CS 432 – Interactive Computer Graphics

Lecture 09 – Part 1

Compositing and Anti-Aliasing

Composite Techniques

- Compositing is the method of compositing an image from several other images
- In OpenGL we can use the A (alpha) component in RGBA color to do this via blending or do stuff in the shader
- This allow for effects like
  - Transparency
  - Compositing images
  - General blending
  - Antialiasing

Blending

- We can either
  - Specify *alpha* values for our colors and allow OpenGL to do blending or
  - Do blending ourselves in the shaders

Alpha Values

- Alpha=1
  - A surface is completely **opaque**
- Alpha = 0
  - A surface is completely **transparent**

OpenGL Blending and Compositing

- The current fragment is considered the *source*
- The frame buffer pixel is considered the *destination*
- There may already be a color in the destination, so we must decide how the source should be combined with the destination
- Must enable blending and pick source and destination factors
  - `glEnable(GL_BLEND)`
  - `glBlendFunc(source_factor, destination_factor)`
- Only certain factors are supported
  - `GL_ZERO, GL_ONE`
  - `GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA`
  - `GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA`
  - And a few more (see Red Book)

Transparency

- Unfortunately rendering order *does* matter
- Solutions:
  - Somehow sort by depth and the render them from front to back
    - Maybe slow and unclear how to do
  - Allow opaque objects read/write access to z-buffer but allow transparent object read only

Transparency with Z-Buffer

1. Enable z-buffer to writing
   - `glDepthMask(GL_TRUE);`
2. Render opaque objects
3. Make z-buffer read only
   - `glDepthMask(GL_FALSE);`
4. Enable blending
   - `glEnable(GL_BLEND)`
   - `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
5. Render translucent objects
6. Re-enable z-buffer for writing
   - `glDepthMask(GL_TRUE);`
7. Disable blending (no next time around it won't start by blending)
   - `glDisable(GL_BLEND);`

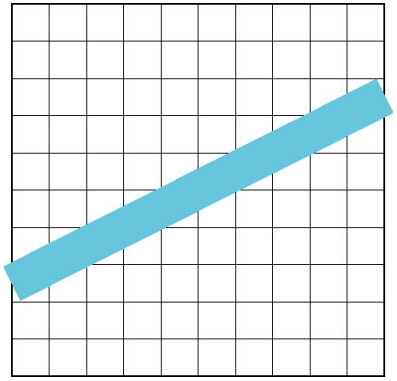
Fog

- We can composite with a fixed color and have the blending factors depend on depth
  - Simulates a fog effect
- Blend source colors C_s and fog color C_f
- f is the *fog factor* and can be
  - Exponential
  - Gaussian
  - Linear (depth cueing)

  based on the depth (distance of vertex from eye)
- We can do this in the vertex shader

Aliasing

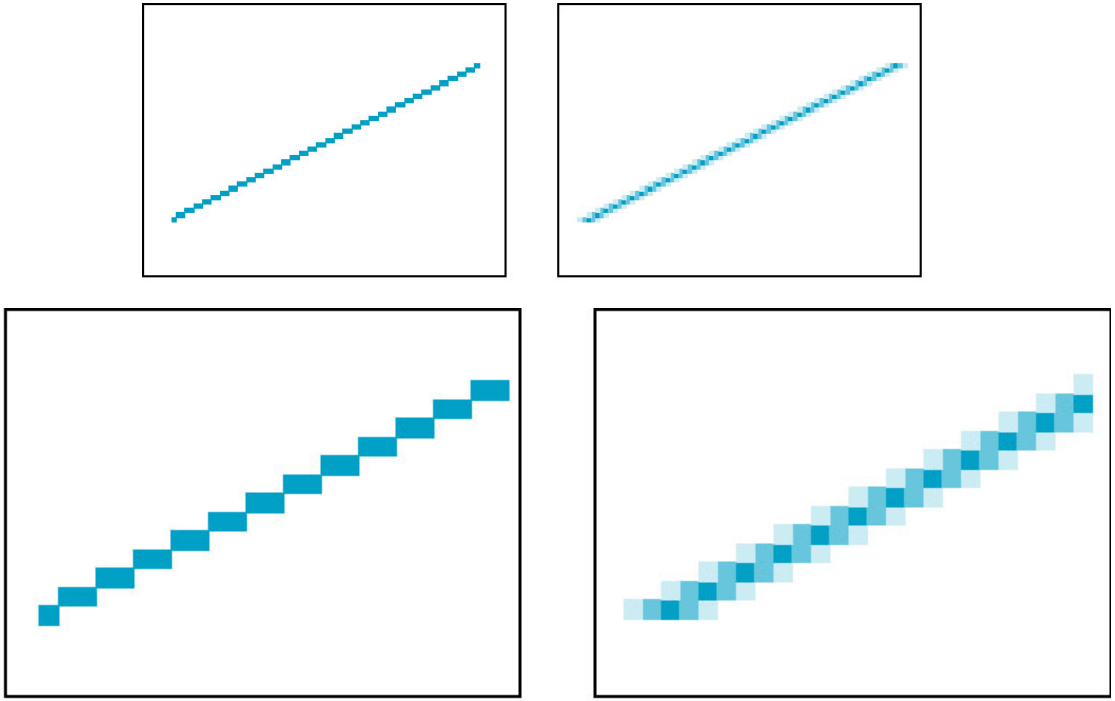
- You may have noticed in your renderings that sometimes lines look oddly jagged
- This is due to a phenomenon known as *aliasing* where multiple values are mapped to the same pixel



