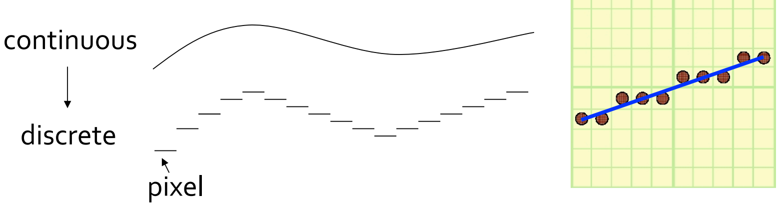
EECS 487: Interactive Computer Graphics

Lecture 8:

- Anti-aliasing
- Alpha-channel and compositing

Limitations of Raster Images

1. Limitations of discretization: limited resolution (sampling rate) and dynamic range (quantization steps (what is quantization?), bits/pixel)



Chenney,O'Brien

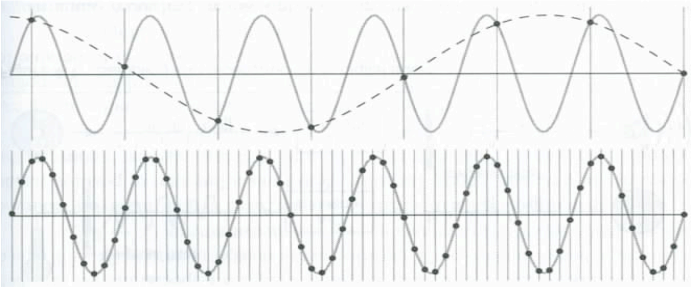
2. Computers have finite precision: real numbers are represented as float or double, causing round-off errors

Cause of Aliasing

Sampling rate (resolution) too low to capture high-frequency signal (small details in image)

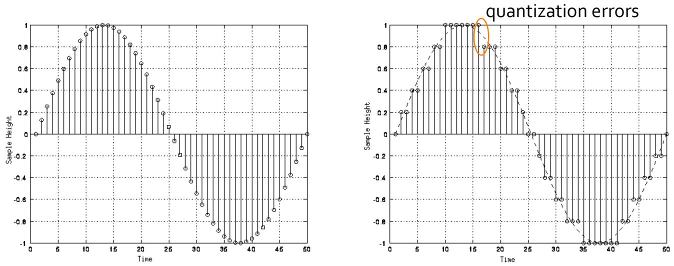
⇒ a high frequency signal reconstructed from samples as a lower frequency signal (an alias)

Nyquist Theorem: to faithfully reconstruct a signal at frequency f requires a sampling rate of $2f$



Quantization

Partitioning potential signal values into levels and represent each level with a single number

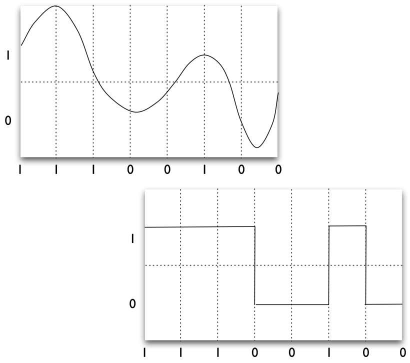


Signal Digitization

Higher sampling rate and finer quantization levels give better signal reconstruction but generates higher data rate

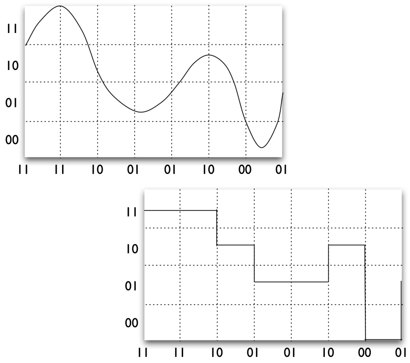
Example 1:

- sampling rate: 8 samples/sec
- quantization: 2 levels ⇒ 1 bit per sample
- data rate: 8 bps



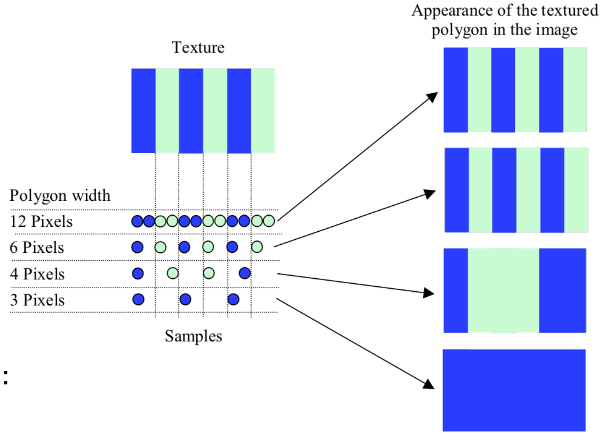
Example 2:

