Announcements

• Computing Lab: Wean 5336
• Accounts and ID’s should work. Send email to me or the TA’s if they don’t.

• Assignment #1 is up (with starter code).

• Today more on OpenGL
  Read Angel up to section 3.6
• Thursday more on transformations
  Pre-read Angel chapter 4
Assignment #1

Height Fields
Height Fields

• Why?
  Get started with OpenGL
  Some room for creativity

• Where?
  Wean 5336 or your machine at your risk!

• How?
  Cross-realm authentication via andrew
  Send problems to me or to the TA’s (soon)
  Make sure that you made directory with correct permissions—most common problem
  Start early!
Height Fields

- What?

- When? -- Due midnight September 12th
More on OpenGL
Recap of Graphics Functions

- Primitive functions
- Attribute functions
- Transformation functions
- Viewing functions
- Input functions
- Control functions

Diagram:

Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
Outline of Today’s Class

• A bit more on primitives
• Color; a more complicated example
• Client/Server Model
• Callbacks
• Double Buffering
• Hidden Surface Removal
• Another example
Polygon Restrictions

- OpenGL Polygons must be **simple**
- OpenGL Polygons must be **convex**

(a) simple, but not convex

(b) non-simple
Why Polygon Restrictions?

• Non-convex and non-simple polygons are expensive to process and render
• Convexity and simplicity is expensive to test
• Better to fix polygons as a pre-processing step
• Some tools in GLU for decomposing complex polygons (tessellations)
• Behavior of OpenGL implementation on disallowed polygons is “undefined”
• Triangles are most efficient in hardware
Attributes

• Part of the state of the graphics pipeline
• Set **BEFORE** primitives are drawn
• **Remain in effect!**
• Examples:
  - Color, including transparency
  - Reflection properties
  - Shading properties
Physics of Color

• Can see only tiny piece of the spectrum
• Screens can show even less
Color Filters

- Eye can perceive only 3 basic colors
- Computer screens designed accordingly
- Many visible colors still not reproducible (high contrast)
Color Spaces

• RGB (Red, Green, Blue)
  Convenient for display
  Can be unintuitive (3 floats in OpenGL)
• HSV (Hue, Saturation, Value)
  Hue: what color
  Saturation: how far away from gray
  Value: how bright
• Others for film, video, and printing
• Getting the colors right is a time consuming problem in the industry
Example: Drawing a shaded polygon

• More complicated example than last time
• Initialization: the “main” function

```c
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    ...
GLUT Callbacks

- Window system independent interaction
- glutMainLoop processes events

```c
...
    glutDisplayFunc (display);
    glutReshapeFunc (reshape);
    glutKeyboardFunc (keyboard);
    glutMainLoop(); /* Finally displays window */
    return 0;
}
Initializing Attributes

• Separate in “init” function

```c
void init(void) {
    glClearColor (0.0, 0.0, 0.0, 0.0);
    /* glShadeModel (GL_FLAT); */
    glShadeModel (GL_SMOOTH);
}
```
The Display Callback

- Handles display events
- Install with glutDisplayFunc(display)

```c
void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT); /* clear buffer */
    triangle (); /* draw triangle */
    glFlush (); /* force display */
}
```
void triangle(void) {
    glBegin (GL_TRIANGLES);
    glColor3f (1.0, 0.0, 0.0); /* red */
    glVertex2f (5.0, 5.0);
    glColor3f (0.0, 1.0, 0.0); /* green */
    glVertex2f (25.0, 5.0);
    glColor3f (0.0, 0.0, 1.0); /* blue */
    glVertex2f (5.0, 25.0);
    glEnd();
}
The Image

- Color of last vertex with flat shading

```plaintext
glShadeModel(GL_FLAT)  
glShadeModel(GL_SMOOTH)
```
Client/Server Model

- Graphics hardware and caching

- Important for efficiency
- Need to be aware where data are stored
- Examples: vertex arrays, display lists
Display Lists

• Encapsulate a sequence of drawing commands
• Optimize and store on server
• *Retained mode* (instead of *immediate mode*)

```c
GLuint listName = glGenLists(1);  /* new name */
glNewList (listName, GL_COMPILE);  /* new list */
    glColor3f(1.0, 0.0, 1.0);
    glBegin(GL_TRIANGLES);
        glVertex3f(0.0, 0.0, 0.0);
        ...  
    glEnd();
    glTranslatef(1.5, 0.0, 0.0);  /* offset next object */
    glEndList();
glCallList(listName);  /* draw one */
```
Display Lists Details

- Useful for sequences of transformations
- Important for complex surfaces
- Hierarchical display lists supported
- Display lists cannot be changed
- Display lists can be replaced
- Not necessary in first assignment