- Ideal rasterized line should be 1 pixel wide
  - Choosing best y for each x (or vice versa) produces aliased raster lines

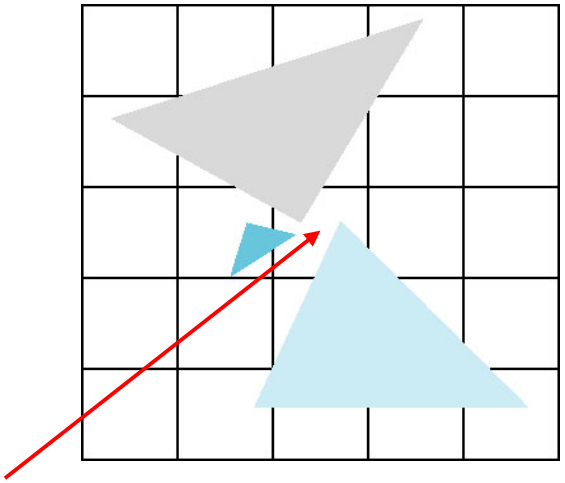
Antialiasing by Area Averaging

- Color multiple pixels for each x depending on coverage by ideal line



Polygon Aliasing

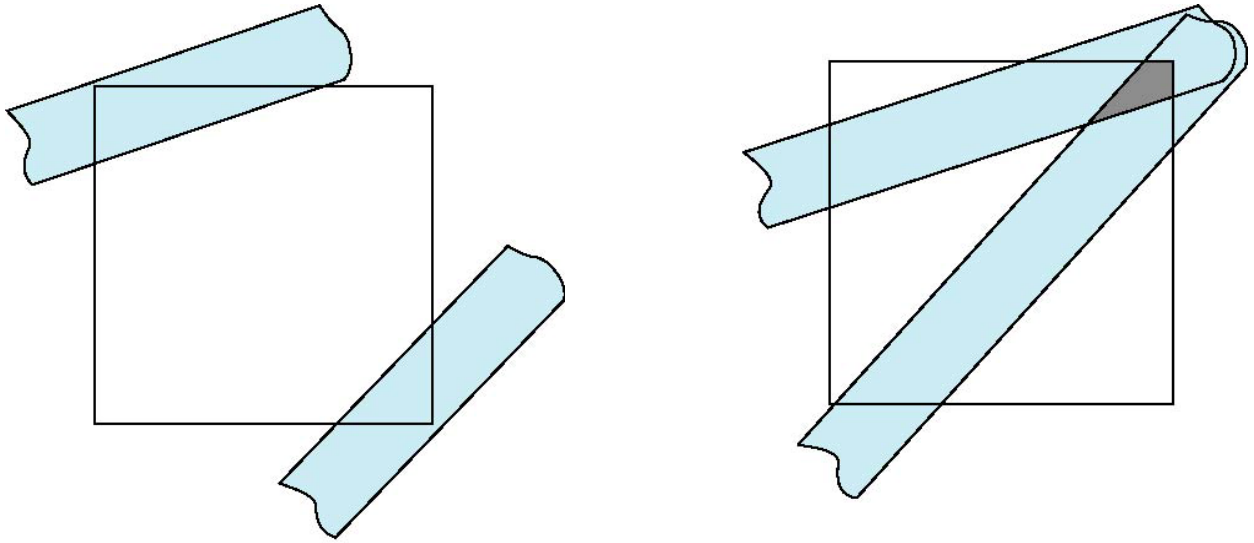
- Aliasing problems can be serious for polygons
  - Jaggedness of edges
  - Small polygons neglected
  - Need compositing so color of one polygon does not totally determine color of pixel



