Perspective-Correct Textures (11.3.1)

- Linear interpolation of positions OK in screen space
- Leads to foreshortening effect in other attributes
  - texture coordinates, colors, etc.

Hyperbolic Interpolation
- Observe that $1/h$ is interpolated in screen space w/o distortion
- Therefore interpolate $(1/h, u/h, v/h)$ in screen space then divide interpolated $(u/h, v/h)$ by interpolated $(1/h)$ value.
- Also known as rational linear interpolation
- Linear interpolation can be done with a DDA as before
- Early implementations attempt to reduce division costs
  - Quake: divide only every 16 pixels of scanline, & lin. interp.

Pixel coverage

- Antialiasing and compositing both deal with questions of pixels that contain unresolved detail
- Antialiasing: how to carefully throw away the detail
- Compositing: how to account for the detail when combining images

Antialiasing & Compositing

CS4620 Lecture 15
Signal processing view

- Recall this picture:

  ![Signal processing diagram]  
  
  we need to do this step

Antialiasing

- A name for techniques to prevent aliasing
  - In image generation, we need to lowpass filter
    - Sampling the convolution of filter & image
    - Boils down to averaging the image over an area
    - Weight by a filter
  - Methods depend on source of image
    - Rasterization (lines and polygons)
    - Point sampling (e.g. ray tracing)
    - Texture mapping

Rasterizing lines

- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside

Point sampling

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: all-or-nothing leads to jaggies
  - this is sampling with no filter (aka. point sampling)
**Aliasing**

- Point sampling is fast and simple
- But the lines have stair steps and variations in width
- This is an aliasing phenomenon
  - Sharp edges of line contain high frequencies
- Introduces features to image that are not supposed to be there!

**Antialiasing**

- Point sampling makes an all-or-nothing choice in each pixel
  - therefore steps are inevitable when the choice changes
  - yet another example where discontinuities are bad
- On bitmap devices this is necessary
  - hence high resolutions required
  - 600+ dpi in laser printers to make aliasing invisible
- On continuous-tone devices we can do better

- Basic idea: replace “is the image black at the pixel center?” with “how much is pixel covered by black?”
- Replace yes/no question with quantitative question.
**Box filtering**

- Pixel intensity is proportional to area of overlap with square pixel area
- Also called “unweighted area averaging”

**Box filtering by supersampling**

- Compute coverage fraction by counting subpixels
- Simple, accurate
- But slow

**Weighted filtering**

- Box filtering problem: treats area near edge same as area near center
  - results in pixel turning on “too abruptly”
- Alternative: weight area by a smoother filter
  - unweighted averaging corresponds to using a box function
  - sharp edges mean high frequencies
    - so want a filter with good extinction for higher freqs.
  - a Gaussian is a popular choice of smooth filter
  - important property: normalization (unit integral)
### Weighted filtering by supersampling

- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow

### Filter comparison

- Point sampling
- Box filtering
- Gaussian filtering

### Antialiasing and resampling

- Antialiasing by regular supersampling is the same as rendering a larger image and then resampling it to a smaller size
- Convolution of filter with high-res image produces an estimate of the area of the primitive in the pixel.
- So we can re-think this
  - one way: we're computing area of pixel covered by primitive
  - another way: we're computing average color of pixel
  - this way generalizes easily to arbitrary filters, arbitrary images
More efficient antialiased lines

- Filter integral is the same for pixels the same distance from the center line
- Just look up in precomputed table based on distance – Gupta-Sproull
- Does not handle ends…

Antialiasing in ray tracing

Aliased image

One sample per pixel

Antialiased image

Four samples per pixel
Antialiasing in ray tracing

One sample/pixel

9 samples/pixel

Details of supersampling

- For image coordinates with integer pixel centers:

```java
// one sample per pixel
for iy = 0 to (ny-1) by 1
    for ix = 0 to (nx-1) by 1 {
        ray = camera.getRay(ix, iy);
        image.set(ix, iy, trace(ray));
    }

// ns^2 samples per pixel
for iy = 0 to (ny-1) by 1
    for ix = 0 to (nx-1) by 1 {
        Color sum = 0;
        for dx = -(ns-1)/2 to (ns-1)/2 by 1
            for dy = -(ns-1)/2 to (ns-1)/2 by 1 {
                x = ix + dx / ns;
                y = iy + dy / ns;
                ray = camera.getRay(x, y);
                sum += trace(ray);
            }
        image.set(ix, iy, sum / (ns*ns));
    }
```