- sampling rate: 8 samples/sec
- quantization: 4 levels ⇒ 2 bit/sample
- data rate: 16 bps



Anti-aliasing

Approach 1: increase resolution



Problems:

Gillies&Rueckert09

Anti-aliasing

Approach 2: filtering/blurring the jaggies

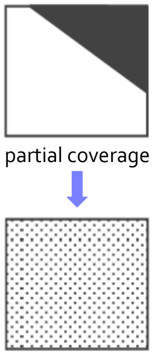
- instead of simply setting each pixel on or off,
- compute the area of a pixel to color,

⇒ translates into and implemented as how much color (intensity) to give a pixel

Alpha channel: encodes pixel coverage (or transparency) info

- $\alpha = 0$: no coverage ($=$ transparent)
- $\alpha = 1$: full coverage (or $255 =$ opaque)
- $0 < \alpha < 1$: partial coverage (or translucent)

Example:
$\alpha = 0.3$



partial coverage

translucency

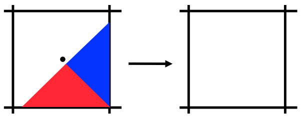
Blurring the Edges

Approaches to filtering:

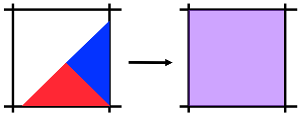
1. Area sampling (pre-(sample) filtering):
   - compute what the pixel color should be, as an average of all fragments covering the pixel
   - sample value == computed average color
   - theoretically robust, not always practical, e.g., requires computation of sub-pixel geometry
2. Super-sampling (post-(sample) filtering):
   - increase sampling frequency
   - average down multiple sample values
   - single-pass or multi-pass
   - (pre-filtering is equivalent to post-filtering with infinite samples)

one pixel

result color not to scale, example only

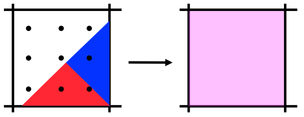


no filter: point sampling



pre-filter: area sampling

$\mathbf{c}_s = \int_{i=1}^{k} \alpha_i \mathbf{c}_i,\ \alpha_i = \frac{area(i)}{area(pixel)},\ k$ fragments



post-filter: multi-sampling

$\mathbf{c}_s = \sum_{j=1}^{n} w_j \mathbf{c}_{s_j}$, e.g., $w_j = 1/n,\ n$ samples

Wattoo

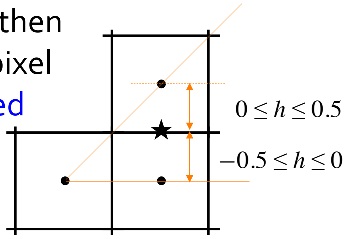
Area Sampling Example: Line Anti-Aliasing

Let $(x+1, y')$ be on the line and h the distance of y' from the midpoint $(y+0.5)$:

$$\begin{aligned} f(x+1, y+0.5) &= \Delta_x(y'+h) - \Delta_y(x+1) + C \quad // \; y'+h = y+0.5 \\ &= \Delta_x(y') + \Delta_x h - \Delta_y(x+1) + C \\ &= \Delta_x(y') - \Delta_y(x+1) + C + \Delta_x h \\ &= f(x+1, y') + h\Delta_x \\ &= 0 + h\Delta_x \qquad // \text{ assume } (x+1, y') \text{ on the line} \end{aligned}$$

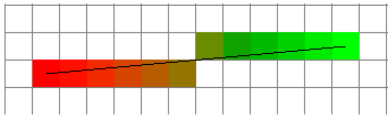
`h=fmid/dx`

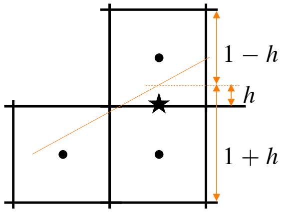
Ideally, at x, line is at pixel center, then for $0 \leq m \leq 1$, $-0.5 \leq h \leq 0.5$ and pixel area coverage can be approximated by $\alpha = |h| + 0.5$ for the "on" pixel and $1 - \alpha$ for the other pixel



Area Sampling Example: Line Anti-Aliasing

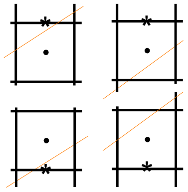
But more likely than not, the line is not at pixel center:





then $-1 \leq h \leq 1$ and we have 4 cases to consider (in PA1):

- line is below midpoint and y is not incremented:
  - 2 cases, whether line is above or below pixel center
- line is above midpoint and y is incremented:
  - 2 cases, whether line is above or below pixel center