All three polygons should contribute to color

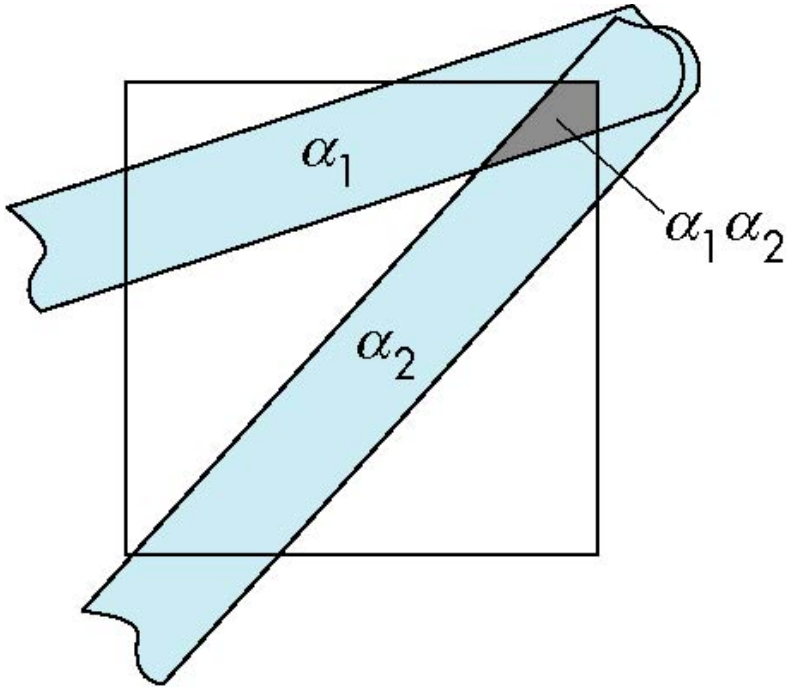
Antialiasing

- Can try to color a pixel by adding a fraction of its color to the frame buffer
  - Fraction depends on percentage of pixel covered by fragment
  - Fraction depends on whether there is an overlap



Area Averaging

- Use average area $\alpha_1 + \alpha_2 - \alpha_1 \alpha_2$ as blending factor



OpenGL Anti-Aliasing

- In OpenGL we can invoke anti-aliasing without having the user combining alpha values themselves
  - OpenGL computes an alpha value that represents the fraction of each pixel covered by the line or point as a function of the distance of the pixel center from the line center
- Can enable separately for points, lines, or polygons

```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_POLYGON_SMOOTH);

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- Like all blending techniques, it is order dependent

Antialiasing via Multisampling

- We already saw multi-rendering
- In this mode, OpenGL samples each pixel in the framebuffer several times and to generate the final image, all the samples are combined for each pixel
- If we use glut, we can specify multi-sampling as an option in the `glutinitDisplayMode` call
  - `glutInitDisplayMode(GLUT_MULTISAMPLE)`
- Then we can enable/disable this as desired
  - `glEnable(GL_MULTI_SAMPLE)`

MultiSampling

Mipmaps

- Recall that when you assign data to textures you can assign the mipmap level
  - `glTexImage2D(GL_TEXTURE_2D,level,…)`
- Or you can let OpenGL make them for you
  - `glGenerateMipmaps(GL_TEXTURE_2D);`
  - ```
    glTexParameterf(GL_TEXTURE_2D,
                    GL_TEXTURE_MAP_FILTER,
                    GL_NEAREST_MIPMAP_NEAREST);
    ```
- If there are textures at multiple resolutions OpenGL can choose from them in order to minimize the effects of magnification and minification
  - Which is another type of aliasing
    ```