

CSCI 420 Computer Graphics

Lecture 4

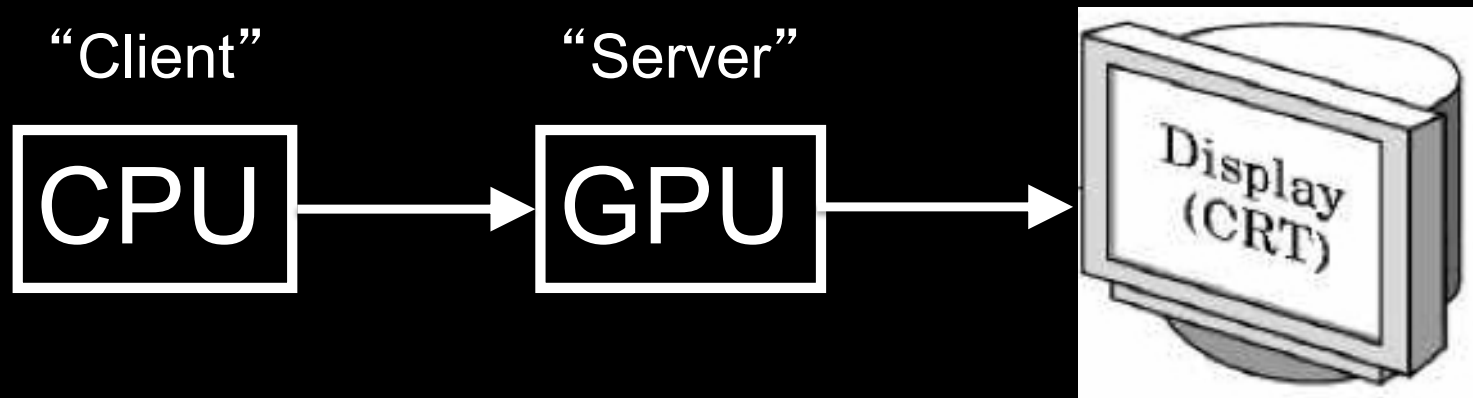
Interaction

Client/Server Model
Callbacks
Double Buffering
Hidden Surface Removal
[Angel Ch. 2]