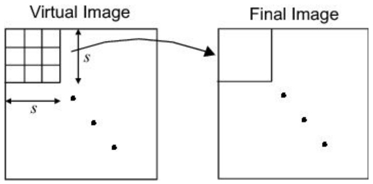
Single-pass Super-sampling

Sample at a higher resolution (higher sampling rate), into a larger buffer ⇒ requires a sampling buffer larger than the framebuffer

Then filter (average) and down sample

Example:

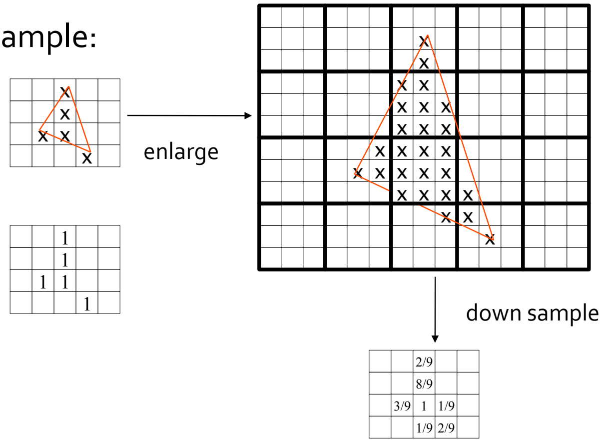
- enlarge an $N \times M$ image to $3N \times 3M$ (vertex coordinates need to be adjusted)
- take 9 (3×3) samples per pixel
- each 3×3 virtual image pixel corresponds to 1 final image pixel
- the final pixel's color is the average of the 9 samples



Single-pass Super-Sampling Triangle

Change each pixel into a 3×3 block

Example:

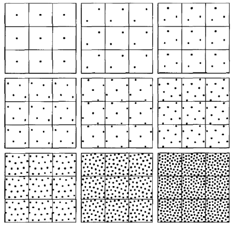


		x		
		x		
	x	x		
			x	

		1		
		1		
	1	1		
			1	

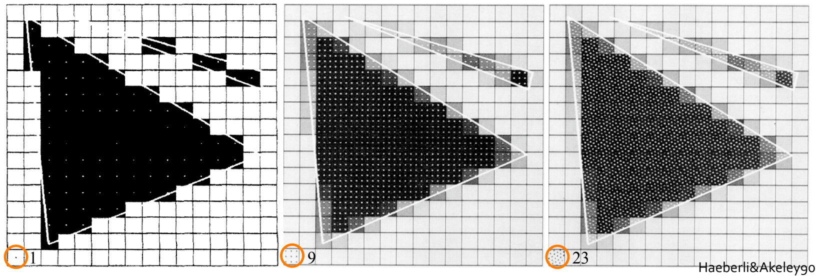
		2/9		
		8/9		
	3/9	1	1/9	
		1/9	2/9	

Multi-pass Super-sampling



Trade off time for space: instead of a sampling buffer larger than the framebuffer, sample each pixel *k* times:

- render the image *k* times in as many passes
- for each pass choose a different sub-pixel offset
- add the resulting image into a multisample buffer
- the final result is averaged out by *k* and stored in the framebuffer



Haeberli&Akeley90

Multisampling in OpenGL

First obtain a GLUT window that supports multisampling:

```
glutInitDisplayMode(… | GLUT_MULTISAMPLE);
```

Next enable multisampling before rendering:

```
glEnable(GL_MULTISAMPLE);
```

Then render as usual

Multisample buffer: an additional color buffer

- same spatial dimensions as the framebuffer
- greater color resolution and range (-1,1)

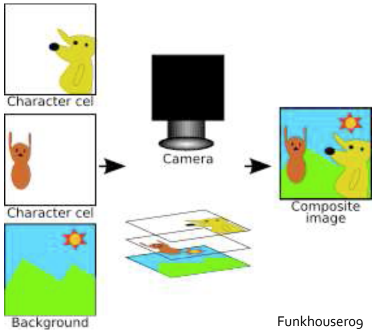
Compositing

Combines components from two or more images to make a new image

- CG FX can be done in layers
- "live action" can be faked

Classical animation technique (Disney)

- superimposition of different layers on translucent films (celluloids or "cels")



Funkhouser09

Matte

Matte: a mask used in compositing to protect a part of the background image

- paint the background image
- put on the matte
- where the matte is white, paint on the foreground object

