CSCI 480 Computer Graphics
Lecture 3

Interaction

Client/Server Model
Callbacks
Double Buffering
Hidden Surface Removal
Simple Transformations

January 19, 2011
Jernej Barbic
University of Southern California
http://www-bcf.usc.edu/~jbarbic/cs480-s11/

Triangles (Clarification)

- Can be any shape or size
- Well-shaped triangles have advantages for numerical simulation
- Little difference with basic OpenGL rendering

Surface Orientation (Clarification)

- Right-hand rule
- Triangle strip drawn 0-1-2, 2-1-3, 2-3-4, etc.
- All triangles face same direction (here: back)
- Similarly for quad strips 0-1-3-2, 2-3-5-4, etc.
- For closed surfaces can discard back faces:
  glEnable(GL_CULL_FACE);
  glCullFace(GL_BACK); /* do not draw back faces */

Choice of Programming Language

- OpenGL lives close to the hardware
- OpenGL is not object-oriented
- OpenGL is not functional
- Use C to expose and exploit low-level details
- Use C++, Java, ... for toolkits
- Support for C and C++ in assignments

Client/Server Model

- Graphics hardware and caching
  "Client"  "Server"

  CPU  GPU  Display (CRT)

- 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 (GPU)

  "Client"  "Server"

  CPU  bus  GPU  Display (CRT)

  Store geometry, colors, lighting properties of objects on the GPU
The CPU-GPU bus

AGP, PCI, PCI Express
Fast, but limited bandwidth
Possible, but very slow

Display Lists
- Encapsulate a sequence of drawing commands
- Optimize and store on server (GPU)

```c
GLuint listName = glGenLists(1); /* new list name */
glNewList (listName, GL_COMPILE); /* new list */
glColor3f(1.0, 0.0, 1.0);
gBegin(GL_TRIANGLES);
gVertex3f(0.0, 0.0, 0.0);
...
gEnd();
gEndList(); /* at this point, OpenGL compiles the list */
gCallList(listName); /* draw the object */
```

Display Lists Details
- Very useful with complex objects that are redrawn often (e.g., with transformations)
- Another example: fonts (2D or 3D)
- Display lists can call other display lists
- Display lists cannot be changed
- Display lists can be erased / replaced
- Not necessary in first assignment
- Display lists are now deprecated in OpenGL
- For complex usage, use the VertexBufferObject (VBO) extension

Vertex Arrays
- Draw cube with 6*4=24 or with 8 vertices?
- Expense in drawing and transformation
- Strips help to some extent
- Vertex arrays provide general solution
- Advanced (since OpenGL 1.2)
  - Define (transmit) array of vertices, colors, normals
  - Draw using index into array(s)
  - Vertex sharing for efficient operations
- Not needed for first assignment

Outline
- Client/Server Model
- Callbacks
- Double Buffering
- Hidden Surface Removal
- Simple Transformations
- Example

Main Event Loop
- Standard technique for interaction
- Main loop processes events
- Dispatch to functions specified by client
- Callbacks also common in operating systems
- Poor man’s functional programming
- Mediates between client and window system
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
- Menu (...): after selection from menu
- Mouse (int button, int state, int x, int y): mouse
- Motion (...): mouse movement
- Reshape (int w, int h): window resize
- Any callback can be NULL

GLUT Program with Callbacks

```
START
  Initialization
  Main event loop
    Idle
    Display
    Keyboard...
    Menu...
    Mouse...
    Motion...
    Reshape...
END
```

Outline

- Client/Server Model
- Callbacks
- Double Buffering
- Hidden Surface Removal
- Simple Transformations
- Example

Screen Refresh

- Common: 60-100 Hz
- Flicker if drawing overlaps screen refresh
- Problem during animation
- Solution: use two separate frame buffers:
  - Draw into one buffer
  - Swap and display, while drawing into other buffer
- Desirable frame rate >= 30 fps (frames/second)

Enabling Single/Double Buffering

- glutInitDisplayMode (GLUT_SINGLE);
- glutInitDisplayMode (GLUT_DOUBLE);

  Single buffering:
  Must call glFinish () at the end of Display()

  Double buffering:
  Must call glutSwapBuffers() at the end of Display()

- If something in OpenGL has no effect or does not work, check the modes in glutInitDisplayMode

Outline

- Client/Server Model
- Callbacks
- Double Buffering
- Hidden Surface Removal
- Simple Transformations
- Example
Hidden Surface Removal

- Classic problem of computer graphics
- 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

Object-Space Approach

- Consider objects pairwise

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!
- One solution: compute intersections
- 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
  - PostScript interpreters
  - OpenGL: must implement Painter’s Algorithm manually

Image-Space Approach

- Raycasting: intersect ray with polygons
  - O(k) worst case (often better)
  - Images can be more jagged
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

Depth Buffer in OpenGL

- glutInitDisplayMode(GLUT_DEPTH);
- glEnable (GL_DEPTH_TEST);
- glClear (GL_DEPTH_BUFFER_BIT);
- Remember all of these!
- Some “tricks” use z-buffer read-only

Specifying the Viewing Volume

- Clip everything not in viewing volume
- Separate matrices for transformation and projection

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
... Set viewing volume ...
glMatrixMode (GL_MODELVIEW);
```

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
- Usage
  - z-Buffering comes standard with OpenGL; disabled by default; must be enabled

Outline

- Client/Server Model
- Callbacks
- Double Buffering
- Hidden Surface Removal
- Simple Transformations
- Example

Parallel Viewing

- Orthographic projection
- Camera points in negative z direction
- glOrtho(xmin, xmax, ymin, ymax, near, far)
Perspective Viewing
• Slightly more complex
• glFrustum(xmin, xmax, ymin, ymax, near, far)

Simple Transformations
• Rotate by given angle (in degrees) about ray from origin through (x, y, z)
  glRotate(fd)(angle, x, y, z);
• Translate by the given x, y, and z values
  glTranslate(fd)(x, y, z);
• Scale with a factor in the x, y, and z direction
  glScale(fd)(x, y, z);

Outline
• Client/Server Model
• Callbacks
• Double Buffering
• Hidden Surface Removal
• Simple Transformations
• Example

Example: Rotating Color Cube
• Adapted from [Angel, Ch. 4]
• Problem:
  – 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

Step 1: Defining the Vertices
• Use parallel arrays for vertices and colors
  /* 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 z-buffer and Double Buffering
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode
    (GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGB);
  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

- Set projection and viewport, 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 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

  ```c
  GLfloat delta = 2.0;
  GLint axis = 2;
  void spinCube()
  {
    /* spin the cube delta degrees about selected axis */
    theta[axis] += delta;
    if (theta[axis] > 360.0) theta[axis] -= 360.0;
    /* display result (do not forget this!) */
    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 == 's')
    stop = !stop;
  if (stop)
    glutIdleFunc(NULL);
  else
    glutIdleFunc(spinCube);
}
```

Summary

- Client/Server Model
- Callbacks
- Double Buffering
- Hidden Surface Removal
- Simple Transformations
- Example

Announcements

- Assignment 1 has been posted
- Microsoft Visual Studio (Windows) access enabled via Microsoft’s MSDN
- Please start early
- Check web page for instructions