

Objectives

- Develop a more sophisticated three-dimensional example
 - Sierpinski gasket: a fractal
- Introduce hidden-surface removal
- Event-driven input
- Introduce the basic input devices
 - Physical and Logical Devices, Input Modes
- Introduce double buffering for smooth animations
- Programming event input with GLUT



Slides by Edward Angel © 2002

Three-dimensional Applications

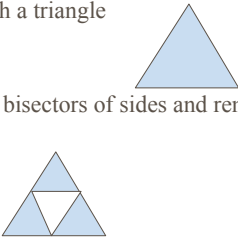
- In OpenGL, two-dimensional applications are a special case of three-dimensional graphics
 - Not many changes
 - Use `glVertex3*` ()
 - Have to worry about the order in which polygons are drawn or use hidden-surface removal
 - Polygons should be simple, convex, flat



Slides by Edward Angel © 2002

Sierpinski Gasket (2D)

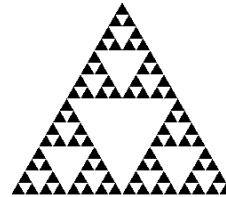
- Start with a triangle
- Connect bisectors of sides and remove central triangle
- Repeat



Slides by Edward Angel © 2002

Example

- Five subdivisions



Slides by Edward Angel © 2002

The gasket as a fractal

- Consider the filled area (black) and the perimeter (the length of all the lines around the filled triangles)
- As we continue subdividing
 - the area goes to zero
 - but the perimeter goes to infinity
- This is not an ordinary geometric object
 - It is neither two- nor three-dimensional
- It is a *fractal* (fractional dimension) object



Slides by Edward Angel © 2002

Fractals

- Possible definitions:
 - <http://www.mrob.com/pub/muency/fractaldefinitionof.html>
 - *divergent measure*: Any shape that when its length, area, surface area or volume is measured in discrete finite units the measured value increases without bound as the discrete unit decreases to zero.
 - *self-similarity*: Any object that is *self-similar* in a non-trivial manner.
 - *Hausdorff definition*: Any geometric form with a non-integral *Hausdorff dimension*.
 - *natural definition*: A geometric figure or object that : a) parts have the same form or structure as the whole, except but differ in scale; b) is extremely irregular or fragmented, independent of scale; c) "distinct elements" with varying scale and large range.



Slides by Edward Angel © 2002

Gasket Program

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

/* a point data type
typedef GLfloat point2[2];

/* initial triangle */

point2 v[]={{-1.0, -0.58}, {1.0, -
0.58}, {0.0, 1.15}};

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



Slides by Edward Angel © 2002

Draw a triangle

```
void triangle( point2 a, point2 b,
point2 c)

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



Slides by Edward Angel © 2002

Triangle Subdivision

```
void divide_triangle(point2 a, point2 b, point2 c, int m){
/* triangle subdivision using vertex numbers */
point2 v0, v1, v2;
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 */}
```



Slides by Edward Angel © 2002

Display and Init Functions

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



Slides by Edward Angel © 2002

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();
}
```



Slides by Edward Angel © 2002

Moving to 3D

- We can easily make the program three-dimensional by using

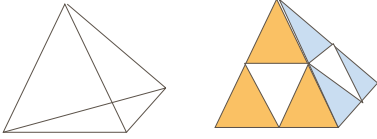
```
typedef GLfloat point3[3]
glVertex3f
glOrtho
```
- But that would not be very interesting
- Instead, we can start with a tetrahedron



Slides by Edward Angel © 2002

3D Gasket

- We can subdivide each of the four faces



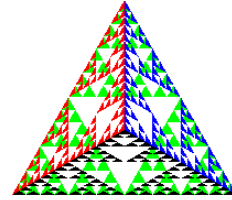
- Appears as if we remove a solid from the center leaving four smaller tetrahedra