- (Useful for programmable shaders)
Main Event Loop

• Standard technique for interaction
• Mediates between client and window system
• Main loop processes events
• Dispatch to functions specified by client
• Callbacks also common in operating systems
Types of Callbacks

- **Display( )** when window must be drawn
- **Idle( )** when no other events to be handled
- **Keyboard(unsigned char key, int x, int y)** key events
- **Menu (...)** after selection from menu
- **Mouse (int button, int state, int x, int y)** mouse events
- **Motion (...)** mouse movement
- **Reshape (int w, int h)** window resize
- Any callback can be **NULL**
Double Buffering

- Screen refreshing technique
- Common refresh rate: 60-100 Hz
- Flicker if drawing overlaps screen refresh
- Problem during animation
- Example (cube_single.c)
- Solution: *use two frame buffers*
  - Draw into one buffer
  - Swap and display, while drawing other buffer
- Desirable frame rate $\geq 30$ fps
  ($fps = \text{frames/second}$)
Enabling Modes

• One example of many
  • glutInitDisplayMode (GLUT_SINGLE);
  • glutInitDisplayMode (GLUT_DOUBLE);
  • glutSwapBuffers();
Hidden Surface Removal

- What is visible after clipping and projection?
- Object-space vs image-space approaches
- Object space: depth sort (Painter’s algorithm)
- Image space: ray cast (z-buffer algorithm)
- Related: back-face culling

We’ll get back to this later in the semester in much more detail!
Object-Space Approach

- Consider pairs of objects

![Diagram](a) ![Diagram](b) ![Diagram](c) ![Diagram](d)

- Complexity $O(k^2)$ where $k =$ # of objects
- Painter’s algorithm: render back-to-front
- “Paint” over invisible polygons
- How to sort and how to test overlap?
Depth Sorting

- First, sort by furthest distance $z$ from viewer
- If minimum depth of A is greater than maximum depth of B, A can be drawn before B
- If either $x$ or $y$ extents do not overlap, A and B can be drawn independently
Some Difficult Cases

• Sometimes cannot sort polygons!

Cyclic overlap

One solution: compute intersections and subdivide \(\rightarrow\) clip polygons against each other

Piercing Polygons

• Do while rasterizing (difficult in object space)
Painter’s Algorithm Assessment

• **Strengths**
  Simple (most of the time)
  Handles transparency well
  Sometimes, no need to sort (e.g., heightfield)

• **Weaknesses**
  Clumsy when geometry is complex
  Sorting can be expensive

• **Usage**
  OpenGL (by default)
  PostScript interpreters
Image-Space Approach

• Raycasting: intersect ray with polygons

• $O(k)$ worst case (often better)
  where $k=\#$ of objects
The z-Buffer Algorithm

- z-buffer with depth value \( z \) for each pixel
- Before writing a pixel into framebuffer
  Compute distance \( z \) of pixel origin from viewer
  If closer write and update z-buffer, otherwise discard
z-Buffer Algorithm Assessment

• **Strengths**
  - Simple (no sorting or splitting)
  - Independent of geometric primitives

• **Weaknesses**
  - Memory intensive (but memory is cheap now)
  - Tricky to handle transparency and blending
  - Depth-ordering artifacts for near values
  - Render some wasted polygons

• **Usage**
  - OpenGL when enabled
Depth Buffer in OpenGL

• `glutInitDisplayMode(GLUT_DEPTH);`
• `glEnable(GL_DEPTH_TEST);`
• `glClear(GL_DEPTH_BUFFER_BIT);`
• Remember all of these!
Specifying the Viewing Volume

• Clip everything not in viewing volume

• Separate matrices for transformation and projection

```c
glMatrixMode (GL_PROJECTION)
glLoadIdentity();

... Set viewing volume ...

glMatrixMode (GL_MODELVIEW)
```
Parallel Viewing

- Orthographic projection
- Camera points in *negative* z direction
- `glOrtho(xmin, xmax, ymin, ymax, near, far)`

- Clarification: $2z_{\text{min}} = -\text{near}$, $2z_{\text{max}} = -\text{far}$
Perspective Viewing

• Slightly more complex
  • `glFrustum(xmin, xmax, ymin, ymax, near, far)`

• Clarification: \(2z_{\text{min}} = -\text{near}, \quad 2z_{\text{max}} = -\text{far}\)
Simple Transformations

- Rotate by given angle (in degrees) about ray from origin through \((x, y, z)\)
  
  \[
  \text{glRotate}(\text{angle}, x, y, z);
  \]

- Translate by the given \(x, y,\) and \(z\) values
  
  \[
  \text{glTranslate}(x, y, z);
  \]

