# EECE 478

Graphics Programming



## Program Structure

- Setup environment
  - Relationship between graphics and window system
  - Create baseline graphics state
- Refresh function(s)
  - Every frame is redrawn from scratch
- Input functions
  - respond to user interaction



## GLUT Structure

- Initialize OpenGL/WS interface
- Manage window(s)
  - Size and position
  - Rendering parameters
  - Set callbacks
- Manage refresh
  - Signal when scene has changed

## GLUT Callbacks

```
void display ();
glutDisplayFunc (display);
```

- called when screen needs to be redrawn
- signalled with `glutPostRedisplay()`

```
void key (char key, int x, int y);
glutKeyboardFunc (key);
```

- called when key struck
- location of mouse in (x,y)



## GLUT Callbacks

```
void mouse (int btn, int st, int x, int y);
glutMouseFunc (mouse);
```

- called when mouse clicked
- `btn` identifies button
- `st` describes button state

```
void idle ();
glutIdleFunc (idle);
```

- called when no other events ready



## OpenGL API Structure

- Primitives
- Attributes
- Viewing
- Transformations
- Inquiries

## Classes of Functions

- Data functions
  - Provide data for pipeline
  - *Primitives*
- State change functions
  - Modify parameters for pipeline stages
  - Modify structure of pipeline

*Non-primitives active until changed!*



## Primitives

- Define geometry
  - vertices and vertex arrays
  - polygons
  - display lists
- Define bitmaps
  - text
  - texture



## Defining Vertices

- glVertex*
  - define a point in current coordinate system
  - variations for dimensionality
  - variations for argument type (int, float, …)
  - variations for arrays

## Defining Polygons

- glBegin/glEnd pairs
  - enclose vertex sequences
  - glBegin argument defines primitive type
  - order of vertices determines normal
  - primitive polygons assumed convex



## Primitive Types

- GL_POINTS
  - vertices represent points
- GL_LINES
  - vertex pairs are line segments
- GL_LINE_STRIP, GL_LINE_LOOP
  - vertices form continuous line

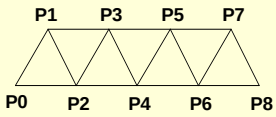


## Polygon Types

- GL_POLYGON
  - vertices form n-sided polygon
  - decomposed into triangles
  - can be filled or outlined (glPolygonMode)
- GL_TRIANGLES
  - vertex triples are triangles
- GL_QUADS
  - vertex quadruples are quadrilaterals

## Polygon Strips

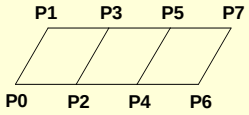
- GL_TRIANGLE_STRIP
  - vertices meshed into triangles






## Polygon Strips

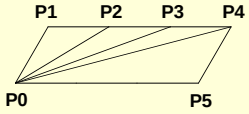
- GL_QUAD_STRIP
  - vertices meshed into quadrilaterals






## Triangle Fan

- GL_TRIANGLE_FAN
  - vertices form triangles from P0



## Attributes

- Properties define how a primitive will be rendered
  - Associated with vertices or primitives
  - Each primitive is sensitive to subset of attributes
  - Once defined, apply to *all* subsequent primitives until changed
  - *Global* state, so must be very careful



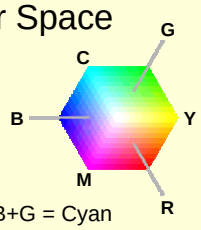
## Attributes

- Points
  - Color, point size
- Lines
  - Color, thickness, dashed type
- Filled polygons
  - Color, material, texture patterns, …



## RGB Color Space

RGB
- (Red, Green, Blue)
- Additive color space
- 1.0 is max intensity




glColor3f (0.0, 1.0, 1.0) = B+G = Cyan
glColor3f (1.0, 0.0, 1.0) = R+B = Magenta
glColor3f (1.0, 1.0, 0.0) = R+G = Yellow