Jernej Barbic
University of Southern California

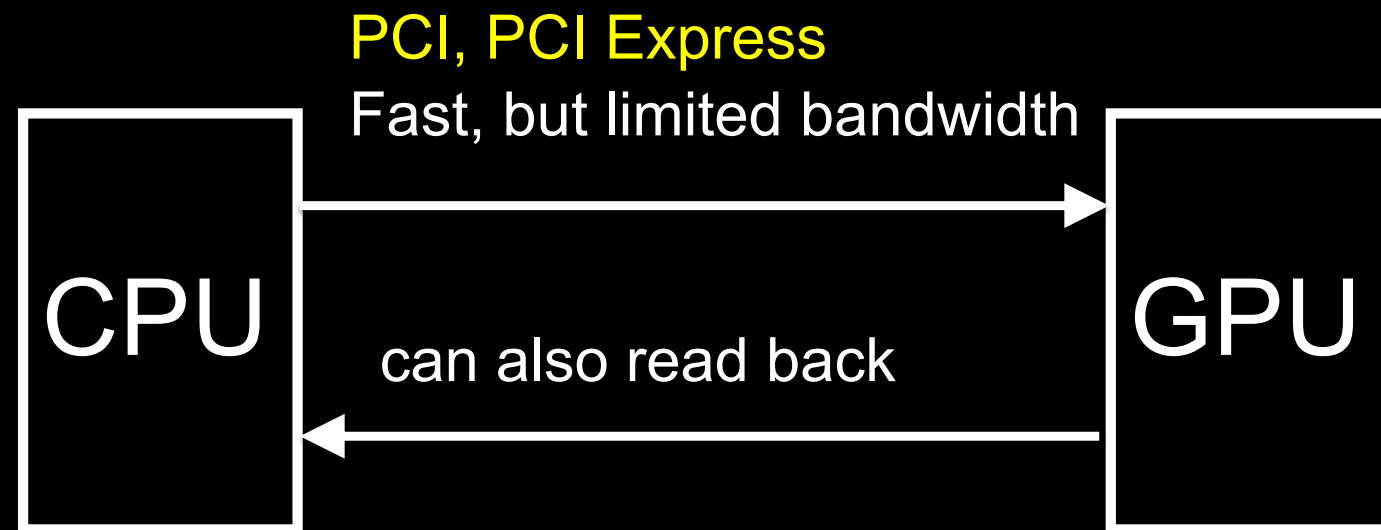
Client/Server Model

- Graphics hardware and caching



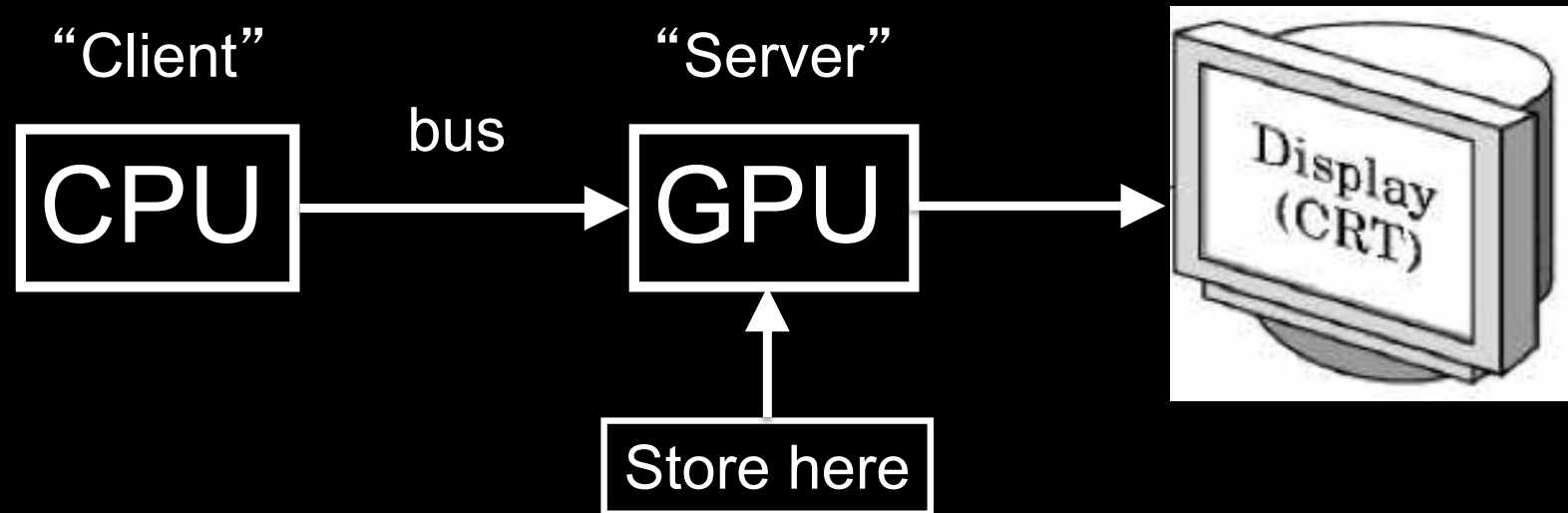
- Important for efficiency
- Need to be aware where data are stored
- Graphics driver code is on the CPU
- Rendering resources (buffers, shaders, textures, etc.) are on the GPU

The CPU-GPU bus



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_TRIANGLES) */
```

```
float positions[6][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}};
```

```
/* colors to be assigned to vertices (4th value is the alpha channel)  
*/
```

```
float colors[6][4] =  
    {{0.0, 0.0, 0.0, 1.0}, {1.0, 0.0, 0.0, 1.0}, {0.0, 1.0, 0.0, 1.0},  
     {0.0, 0.0, 1.0, 1.0}, {1.0, 1.0, 0.0, 1.0}, {1.0, 0.0, 1.0, 1.0}};
```

Vertex Buffer Object: Initialization