- Scale with a factor in the \(x, y,\) and \(z\) direction
  
  \[
  \text{glScale}(x, y, z);
  \]
Example: Rotating Color Cube

• Draw a color cube
• Rotate it about x, y, or z axis, depending on left, middle or right mouse click
• Stop when space bar is pressed
• Quit when q or Q is pressed

• See Angel: Sec. 4.4 and CD program
• Demo
Step 1: Defining the Vertices

- Use parallel arrays for vertices and colors

```c
/* vertices of cube about the origin */
GLfloat vertices[8][3] =
    {{-1.0, -1.0, -1.0}, {1.0, -1.0, -1.0},
     {1.0, 1.0, -1.0}, {-1.0, 1.0, -1.0},
     {-1.0, -1.0, 1.0}, {1.0, -1.0, 1.0},
     {1.0, 1.0, 1.0}, {-1.0, 1.0, 1.0}};

/* colors to be assigned to edges */
GLfloat colors[8][3] =
    {{0.0, 0.0, 0.0}, {1.0, 0.0, 0.0},
     {1.0, 1.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0},
     {1.0, 0.0, 1.0}, {1.0, 1.0, 1.0}, {0.0, 1.0, 1.0}};
```
Step 2: Set Up

• Enable depth testing and double buffering

```c
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    /* double buffering for smooth animation */
    glutInitDisplayMode
        (GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGB);
    /* window creation and callbacks here */
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return(0);
}
```
Step 3: Install Callbacks

• Create window and set callbacks

```c
glutInitWindowSize(500, 500);
glutCreateWindow("cube");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spinCube);
glutMouseFunc(mouse);
glutKeyboardFunc(keyboard);
```
Step 4: Reshape Callback

- Enclose cube, preserve aspect ratio

```c
void myReshape(int w, int h)
{
    GLfloat aspect = (GLfloat) w / (GLfloat) h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h) /* aspect <= 1 */
        glOrtho(-2.0, 2.0, -2.0/aspect, 2.0/aspect, -10.0, 10.0);
    else /* aspect > 1 */
        glOrtho(-2.0*aspect, 2.0*aspect, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
}
```
Step 5: Display Callback

- Clear, rotate, draw, flush, swap

```c
GLfloat theta[3] = {0.0, 0.0, 0.0};

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    colorcube();
    glFlush();
    glutSwapBuffers();
}
```
Step 6: Drawing Faces

- Call \texttt{face(a,b,c,d)} with vertex index
- Orient consistently

```c
void colorcube(void) {
    face(0,3,2,1);
    face(2,3,7,6);
    face(0,4,7,3);
    face(1,2,6,5);
    face(4,5,6,7);
    face(0,1,5,4);
}
```
Step 7: Drawing a Face

- Use vector form of primitives and attributes

```c
void face(int a, int b, int c, int d) {
    glBegin(GL_POLYGON);
    glColor3fv (colors[a]);
    glVertex3fv(vertices[a]);
    glColor3fv (colors[b]);
    glVertex3fv(vertices[b]);
    glColor3fv (colors[c]);
    glVertex3fv(vertices[c]);
    glColor3fv (colors[d]);
    glVertex3fv(vertices[d]);
    glEnd();
}
```
Step 8: Animation

• Set idle callback: `spinCube()`

```c
GLfloat delta = 2.0;
GLint axis = 2;
void spinCube()
{
  /* spin cube delta degrees about selected axis */
  theta[axis] += delta;
  if (theta[axis] > 360.0) theta[axis] -= 360.0;

  /* display result */
  glutPostRedisplay();
}
```
Step 9: Change Axis of Rotation

- Mouse callback

```c
void mouse(int btn, int state, int x, int y) {
    if (btn == GLUT_LEFT_BUTTON
        && state == GLUT_DOWN) axis = 0;
    if (btn == GLUT_MIDDLE_BUTTON
        && state == GLUT_DOWN) axis = 1;
    if (btn == GLUT_RIGHT_BUTTON
        && state == GLUT_DOWN) axis = 2;
}
```
Step 10: Toggle Rotation or Exit

- Keyboard callback

```c
void keyboard(unsigned char key, int x, int y) {
    if (key=='q' || key == 'Q') exit(0);
    if (key==' ') {stop = !stop;};
    if (stop)
        glutIdleFunc(NULL);
    else
        glutIdleFunc(spinCube);
}
```
Announcements

• Computing Lab: Wean 5336
• Accounts and ID’s should work. Send email to me or the TA’s if they don’t.

• Assignment #1 is up (with starter code)
  Start early!

• Thursday more on transformations
  Pre-read Angel chapter 4