CSCI 420 Computer Graphics
Lecture 3

Interaction

Client/Server Model
Callbacks
Double Buffering
Hidden Surface Removal
Simple Transformations
(Angel Ch. 2)

Jernej Barbic
University of Southern California

Triangles (Clarification)

- Can be any shape or size
- Well-shaped triangles have advantages for numerical simulation
- Shape quality makes little difference for basic OpenGL rendering

Choice of Programming Language

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

Client/Server Model

- Graphics hardware and caching
- Important for efficiency
- Need to be aware where data are stored
- Graphics driver code is on the CPU
- Rendering resources (buffers, shaders, textures, etc.) are on the GPU

The CPU-GPU bus

AGP, PCI, PCI Express
Fast, but limited bandwidth

Can also read back

Buffer Objects

- Store rendering data: vertex positions, normals, texture coordinates, colors, vertex indices, etc.
- Optimize and store on server (GPU)
Vertex Buffer Objects
- Caches vertex geometric data: positions, normals, texture coordinates, colors
- Optimize and store on server (GPU)
- Required for core OpenGL profile

```
/* vertices of the quad (will form two triangles; rendered via GL_TRIANGLE_STRIP) */
float positions[4][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}};
/* colors to be assigned to vertices */
float colors[4][4] =
    {{0.0, 0.0, 0.0, 1.0}, {1.0, 0.0, 0.0, 1.0},
     {1.0, 1.0, 0.0, 1.0}, {0.0, 1.0, 0.0, 1.0}};
```

Vertex Buffer Object: Initialization
```
GLuint buffer;

void initVBO()
{
    glGenBuffers(1, &buffer);
    glBindBuffer(GL_ARRAY_BUFFER, buffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(positions) +
                 sizeof(colors), NULL, GL_STATIC_DRAW); // init buffer's size, but don't assign any data to it

    // upload position data
    glBufferSubData(GL_ARRAY_BUFFER, 0,
                    sizeof(positions), positions);
    // upload color data
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(positions),
                    sizeof(colors), colors);
}
```

Old technology: Display Lists (compatibility profile only)
- Cache a sequence of drawing commands
- 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
- Display lists are now deprecated in OpenGL
- Replaced with VBOs

```
GLuint listName = glGenLists(1);    /* new list name */
gNewList(listName, GL_COMPILE); /* new list */
gColor3f(1.0, 0.0, 1.0);
 glBegin(GL_TRIANGLES);
      glVertex3f(0.0, 0.0, 0.0);
      glVertex3f(1.0, 0.0, 0.0);
      glVertex3f(0.0, 1.0, 0.0);
 glEnd();
 glEndList(); /* at this point, OpenGL compiles the list */
gCallList(listName); /* draw the object */
```

Element Arrays
- Draw cube with 6*2^3=36 or with 8 vertices?
- Expense in drawing and transformation
- Triangle strips help to some extent
- Element arrays provide general solution
- Define (transmit) array of vertices, colors, normals
- Draw using index into array(s):
  // (must first set up the GL_ELEMENT_ARRAY_BUFFER) ...
  glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
- Vertex sharing for efficient operations
- Extra credit for first assignment

Display Lists
- Cache a sequence of drawing commands
- Optimize and store on server (GPU)
- Another example: fonts (2D or 3D)
- Display lists cannot be changed
- Display lists can be erased / replaced
- Display lists are now deprecated in OpenGL
- Replaced with VBOs

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

Main Event Loop
- Standard technique for interaction (GLUT, Qt, wxWidgets, ...)
- Main loop processes events
- Dispatch to functions specified by client
- Callbacks also common in operating systems
- "Poor man’s functional programming"

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 pressed
- Menu (...): after selection from menu
- Mouse (int button, int state, int x, int y): mouse button
- Motion (...): mouse movement
- Reshape (int w, int h): window resize
- Any callback can be NULL

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()
- Must call glutPostRedisplay() at the end of Idle()
- 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: z-buffer algorithm
• Related: back-face culling

Object-Space Approach

• Consider objects pairwise

○ 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!

○ One solution: compute intersections & subdivide
○ 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: not supported
  (must implement Painter’s Algorithm manually)
Image-space approach

3D geometry
Depth image
darker color is closer


Depth sensor camera

Image-Space Approach

- Raycasting: intersect ray with polygons

- $O(k)$ worst case (often better)
- Images can be more jagged (need anti-aliasing)

The z-Buffer Algorithm

- z-buffer stores depth values $z$ for each pixel
- Before writing a pixel into framebuffer:
  - Compute distance $z$ of pixel 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
- Usage
  - z-Buffering comes standard with OpenGL; disabled by default; must be enabled

Depth Buffer in OpenGL

- `glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);`
- `glEnable(GL_DEPTH_TEST);`
- Inside Display():
  - `glClear(GL_DEPTH_BUFFER_BIT);`
- Remember all of these!
- Some “tricks” use z-buffer in read-only mode
Note for Mac computers

Must use the GLUT_3_2_CORE_PROFILE flag to use the core profile:

```c
glutInitDisplayMode(GLUT_3_2_CORE_PROFILE | GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
```

Outline

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

Specifying the Viewing Volume: Compatibility Mode

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

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

Specifying the Viewing Volume: Core Profile

- Clip everything not in viewing volume
- Set the 4x4 projection matrix manually (or via our provided “openGLMatrix” library)

Parallel Viewing

- Orthographic projection
- Camera points in negative z direction
- Compatibility profile:
  ```c
  glOrtho(xmin, xmax, ymin, ymax, near, far)
  ```
- Core profile: set the 4x4 matrix manually (or via our provided “openGLMatrix” library)