```
GLuint vbo;
```

```
void initVBO()
```

```
{
```

```
    glGenBuffers(1, &vbo);
```

```
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

```
    glBufferData(GL_ARRAY_BUFFER, sizeof(positions) + sizeof(colors),  
        nullptr, GL_STATIC_DRAW); // init VBO'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

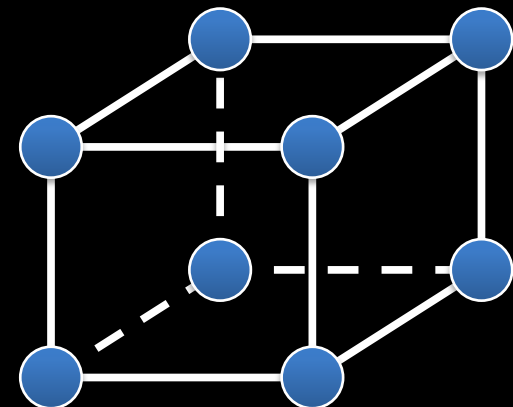
Display Lists

- Cache a sequence of drawing commands
- Optimize and store on server (GPU)

```
GLuint listName = glGenLists(1);  /* new list 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();
glEndList(); /* at this point, OpenGL compiles the list */
glCallList(listName);  /* draw the object */
```


Element Arrays

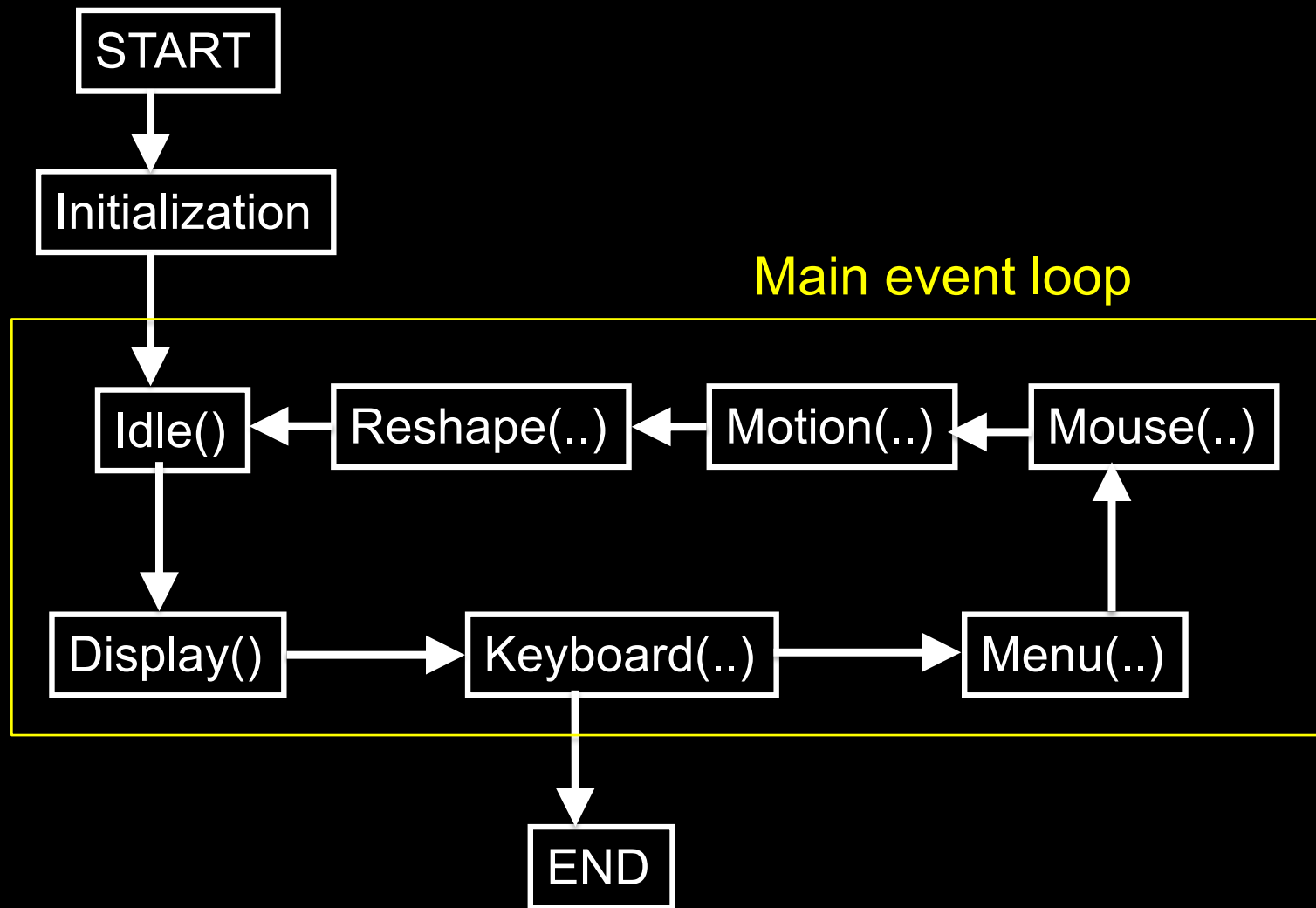
- Draw cube with $6 \times 2 \times 3 = 36$ or with 8 vertices?
- Expense in drawing and transformation
- Triangle strips help to some extent
- Element arrays provide general solution
- Define (transmit) array of vertices, colors, normals
- Draw using index into array(s) :
