Rasterization

Blending

- Frame buffer
  - Simple color model: R, G, B; 8 bits each
  - α-channel A, another 8 bits
- Alpha determines opacity, pixel-by-pixel
  - $\alpha = 1$: opaque
  - $\alpha = 0$: transparent
- Blend translucent objects during rendering
- Achieve other effects (e.g., shadows)

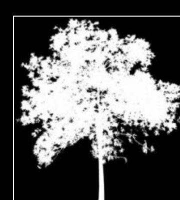


Image Compositing

- Compositing operation
  - Source: $\mathbf{s} = [s_r \ s_g \ s_b \ s_a]$
  - Destination: $\mathbf{d} = [d_r \ d_g \ d_b \ d_a]$
  - $\mathbf{b} = [b_r \ b_g \ b_b \ b_a]$ source blending factors
  - $\mathbf{c} = [c_r \ c_g \ c_b \ c_a]$ destination blending factors
  - $\mathbf{d'} = [b_r s_r + c_r d_r \ \ b_g s_g + c_g d_g \ \ b_b s_b + c_b d_b \ \ b_a s_a + c_a d_a]$
- Overlay n images with equal weight
  - Set α-value for each pixel in each image to 1/n
  - Source blending factor is "α"
  - Destination blending factor is "1"

Blending in OpenGL

- Enable blending

```
glEnable(GL_BLEND);
```

- Set up source and destination factors

```
glBlendFunc(source_factor, dest_factor);
```

- Source and destination choices
  - `GL_ONE, GL_ZERO`
  - `GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA`
  - `GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA`

Blending Errors

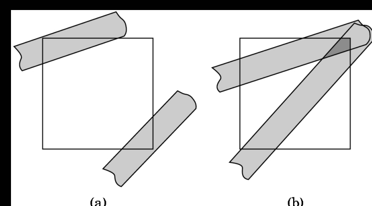
- Operations are not commutative
- Operations are not idempotent
- Interaction with hidden-surface removal
  - Polygon behind opaque one should be culled
  - Translucent in front of others should be composited
  - Solution:
    - Two passes using *alpha testing* (glAlphaFunc): 1st pass alpha=1 accepted, and 2nd pass alpha<1 accepted
    - make z-buffer read-only for translucent polygons (alpha<1) with `glDepthMask(GL_FALSE);`

Antialiasing Revisited

- Single-polygon case first
- Set α-value of each pixel to covered fraction
- Use destination factor of "$1 - \alpha$"
- Use source factor of "α"
- This will blend background with foreground
- Overlaps can lead to blending errors

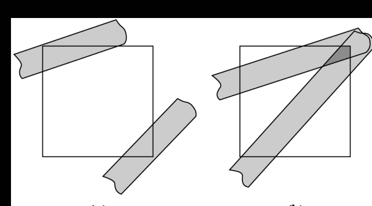
Antialiasing with Multiple Polygons

- Initially, background color $\mathbf{C}_0$, $\alpha_0 = 0$
- Render first polygon; color C_1fraction α_1
  - $\mathbf{C}_d = (1 - \alpha_1)\mathbf{C}_0 + \alpha_1\mathbf{C}_1$
  - $\alpha_d = \alpha_1$
- Render second polygon; assume fraction α_2
- If no overlap (a), then
  - $\mathbf{C'}_d = (1 - \alpha_2)\mathbf{C}_d + \alpha_2\mathbf{C}_2$
  - $\alpha'_d = \alpha_1 + \alpha_2$



Antialiasing with Overlap

- Now assume overlap (b)
- Average overlap is $\alpha_1\alpha_2$
- So $\alpha_d = \alpha_1 + \alpha_2 - \alpha_1\alpha_2$
- Make front/back decision for color as usual