Perspective Viewing

- Slightly more complex
- Compatibility profile:
  ```c
  glFrustum(left, right, bottom, top, near, far)
  ```
- Core profile: set the 4x4 matrix manually (or via our provided “openGLMatrix” library)
Simple Transformations: Compatibility Profile

- Rotate by given angle (in degrees) about axis given by \((x, y, z)\)
  
  \[ \text{glRotate}(\text{fd}, x, y, z); \]

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

- Scale with a factor in the \(x\), \(y\), and \(z\) direction
  
  \[ \text{glScale}(\text{fd}, x, y, z); \]

---

Simple Transformations: Core Profile

- Rotate by given angle (in degrees) about axis given by \((x, y, z)\)

- Translate by the given \(x\), \(y\), and \(z\) values

- Scale with a factor in the \(x\), \(y\), and \(z\) direction

  Create these 4x4 matrices manually
  
  (or via our provided "openGLMatrix" library)

  (Lecture: "Transformations")

---

Outline

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

---

Example: Rotating Colored Quad

- Problem:
  - Draw a colored quad (two triangles)
  - Rotate it about \(x\), \(y\), or \(z\) axis, depending on left, middle or right mouse click
  - Stop when the space bar is pressed
  - Quit when q or Q is pressed

---

Step 1: Defining the Vertices

- Use separate arrays for vertices and colors

  "vertices of the quad (will form two triangles; rendered via GL_TRIANGLE_STRIP) 
  float positions[4][3] = 
  \[ \begin{array}{c}
    (-1.0, -1.0, -1.0), \\
    (1.0, -1.0, -1.0), \\
    (-1.0, 1.0, -1.0), \\
    (1.0, 1.0, -1.0) \\
  \end{array} \]

  "colors to be assigned to vertices 
  float colors[4][4] = 
  \[ \begin{array}{c}
    (0.0, 0.0, 0.0, 1.0), \\
    (1.0, 0.0, 0.0, 1.0), \\
    (1.0, 0.0, 1.0, 1.0), \\
    (0.0, 1.0, 0.0, 1.0) \\
  \end{array} \]

  // black, red, yellow, green

---

Step 2: Set Up z-buffer and Double Buffering

    int main(int argc, char **argv)
    {
        glutInit(&argc, argv);
        // double buffering for smooth animation
        glutInitDisplayMode
        (GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA);
        // window creation and callbacks here
        init(); // our custom initialization
        glutMainLoop();
        return(0);
    }
Step 3: Install Callbacks

- Create window and set callbacks
  glutInitWindowSize(800, 800);
  glutCreateWindow("quad");
  glutReshapeFunc(myReshape);
  glutDisplayFunc(display);
  glutIdleFunc(spinQuad);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboard);

Step 5: Init Vertex Buffer Object (VBO)

GLuint buffer;
void initVBO() {
  glGenBuffers(1, &buffer);
  glBindBuffer(GL_ARRAY_BUFFER, buffer);
  glBufferData(GL_ARRAY_BUFFER, sizeof(positions) + sizeof(colors), NULL, GL_STATIC_DRAW); // init buffer's size, but don't assign any data to it
  // upload position data
  glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(positions), positions);
  // upload color data
  glBufferSubData(GL_ARRAY_BUFFER, sizeof(positions), sizeof(colors), colors);
}

Step 6: Init Pipeline Program

void initPipelineProgram() {
  // initialize shader pipeline program (next lecture)
  // ...
}

Step 7: Reshape Callback

Set projection and viewport, preserve aspect ratio
void myReshape(int w, int h) {
  GLfloat aspect = (GLfloat) w / (GLfloat) h;
  glViewport(0, 0, w, h);
  matrix->SetMatrixMode(OpenGLMatrix::Projection);
  matrix->LoadIdentity();
  matrix->Ortho(2.0, 2.0, -2.0/aspect, 2.0/aspect, 0.0, 10.0);
  matrix->SetMatrixMode(OpenGLMatrix::ModelView);
}

Step 8: Display Callback

Clear, rotate, draw, swap
GLfloat theta[3] = {0.0, 0.0, 0.0};
void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  matrix->LoadIdentity();
  matrix->LookAt(0, 0, 0, 0, 0, -1, 0, 1, 0); // default camera
  matrix->Rotate(theta[0], 1.0, 0.0, 0.0);
  matrix->Rotate(theta[1], 0.0, 1.0, 0.0);
  matrix->Rotate(theta[2], 0.0, 0.0, 1.0);
  bindProgram();
  renderQuad();
  glutSwapBuffers();
}
Step 9: Bind Program

```c
void bindProgram() {
    glBindBuffer(GL_ARRAY_BUFFER, buffer); // so that glVertexAttribPointer
    // calls are correct
    GLuint loc = glGetUniformLocation(program, "position");
    glEnableVertexAttribArray(loc);
    glVertexAttribPointer(loc, 3, GL_FLOAT, GL_FALSE, 0, offset);
    GLuint loc2 = glGetUniformLocation(program, "color");
    glEnableVertexAttribArray(loc2);
    glVertexAttribPointer(loc2, 4, GL_FLOAT, GL_FALSE, 0, offset);
    // write projection and modelview matrix to shader
    // next texture...
}
```

Step 10: Drawing the Quad

```c
void renderQuad() {
    GLint first = 0;
    GLsizei count = 4;
    glDrawArrays(GL_TRIANGLE_STRIP, first, count);
}
```

Step 11: Animation

- Set idle callback

```c
GLfloat delta = 2.0;
GLuint axis = 2;
void spinQuad() {
    // spin the quad delta degrees around the selected axis
    theta[axis] += delta;
    if (theta[axis] > 360.0) theta[axis] -= 360.0;
    // display result (do not forget this!)
    glutPostRedisplay();
}
```

Step 12: 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 13: 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(spinQuad);
}
```

Summary

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

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