 // (must first set up the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



Outline

- Client/Server Model
- **Callbacks**
- Double Buffering
- Hidden Surface Removal

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

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

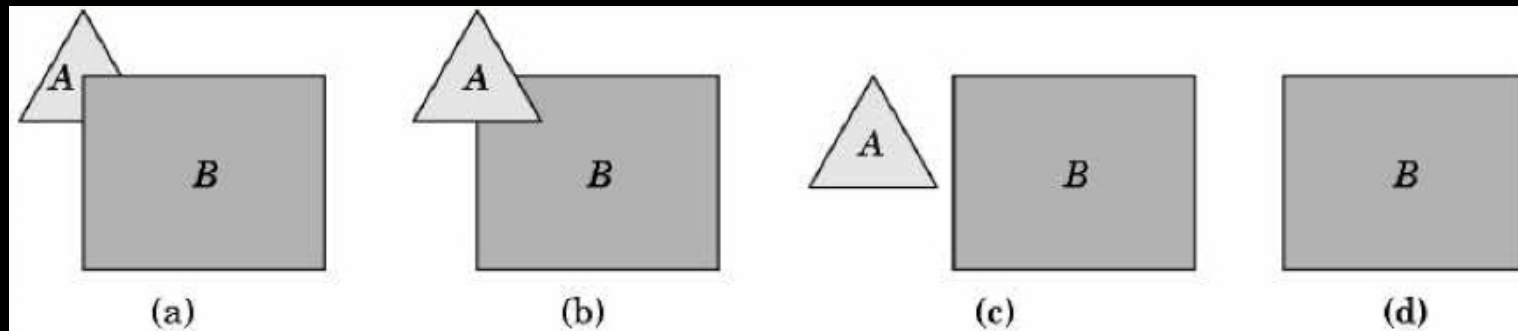
- Client/Server Model
- Callbacks
- Double Buffering
- Hidden Surface Removal

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

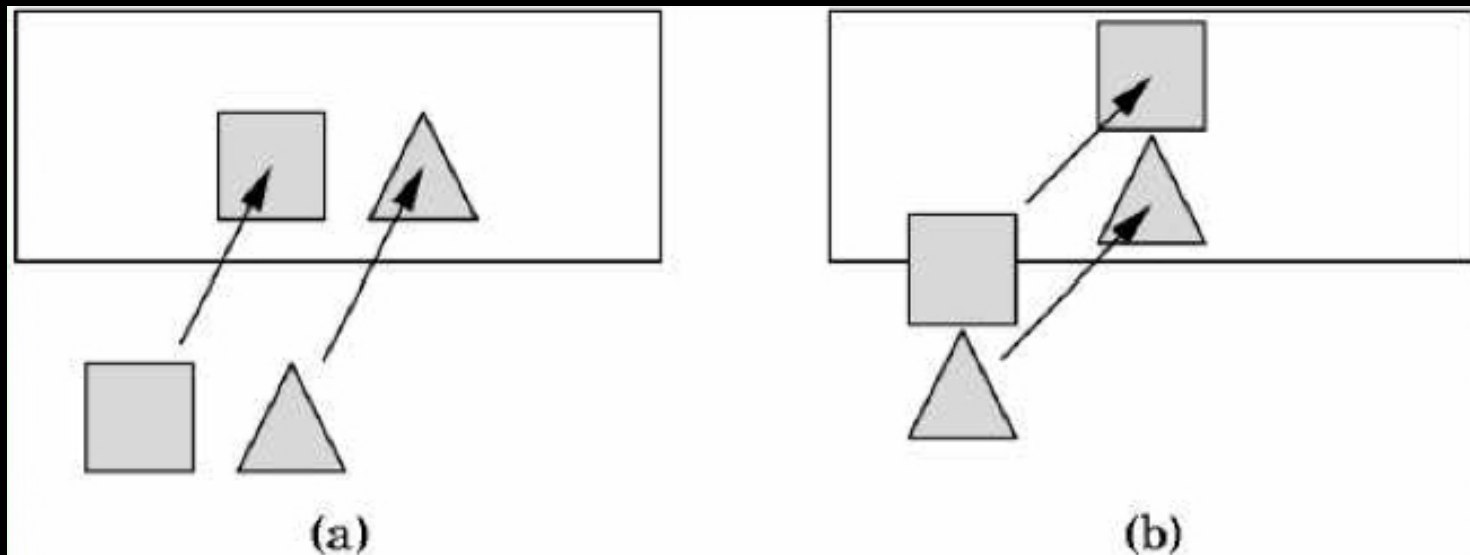