Details of supersampling (continued)

- For image coordinates in unit square

```java
// one sample per pixel
for iy = 0 to (ny-1) by 1
    for ix = 0 to (nx-1) by 1 {
        double x = (ix + 0.5) / nx;
        double y = (iy + 0.5) / ny;
        ray = camera.getRay(x, y);
        image.set(ix, iy, trace(ray));
    }

// ns^2 samples per pixel
for iy = 0 to (ny-1) by 1
    for ix = 0 to (nx-1) by 1 {
        Color sum = 0;
        for dx = 0 to (ns-1) by 1
            for dy = 0 to (ns-1) by 1 {
                x = (ix + (dx + 0.5) / ns) / nx;
                y = (iy + (dy + 0.5) / ns) / ny;
                ray = camera.getRay(x, y);
                sum += trace(ray);
            }
        image.set(ix, iy, sum / (ns*ns));
    }
```

Antialiasing in textures

- Would like to render textures with one (or few) s/p
- Need to filter first!
  - perspective produces very high image frequencies

...
Mipmap image pyramid

Texture minification

Texture minification

Compositing
Combining images

• Often useful combine elements of several images
• Trivial example: video crossfade
  – smooth transition from one scene to another
  \[
  r_C = tr_A + (1-t)r_B \\
  g_C = tg_A + (1-t)g_B \\
  b_C = tb_A + (1-t)b_B
  \]
  – note: weights sum to 1.0
  • no unexpected brightening or darkening
  • no out-of-range results
  – this is linear interpolation

Foreground and background

• In many cases just adding is not enough
• Example: compositing in film production
  – shoot foreground and background separately
  – also include CG elements
  – this kind of thing has been done in analog for decades
  – how should we do it digitally?

Foreground and background

• How we compute new image varies with position
• Therefore, need to store some kind of tag to say what parts of the image are of interest

Binary image mask

• First idea: store one bit per pixel
  – answers question “is this pixel part of the foreground?”
  – causes jaggies similar to point-sampled rasterization
  – same problem, same solution: intermediate values
Partial pixel coverage

- The problem: pixels near boundary are not strictly foreground or background
  - how to represent this simply?
  - interpolate boundary pixels between the fg. and bg. colors

Alpha compositing

- Formalized in 1984 by Porter & Duff
- Store fraction of pixel covered, called $\alpha$

$$C = A \text{ over } B$$
$$r_C = \alpha_A r_A + (1 - \alpha_A) r_B$$
$$g_C = \alpha_A g_A + (1 - \alpha_A) g_B$$
$$b_C = \alpha_A b_A + (1 - \alpha_A) b_B$$

- this exactly like a spatially varying crossfade
- Convenient implementation
  - 8 more bits makes 32
  - 2 multiplies + 1 add per pixel for compositing

Alpha compositing—example

Compositing composites

- so far have only considered single fg. over single bg.
- in real applications we have $n$ layers
  - Titanic example
    - compositing foregrounds to create new foregrounds
      - what to do with $\alpha$?
    - desirable property: associativity
      $$A \text{ over } (B \text{ over } C) = (A \text{ over } B) \text{ over } C$$
      - to make this work we need to be careful about how $\alpha$ is computed
Compositing composites

- Some pixels are partly covered in more than one layer

- in \( D = A \text{ over} (B \text{ over} C) \) what will be the result?

\[
c_D = \alpha_A c_A + (1 - \alpha_A)[\alpha_B c_B + (1 - \alpha_B)c_C]
= \alpha_A c_A + (1 - \alpha_A)\alpha_B c_B + (1 - \alpha_A)(1 - \alpha_B)c_C
\]

Fraction covered by neither A nor B

An optimization

- Compositing equation again

\[
c_C = \alpha_A c_A + (1 - \alpha_A)c_B
\]

- Note \( c_A \) appears only in the product \( \alpha_A c_A \)

- so why not do the multiplication ahead of time?