Antialiasing in OpenGL

- Avoid explicit α-calculation in program
- Enable both smoothing and blending

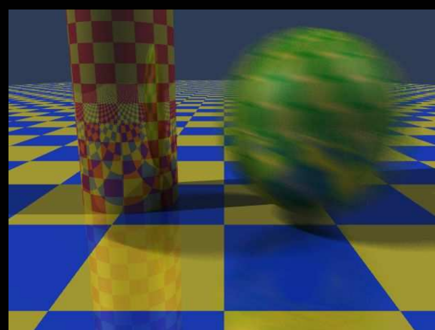
```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Temporal Aliasing

- Sampling rate is frame rate (30 Hz for video)
- Example: spokes of wagon wheel in movie
- Possible to supersample and average
- Fast-moving objects are blurred
- Happens automatically in video and movies
  - Exposure time (shutter speed)
  - Memory persistence (video camera)
  - Effect is motion blur

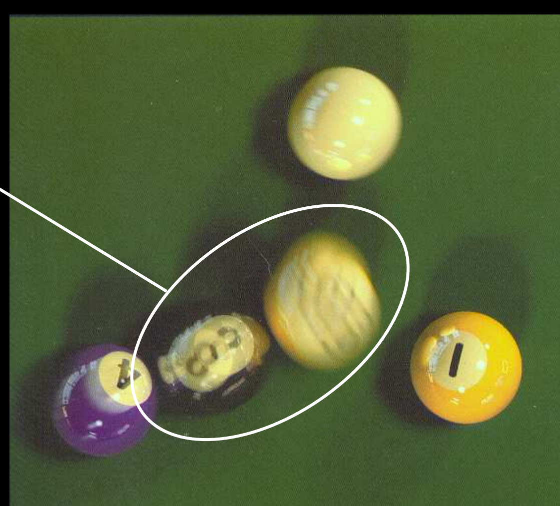
Motion Blur

- Achieve by stochastic sampling in time
- Still-frame motion blur, but smooth animation



Motion Blur Example

Looks like squash
and stretch!!

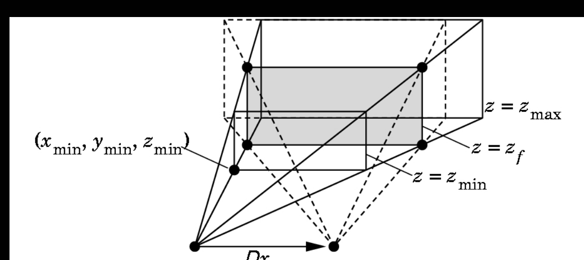


T. Porter, Pixar, 1984
16 samples/pixel

Depth of Field

Filter for Depth-of-Field

- Simulate camera depth-of-field
  - Keep plane $z = z_f$ in focus
  - Keep near and far planes unchanged
- Move viewer by Δx
- Compute x'_{min}, x'_{max}, y'_{min}, y'_{max} for new frustum



Depth-of-Field Jitter

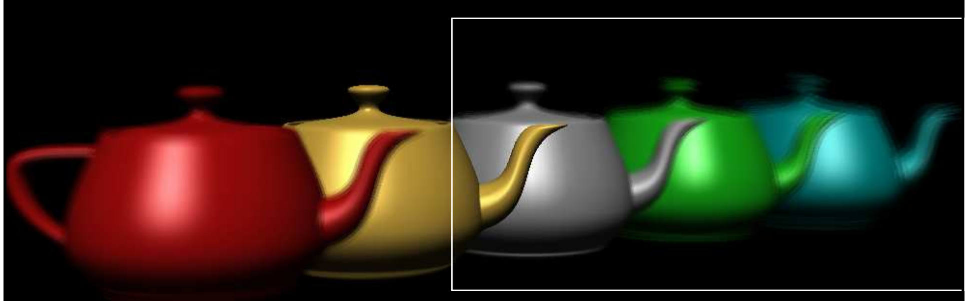
- Compute

$$x'_{min} = x_{min} + \frac{\Delta x}{z_f}(z_f - z_{min})$$

- Blend the two images in accumulation buffer

OpenGL Depth of Field Example

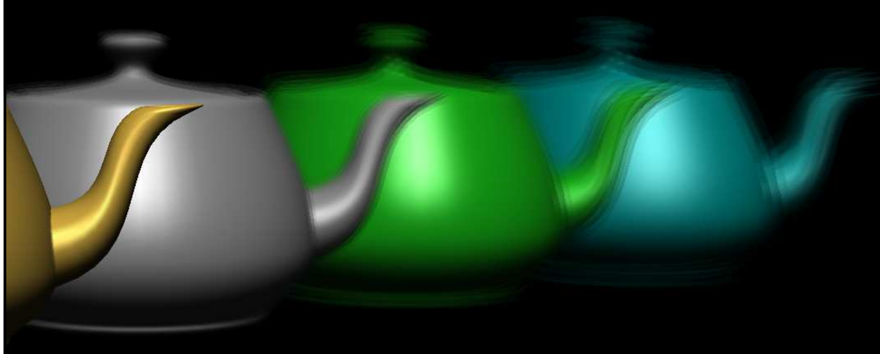
Can jitter in both x- and y-directions…



See depth of field example:

http://www.opengl.org/developers/code/examples/redbook/redbook.html

Close-up



Soft shadows too…

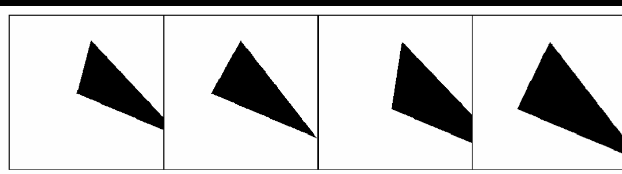


Figure 1: Hard shadow images from 2×2 grid of sample points on light source.