- Consider objects pairwise



- Painter's algorithm: render back-to-front
- "Paint" over invisible polygons
- How to sort and how to test overlap?

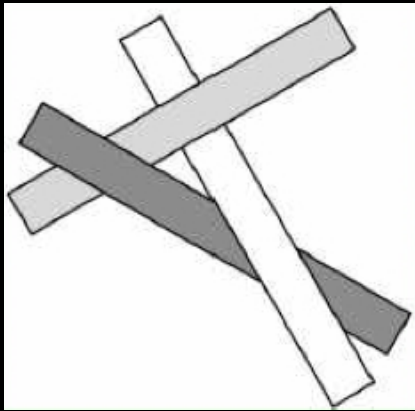
Depth Sorting

- 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

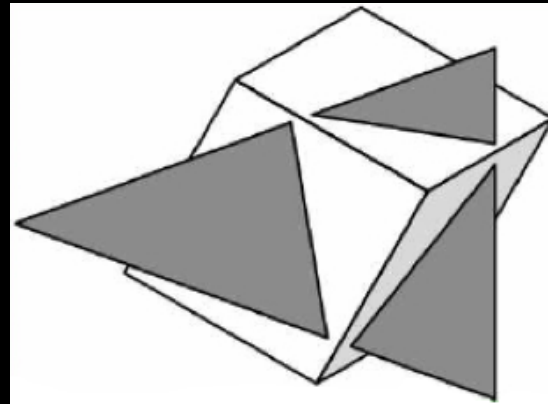


Some Difficult Cases

- Sometimes cannot sort polygons!



Cyclic overlap



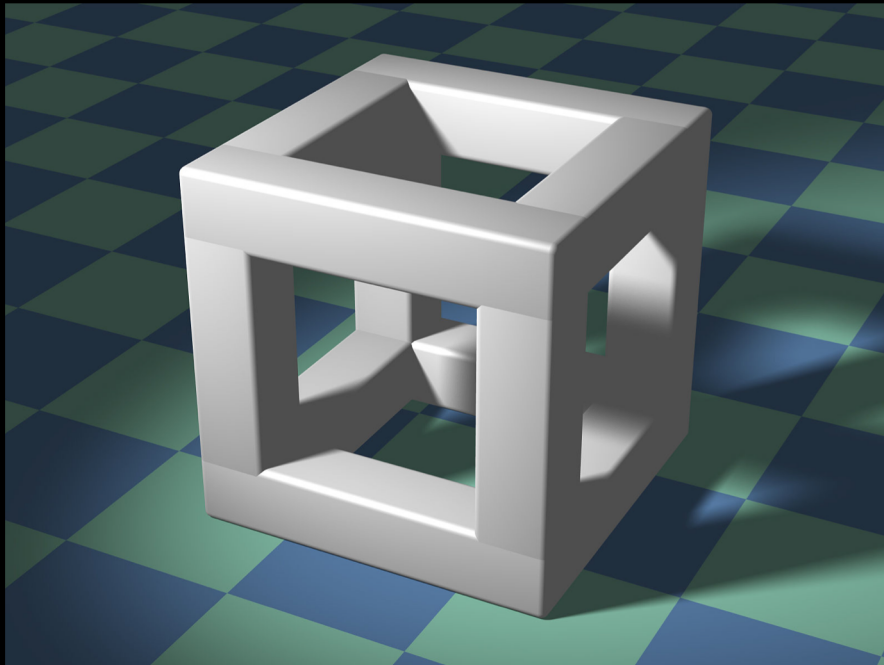
Piercing Polygons

- One solution: compute intersections & subdivide