## Setting Color

glColor*
- sets vertex color

glClearColor
- sets color to clear screen



## Other Attributes

glPointSize
- pixel diameter of drawn points

glLineWidth
- pixel width of drawn lines



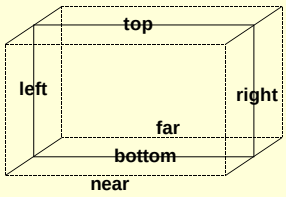
## 2D Viewing

- Orthographic projection
  - special case of perspective projection
  - no foreshortening (division)

## glOrtho

glOrtho(*left,right, bottom,top, near,far*)

– creates orthographic view volume






## Matrix modes

- Projection matrix
  - controls final projection to screen
  - glMatrixMode (GL_PROJECTION)
- Model-view matrix
  - transforms from object coordinates to screen coordinates
  - combines *model* and *view* matrices
  - glMatrixMode (GL_MODELVIEW)



## Sierpinski Example: ex1.c

- Full GLUT/OpenGL program
  - Recursively draws Sierpinski gasket
- Includes
  - 2D (and 3D) rendering
  - Drawing (various primitives and modes)
  - Interaction via keyboard
  - Animation

## main(): GLUT Initialization

- Initialize GLUT
- Define 512x512 window
  - RGB colorspace
  - double buffering and a depth buffer

```
glutInit (&argc, argv);
glutInitWindowSize (512, 512);
glutInitDisplayMode (GLUT_DOUBLE
                     |GLUT_RGB|GLUT_DEPTH);
glutCreateWindow ("EECE 478 - Example 1");
```



## main(): Callbacks

- Provide event handlers
  - Redraw screen and animate
  - Handle window resizing
  - Deal with input events

```
glutDisplayFunc (display);
glutIdleFunc (idle);
glutReshapeFunc (reshape);
glutKeyboardFunc (key);
glutSpecialFunc (special);
glutMouseFunc (mouse);
```



## initDisplay(): OpenGL Setup

- Establish background (clear) color
- Enable depth testing pipeline stage

```
glClearColor (0.2, 0.2, 0.4, 0.0);
glEnable (GL_DEPTH_TEST);
```

## initDisplay(): Projection

- Switch to projection matrix
- Use orthographic projection
  - screen box x:[-1,1], y:[-1,1]
- Switch back to model-view matrix

```
// Setup orthographic projection
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho (-1.0, 1.0, -1.0, 1.0, -1.5, 1.5);
glMatrixMode (GL_MODELVIEW);
```



## initDisplay(): Attributes

- Point diameter 2 pixels
- Line thickness 2 pixels

```
glPointSize (2.0);
glLineWidth (2.0);
```



## main(): Main Loop

- Main loop calls event handlers

```
initDisplay ();
glutMainLoop ();
```

## params: Draw control

- *Global* variable
  - all drawing parameters in one place
  - *necessary evil with GLUT*
- Easily change draw function (displayFunc)

```
struct {
  int height;       // Depth of recursion
  GLfloat theta;    // Rotation angle
  int rotating;     // Animating rotation?
  GLuint drawMode;  // Drawing primitives used
  GLuint polyMode;  // Polygon draw mode
  void (*displayFunc)(GLuint drawMode, int height);
} params = {
  5, 0.0, 0, GL_POINTS, GL_FILL, sierpinski2D };
```



## display(): Redraw

- Clear screen
  - Color buffer
  - Depth buffer
- Establish default draw color (white)

```
glClear (GL_COLOR_BUFFER_BIT
         | GL_DEPTH_BUFFER_BIT);
glColor3f (1.0, 1.0, 1.0);
```



## display(): Redraw gasket

- Isolate model-view matrix modification
  - Push/pop active matrix
- Rotate about vertical (nearly) axis
- Display gasket

```
glPushMatrix ();
glRotatef (params.theta, 0.2, 1.0, 0.0);
glPolygonMode (GL_FRONT_AND_BACK,
               params.polyMode);
(*params.displayFunc) (params.drawMode,
                       params.height);
glPopMatrix ();
glutSwapBuffers ();
```

