CS 112 - Compositing Techniques
The Alpha Channel

- In addition to RGB, the fourth alpha channel
- Alpha blending
- Application can control the value of alpha at every pixel
Compositing Functions

- **Source** $\alpha$ - associated with the triangle
- **Destination** $\alpha$ – associated with a pixel in the frame buffer

$S = [s_r, s_g, s_b, s_a]$, $D = [d_r, d_g, d_b, d_a]$  
$D' = f_s(s_a,d_a)S + f_d(s_a,d_a)D$  
$= s_aS + (1-s_a)D$ - Transparency
Transparency

- \[ D' = s_a S + (1-s_a)D \]
- Color of the triangle being rendered is attenuated by \( s_a \)
- Color existing in the framebuffer attenuated by \( 1-s_a \)
- These colors are added to create the new color in the framebuffer
Transparency

- $D' = s_a S + (1-s_a)D$
- Opaque triangle has $s_a = 1$
  - Framebuffer gets overwritten
- Transparent triangle has $s_a = 0$
  - Framebuffer remains unchanged
- Translucent triangle has $0 < s_a < 1$
  - Color gets blended
Transparency

Chicken = 1, Egg = 0  Chicken = 0.5, Egg = 0.5  Chicken = 0, Egg = 1
Problems

- Will show only A - Wrong
Problems

- Will have contribution from B – Wrong
- Depends on the order of rendering
How to solve this?

- Order triangles back to front and render
- Order-dependent rendering
  - Very Expensive
Optimizations

- Render opaque objects first (occlusion resolved)
- Order translucent objects back to front
- Render them back to front
Problems

- This does not solve this problem
Optimizations

- Render opaque objects first (occlusion resolved)
- Set z-buffer to read only
  - Retains the depth of opaque objects only
- Order translucent objects back to front
- Render them back to front
  - Only if they pass the z-buffer test
  - Only if no opaque objects are in-front of it
Results
Accumulation Buffer

- Compositing images in framebuffer
- Limited color resolution
- Clamping and washed out appearance
- Accumulation buffer – floating point colors
  - Higher *color* resolution
- Do weighted accumulation
  - Transfer the result to framebuffer
  - Greater *color* precision
Anti-aliasing

- Say we have a frame buffer of 100x100
- And our scene has frequencies till 100 Hz
- Approach
  - Render the scene in a 200x200 framebuffer
    - Sufficient sampling and hence no artifacts
  - Filter it to 100x100 to remove the higher frequencies
    - Suitable for 100x100 and hence anti-aliased
Anti-aliasing

- Say we have a frame buffer of 100x100
- And our scene has frequencies till 100 Hz
- Approach
  - **Super-sample** the scene at a higher resolution
  - **Filter** it to a lower resolution
- How to achieve this if your framebuffer has a limited *spatial* resolution?
  - Say, cannot have a framebuffer of more than 100x100 resolution
Use Accumulation buffer

- Each pixel generated is a point sample of the scene
- Rendering is a process of generating the point samples
Use Accumulation buffer

- If we can generate more than one sample per pixel
- Average the samples
- Same effect
Jittering the view point

- Jitter the view point
- The projected screen coordinate jitter should be less than a pixel
- Keep accumulating with appropriate weight
- Can achieve the effect with a low *spatial* resolution accumulation buffer

\[
\frac{1}{4} \text{ of green} + \frac{1}{4} \text{ of red} + \frac{1}{4} \text{ of blue} + \frac{1}{4} \text{ of orange}
\]