- Do while rasterizing (difficult in object space)

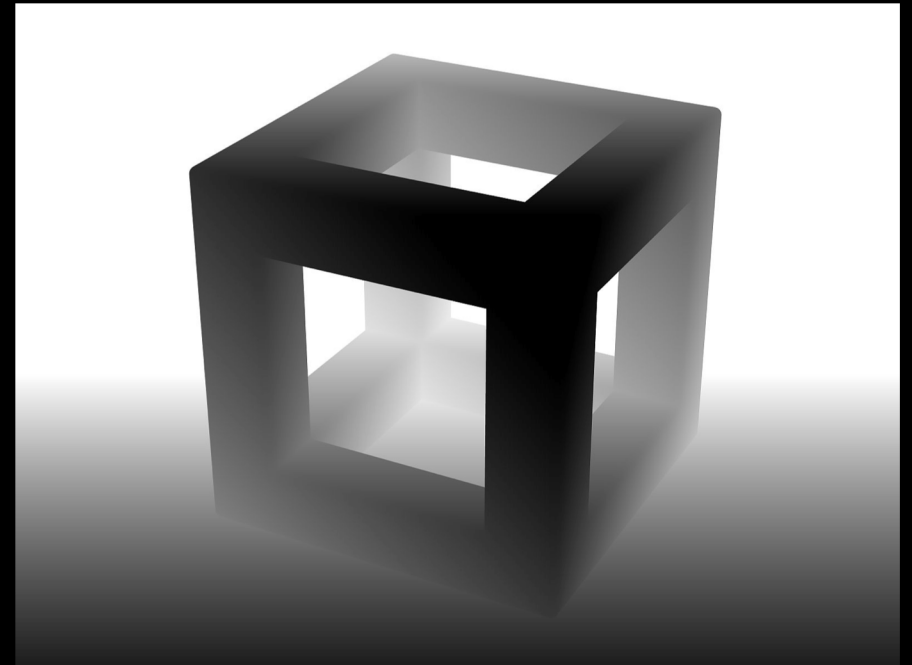
Painter's Algorithm Assessment

- 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



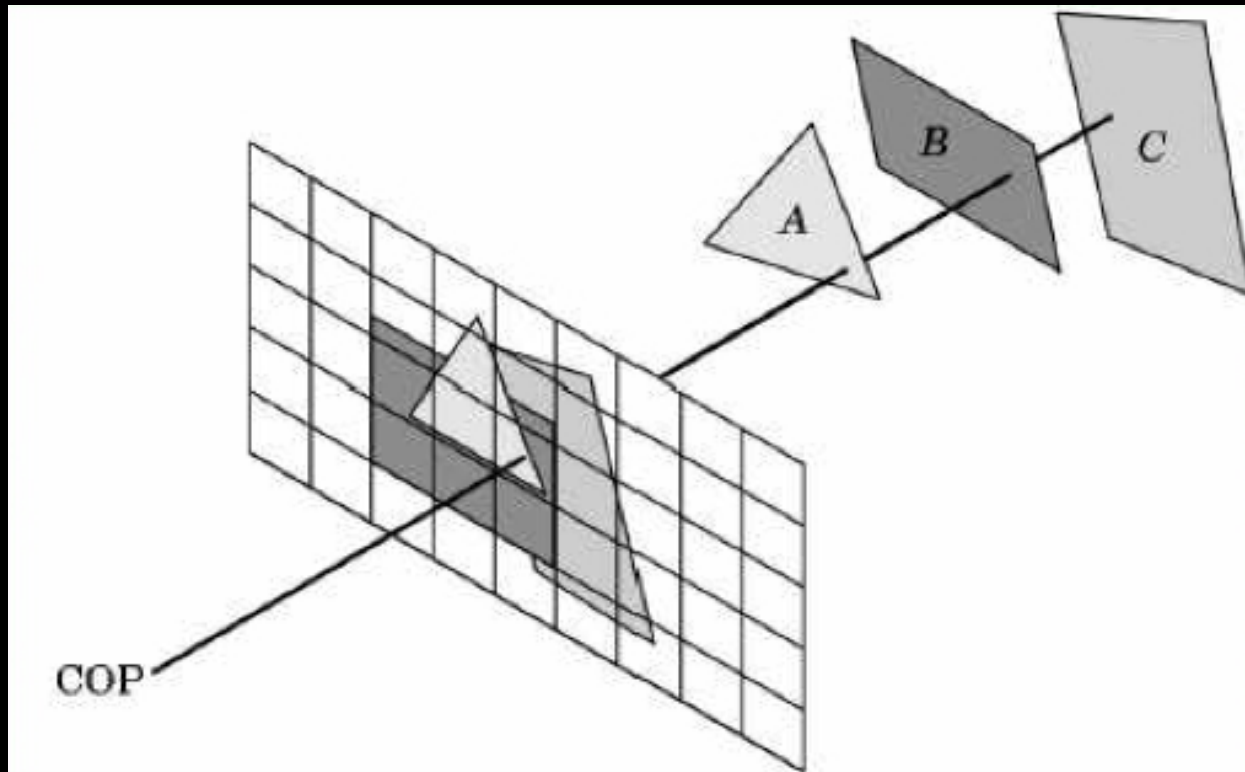
Depth image
darker color is closer

Depth sensor camera



Image-Space Approach

- Raycasting: intersect ray with polygons

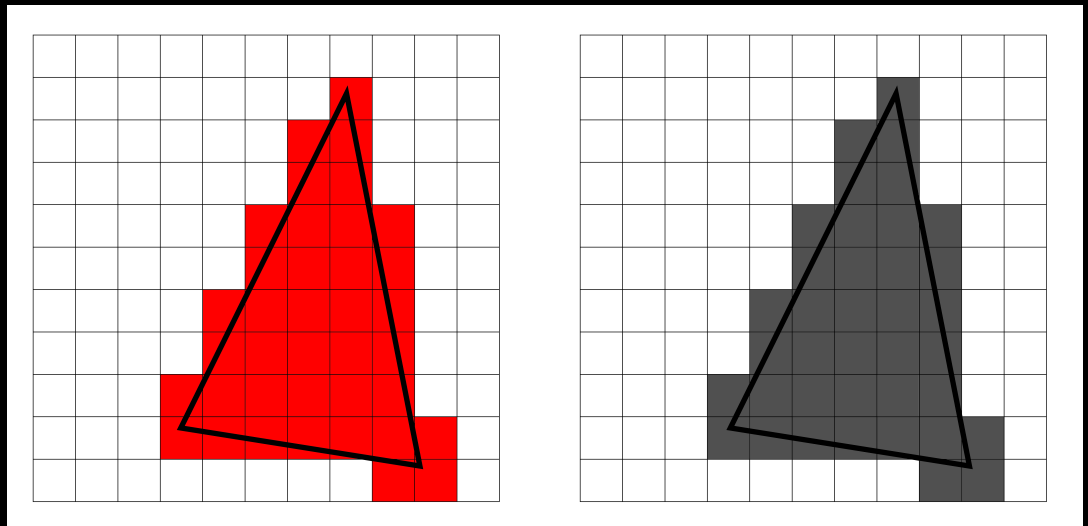
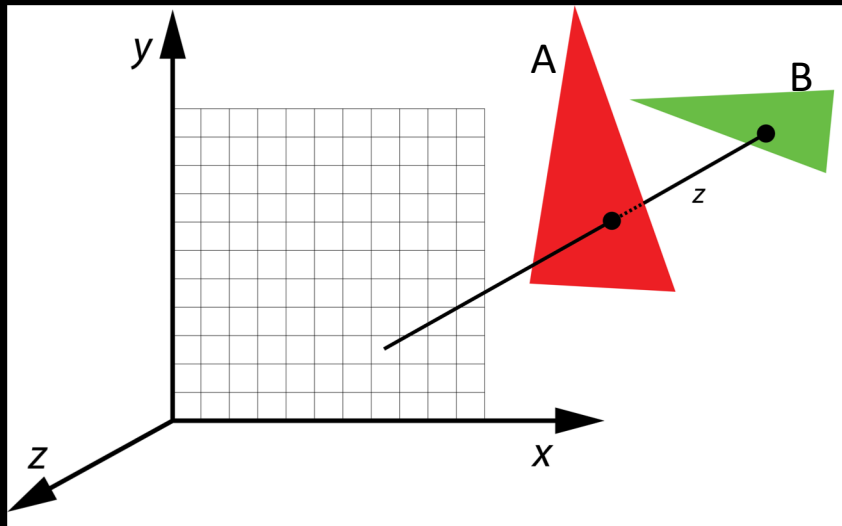


- $O(k)$ worst case (often better)