- Leads to \textit{premultiplied alpha}:

- store pixel value \((r', g', b', \alpha)\) where \( c' = \alpha c \)

- \( C = A \text{ over} B \) becomes

\[
c'_C = c'_A + (1 - \alpha_A)c'_B
\]

- Turns out to be more than an optimization…

- Hint: so far the background has been opaque!

Associativity?

- What does this imply about \((A \text{ over} B)\)?

- Coverage has to be

\[
\alpha_{(A \text{ over} B)} = 1 - (1 - \alpha_A)(1 - \alpha_B)
= \alpha_A + (1 - \alpha_A)\alpha_B
\]

- ...but the color values then don’t come out nicely

in \( D = (A \text{ over} B) \text{ over} C \):

\[
c_D = \alpha_A c_A + (1 - \alpha_A)\alpha_B c_B + (1 - \alpha_A)(1 - \alpha_B)c_C
= \alpha_{(A \text{ over} B)}(\cdots) + (1 - \alpha_{(A \text{ over} B)})c_C
\]

Compositing composites

- What about just \( C = A \text{ over} B \) (with B transparent)?

- in premultiplied alpha, the result

\[
\alpha_C = \alpha_A + (1 - \alpha_A)\alpha_B
\]

looks just like blending colors, and it leads to associativity.
Associativity!

\[ c_D = c'_A + (1 - \alpha_A)[c'_B + (1 - \alpha_B)c'_C] \]
\[ = [c'_A + (1 - \alpha_A)c'_B] + (1 - \alpha_A)(1 - \alpha_B)c'_C \]
\[ = c'_{(A \text{ over } B)} + (1 - \alpha_{(A \text{ over } B)}))c'_C \]

- This is another good reason to premultiply

Independent coverage assumption

- Why is it reasonable to blend \( \alpha \) like a color?
- Simplifying assumption: covered areas are independent
  - that is, uncorrelated in the statistical sense

**Description** | **Area**
---|---
\( \bar{A} \cap \bar{B} \) | \( (1 - \alpha_A)(1 - \alpha_B) \)
\( \bar{A} \cap \bar{B} \) | \( \alpha_A(1 - \alpha_B) \)
\( \bar{A} \cup \bar{B} \) | \( (1 - \alpha_A)\alpha_B \)
\( \bar{A} \cup \bar{B} \) | \( \alpha_A\alpha_B \)

Independent coverage assumption

- Holds in most but not all cases
- This will cause artifacts
  - but we’ll carry on anyway because it is simple and usually works…

Alpha compositing—failures

- Positive correlation: too much foreground
- Negative correlation: too little foreground
Other compositing operations

- Generalized form of compositing equation:
  \[ \alpha C = A \text{ op } B \]
  \[ c = F_A a + F_B b \]

\[ \begin{array}{c|c|c|c|c|c}
\text{operation} & \text{quadruple} & \text{diagram} & F_A & F_B \\
\hline
\text{clear} & (0,0,0,0) & & 0 & 0 \\
\text{A} & (1,0,0,0) & & 1 & 0 \\
\text{B} & (0,0,1,0) & & 0 & 1 \\
A \text{ over } B & (0,1,0,0) & & 1 & 1 - \alpha_B \\
B \text{ over } A & (0,0,1,1) & & 1 - \alpha_B & 1 \\
A \text{ in } B & (0,0,0,1) & & \alpha_B & 0 \\
B \text{ in } A & (0,1,1,0) & & 0 & \alpha_B \\
A \text{ out } B & (0,1,0,1) & & 1 - \alpha_B & 0 \\
B \text{ out } A & (0,0,1,1) & & 0 & 1 - \alpha_B \\
A \text{ atop } B & (1,1,0,0) & & \alpha_B & 1 - \alpha_B \\
B \text{ atop } A & (1,1,1,0) & & \alpha_B & \alpha_B \\
A \text{ over } B & (0,0,0,1) & & 1 - \alpha_B & 1 - \alpha_B \\
\end{array} \]

\[ 1 \times 2 \times 3 \times 2 = 12 \text{ reasonable choices} \]