## sierpinski2D(drawMode, height)

- Create context for primitive
  - GL_TRIANGLES or GL_POINTS
- Polygon mode GL_FILL or GL_LINE
- Start recursion

```
glBegin (drawMode);
sierpinski (height, 0.0, 1.0,
            -1.0, -1.0,
             1.0, -1.0);
glEnd ();
```



## sierpinski(height): Bottom

- If at leaves of recursion
  - Draw a triangle
- Otherwise…

```
if (height == 0) {
  drawTri (x0, y0, x1, y1, x2, y2);
} else {
  ..
}
```



## sierpinski(): Recursion

- Otherwise…
  - Calculate centers of sides
  - Reduce height
  - Recurse on three subtriangles

```
float mx0 = (x0+x1)/2, my0 = (y0+y1)/2;
float mx1 = (x1+x2)/2, my1 = (y1+y2)/2;
float mx2 = (x2+x0)/2, my2 = (y2+y0)/2;
--height;
sierpinski (height,  x0,  y0, mx0, my0, mx2, my2);
sierpinski (height, mx0, my0,  x1,  y1, mx1, my1);
sierpinski (height, mx2, my2, mx1, my1,  x2,  y2);
```

## drawTri(): Drawing

- Just three 2D vertices
  - z is assumed 0.0
- *N.B. We are inside a glBegin(prim)*
  - From sierpinski2D()

```
glVertex2f (x0, y0);
glVertex2f (x1, y1);
glVertex2f (x2, y2);
```



## sierpinski3D()

- Left as an exercise…

```
glBegin (drawMode);
GLfloat p0[] = { 0,  1,  0};
GLfloat p1[] = {-1, -1,  1};
GLfloat p2[] = { 1, -1,  1};
GLfloat p3[] = { 0, -1, -1};
tetra (height, p0, p1, p2, p3)
glEnd ();
```



## key(key,px,py): Keyboard

- Signalled when key struck
  - key = keyboard character
  - (px,py) = location of mouse in pixels
- Exit program if ESC

```
switch (key) {
..
case '\e':
  exit (0);
  break;
}
glutPostRedisplay (); // Signal redisplay!
```

## key(): Draw modes

- Change *global* variable params
  - 'p' ⇒ GL_POINTS
  - 't' ⇒ GL_TRIANGLES & GL_FILL
  - 'l' ⇒ GL_TRIANGLES & GL_LINE

```
case 'p':// Draw point
 params.drawMode = GL_POINTS;
 break;
case 't':// Draw triangles
 params.drawMode = GL_TRIANGL
 params.polyMode = GL_FILL;
 break;
```

```
case 'l':// Draw lines
 params.drawMode = GL_TRIANGLES;
 params.polyMode = GL_LINE;
 break;
```



## key(): Animation

- Interact with idle() callback
  - rotating = TRUE ⇒ spinning motion
  - spacebar toggles motion state
  - 'r' stops and resets to unrotated

```
case ' ':// Toggle rotation state
 params.rotating = !params.rotating;
 break;
case 'r':// Reset (and stop) rotation
 params.rotating = 0;
 params.theta = 0;
 break;
```



## idle(): Animation

- Called whenever no other events
  - only spin when rotate non-zero
  - rotate 2º more every cycle (keep small)
  - ask to be redisplayed

```
if (params.rotating) {
 params.theta += 2;
 if (params.theta > 360)
  params.theta -= 360;
 glutPostRedisplay ();
}
```

# Assignment #1

- Start with newpaint.c program from Book CDROM (§3.8)
- Add features
  - Highlight “button” when active
  - Stay in primitive mode until changed
    - e.g. every set of 3 points makes new triangle
  - Rubber banding with XOR or overlay visual