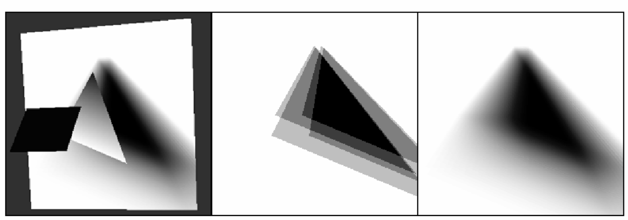


Figure 2: Left: scene with square light source (foreground), triangular occluder (center), and rectangular receiver (background), with shadows on receiver. Center: Approximate soft shadows resulting from 2×2 grid of sample points; the average of the four hard shadow images in Figure 1. Right: Correct soft shadow image (generated with 16×16 sampling). This image is used as the texture on the receiver at left.

Simulating Soft Shadows with Graphics Hardware (1997)
Paul S. **Heckbert** & Michael **Herf,** CMU Technical Report

Depth Cueing and Fog

- Another application of blending
- Use distance-dependent (z) blending
  - Linear dependence: depth cueing effect
  - Exponential dependence: fog effect
  - This is not a physically-based model

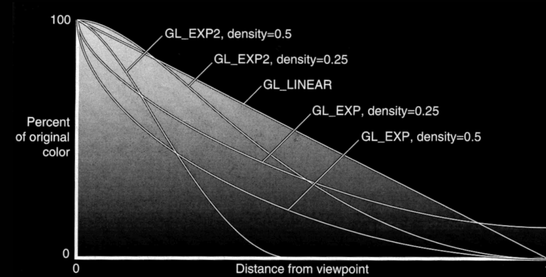
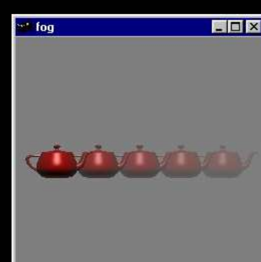


Figure 6-4 Fog-Density Equations

Example: Fog



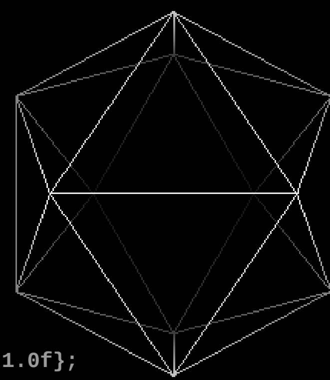
- Fog in RGBA mode:

$$C = fC_i + (1-f)C_f$$

  - f : depth-dependent fog factor

```
GLfloat fcolor[4] = {...};
glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_EXP);
glFogf(GL_FOG_DENSITY, 0.5);
glFogfv(GL_FOG_COLOR, fcolor);
```

Example: Depth Cue



```
float fogColor[] = {0.0f, 0.0f, 0.0f, 1.0f};
gl.glEnable(GL_FOG);
gl.glFogi (GL_FOG_MODE, GL_LINEAR);
gl.glHint (GL_FOG_HINT, GL_NICEST);  /*  per pixel   */
gl.glFogf (GL_FOG_START, 3.0f);
gl.glFogf (GL_FOG_END, 5.0f);
gl.glFogfv (GL_FOG_COLOR, fogColor);
gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```

Antialiasing – again!

(Jim Blinn article)