- Images can be more jagged (need anti-aliasing)

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

After rendering A:



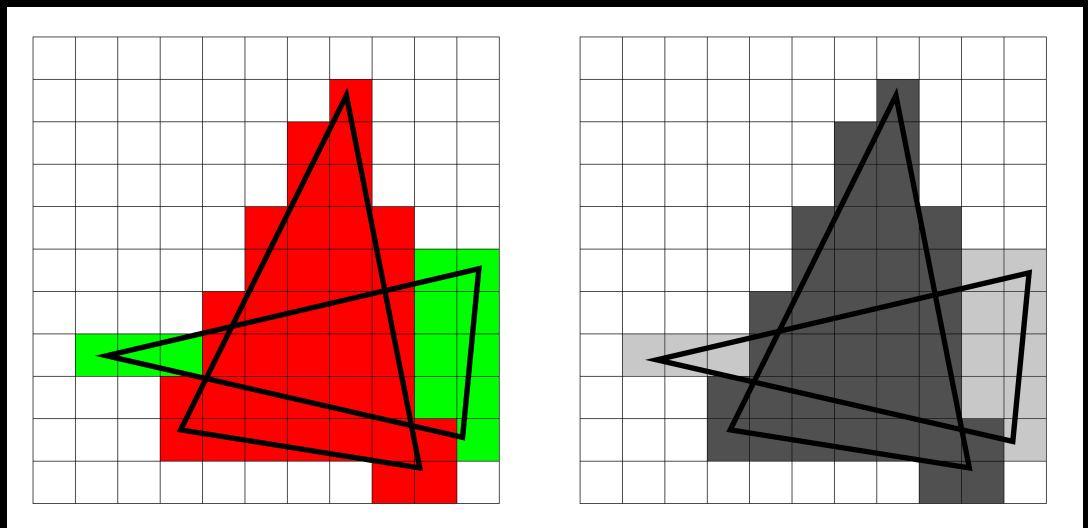
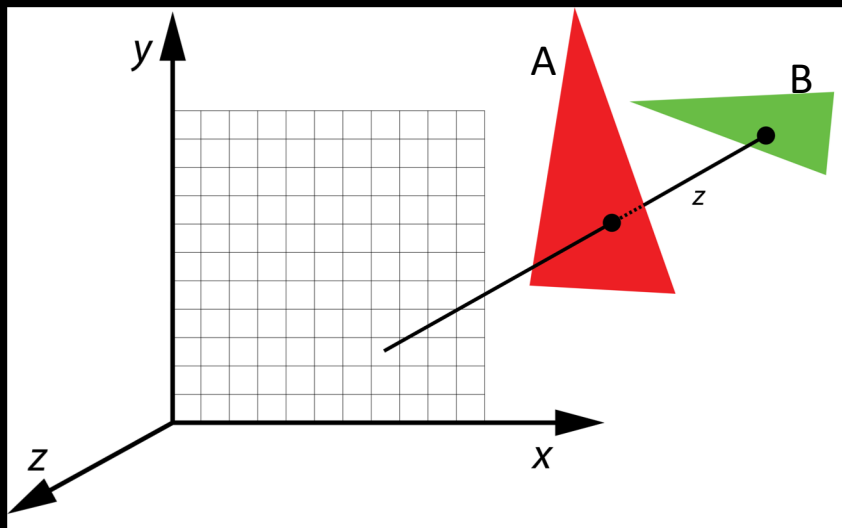
color

depth

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

After rendering A and B:



color

depth

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:

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

Summary

- Client/Server Model
- Callbacks
- Double Buffering
- Hidden Surface Removal