Slides by Edward Angel © 2002

Example

after 5 iterations



Slides by Edward Angel © 2002

triangle code

```
void triangle( point a, point b, point
c)
{
    glBegin(GL_POLYGON);
    glVertex3fv(a);
    glVertex3fv(b);
    glVertex3fv(c);
    glEnd();
}
```



Slides by Edward Angel © 2002

subdivision code

```
void divide_triangle(point a, point b, point c, int m){
    point v1, v2, v3;
    int j;
    if(m>0) {
        for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;
        for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;
        for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;
        divide_triangle(a, v1, v2, m-1);
        divide_triangle(c, v2, v3, m-1);
        divide_triangle(b, v3, v1, m-1);
    } else(triangle(a,b,c));}
```



Slides by Edward Angel © 2002

tetrahedron code

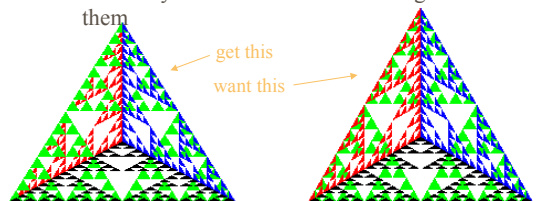
```
void tetrahedron( int m)
{
    glColor3f(1.0,0.0,0.0);
    divide_triangle(v[0],v[1],v[2],m);
    glColor3f(0.0,1.0,0.0);
    divide_triangle(v[3],v[2],v[1],m);
    glColor3f(0.0,0.0,1.0);
    divide_triangle(v[0],v[3],v[1],m);
    glColor3f(0.0,0.0,0.0);
    divide_triangle(v[0],v[2],v[3],m);
}
```



Slides by Edward Angel © 2002

Almost Correct

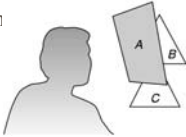
- Because the triangles are drawn in the order they are defined in the program, the front triangles are not always rendered in front of triangles behind them



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

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)`



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

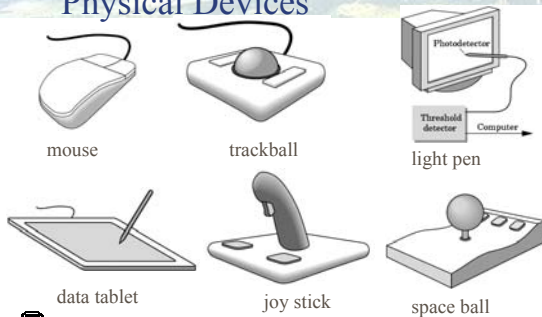
Graphical Input

- 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



Slides by Edward Angel © 2002

Physical Devices



Slides by Edward Angel © 2002

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 wheels in mouse
 - Roll of trackball
 - Difficult to obtain absolute position
 - Can get variable sensitivity



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

Graphical Logical Devices

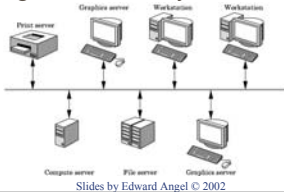
- Graphical input is more varied than input to standard programs which is usually numbers, characters, or bits
- Two older APIs (GKS, PHIGS) defined six types of logical input
 - **Locator**: return a position
 - **Pick**: return ID of an object
 - **Keyboard**: return strings of characters
 - **Stroke**: return array of positions
 - **Valuator**: return floating point number
 - **Choice**: return one of n items



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

Input Modes

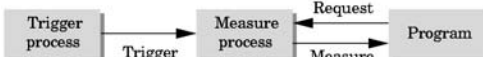
- Input devices contain a *trigger* which can be used to send a signal to the operating system
 - Button on mouse
 - Pressing or releasing a key
- When triggered, input devices return information (their *measure*) to the system
 - Mouse returns position information
 - Keyboard returns ASCII code



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

GLUT callbacks

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

- `glutDisplayFunc`
- `glutMouseFunc`
- `glutReshapeFunc`
- `glutKeyFunc`
- `glutIdleFunc`
- `glutMotionFunc`, `glutPassiveMotionFunc`



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002