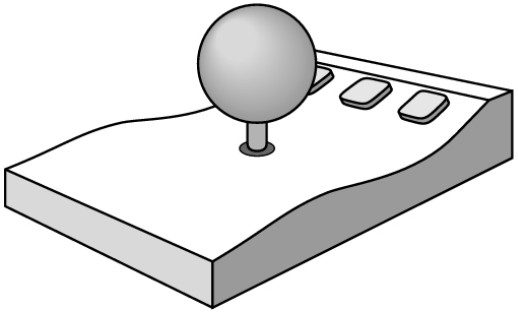
light pen



data tablet



joy stick



space ball

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

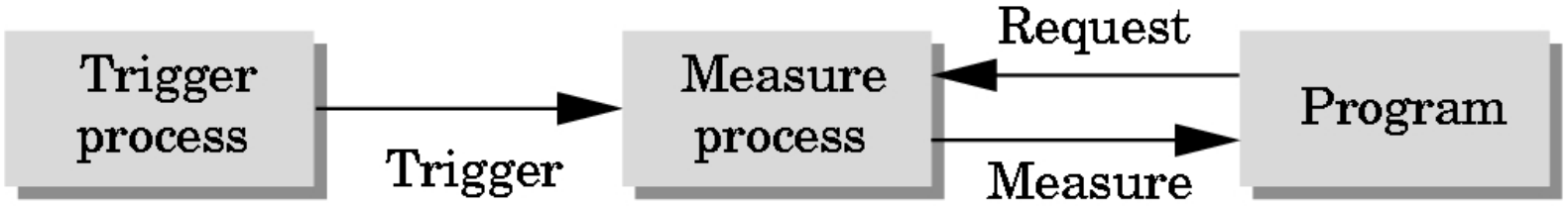
Request Mode

Input provided to program only when user triggers the device

The application and input process cannot work at the same time

Typical of keyboard input

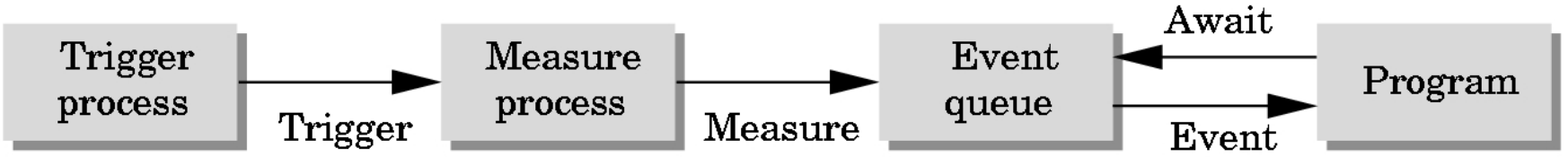
- Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed



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

Callback Functions

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)`

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

Interactive Programs

Learn to build interactive programs using GLUT callbacks

- Mouse
- Keyboard
- Reshape

Introduce menus in GLUT

Mouse Event

- **Move event** happens when the mouse is moved with one or more buttons pressed
- **Passive move event** happens when the mouse is moved with no button pressed
- **Mouse event** happens when one of the buttons is
  - Depressed – Mouse down event
  - Released – mouse up event

Information includes

- which button triggers the event
  - **`GLUT_LEFT_BUTTON`**, **`GLUT_MIDDLE_BUTTON`**, **`GLUT_RIGHT_BUTTON`**
- The state of the button after the event (**`GLUT_UP`**, **`GLUT_DOWN`**)
- The position in window coordinates

Mouse Event

Register the mouse callback function in *main()*

- **Mouse move event**: *glutMotionFunc(myMouseMotion);*
  - Defined as

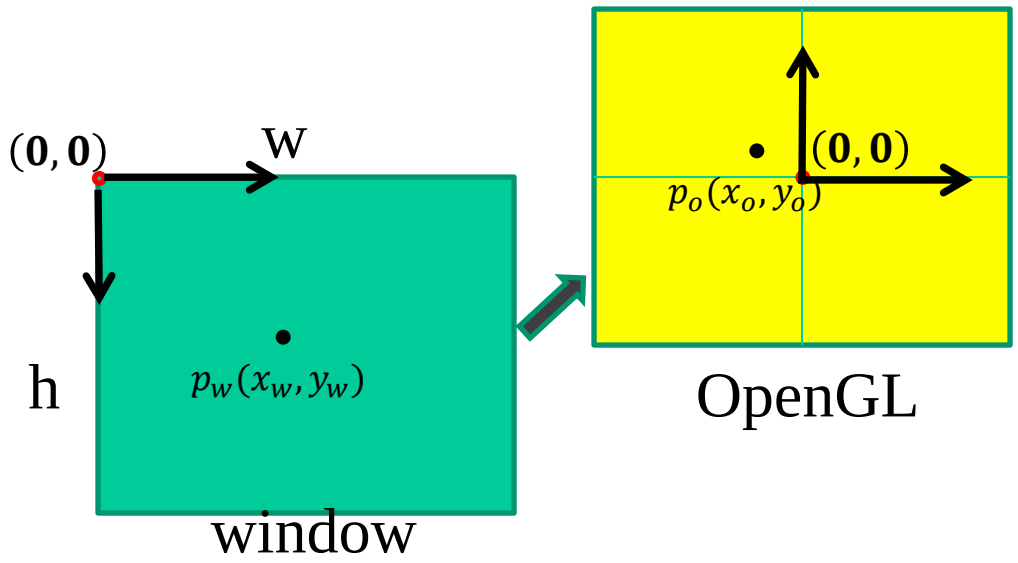
    void myMouseMotion(int x, int y);
- **Passive mouse move event**: *glutPassiveMotionFunc(myMousePMotion)*
  - Defined as

    void myMousePMotion(int x, int y);

- **Mouse event:** *glutMouseFunc(myMouse);*
  - Defined as

    void myMouse (int button, int state, int x, int y);

Coordinate Mapping



$$x_o = \frac{x_w}{0.5w} - 1$$

$$y_o = 1 - \frac{y_w}{0.5h}$$

How to get the values of w and h?
Using query functions
w= glutGet(GLUT_WINDOW_WIDTH);
h=glutGet(GLUT_WINDOW_HEIGHT);

Example: Terminating a program

In our original programs, there was no way to terminate them through OpenGL

We can use the simple mouse callback

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

Example: Mouse Event

```
int w, h;
int count = 0;
void myMouse (int button, int state, int x, int y)
{
   if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
   {
       exit(0);
    }
   if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
   {
       vertices[count].x = (float) x / (w/2) - 1.0;
       vertices[count].y = (float) (h-y) / (h/2) - 1.0;
       count++;
   }
   if(count == 3)
   {
       glutPostRedisplay();
       count = 0;
    }
}
```

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

An Example: Draw a Triangle Using Mouse

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBindVertexArray(VAOs[Triangles]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);
 glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
              vertices, GL_STATIC_DRAW);
    glFlush();
}
```

What did we miss?

Idle Callback

When no events are pending, use an idle callback function

void **glutIdleFunc**(void (**func*)(void));

- Null or an idle function
- Background process
- Continuous animation

```
void idle()
{
glutPostRedisplay();
}
```