Chenney

Blue Screen Matting

Background image (e.g., weather map) generated separately

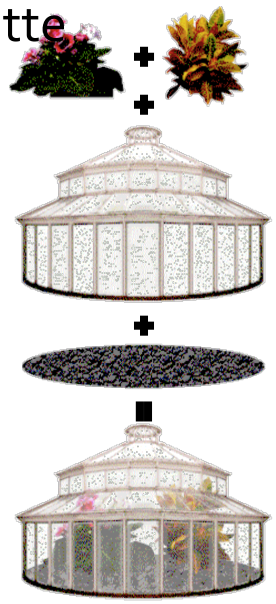
Foreground image created with blue or green background (a matte)

Insert non-blue foreground pixel into background

Really hard:

- lighting change background color
- shadows, color bleed, transparent objects
- hair is partly background
- modern system uses computer vision

Alpha Channel: Digital Matte



Lozano-Perez&Popovic01

Digital matte: store the matte along with the foreground image; pixels on the matte has full transparency, foreground pixels full opacity

Alpha channel: simulates the matting of celluloid layers

- usually process layers back to front

But α is useful for more than matting:

- to encode transparency
- to encode partial coverage for anti-aliasing!
- to encode the shape of an object ("sprite")

Pixel Coverage

Anti-aliasing and compositing both deal with pixels with ambiguous detail

- anti-aliasing: how to carefully smooth down the detail
- compositing: how to account for the detail when combining images

In both cases, simple binary "in" or "out" values are insufficient:

- causes jaggies in compositing, similar to point-sampled rasterization
- same problem, same solution: need intermediate values

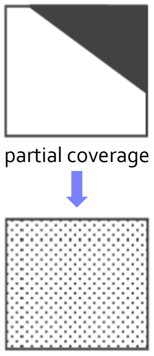
Marschner

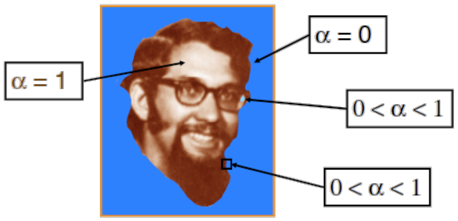
Alpha Channel

Encodes pixel coverage (or transparency) info

- $\alpha = 0$: no coverage (= transparent)
- $\alpha = 1$: full coverage (= opaque, or 255)
- $0 < \alpha < 1$: partial coverage (or translucent)
- controls the amount of foreground and background pixel colors used when linear interpolating images in composition: $\mathbf{c}_c = \alpha_f \mathbf{c}_f + (1-\alpha_f)\,\mathbf{c}_b$

Example:
$\alpha = 0.3$



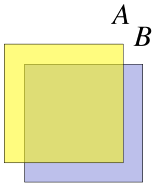


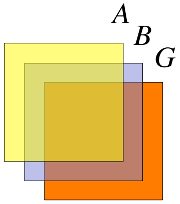
Funkhouser09

Compositing Translucent Objects

Consider the composition on the right:

- two colors: $\mathbf{c}_A$ and $\mathbf{c}_B$
- two α's: α_A and α_B
- how much of B is blocked by A?
- how much of B shows through A?
- resulting composition: C = "A over B"
  $\mathbf{c}'_C = \alpha_A\, \mathbf{c}_A + (1-\alpha_A)\, \alpha_B\, \mathbf{c}_B$
  $\alpha_C = \alpha_A + (1-\alpha_A)\, \alpha_B$
- but it turns out $\mathbf{c}'_C$ is not just the color of C, it's actually the color of C pre-multiplied by α_C, so if it's to be used in further compositing, it doesn't need to be multiplied by α_C again





Funkhouser09

Pre-Multiplied α

Instead of storing (R, G, B, α) in a pixel, store (αR, αG, αB, α)

- α value constrains color magnitude
- α shapes sprite
- mathematically clean: multiple composites are well defined

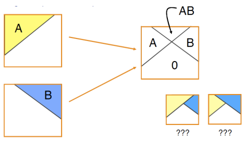
To display and operate on the actual color, extract RGB by dividing by α:

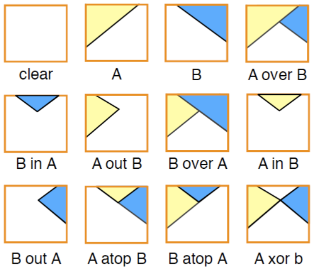
- $\alpha = 0$ is no color (or black)
- division by small α loses some precision, usually ok (and would have had to be done on composite color to extract RGB anyway)

Compositing Opaque Objects

How do we combine 2 partially covered pixels?

- "over" is not the only possible combination
- 12 reasonable combinations:





Compositing factors assuming pre-multiplied α:

Operation	F_A	F_B
Clear	0	0
A	1	0
B	0	1
A over B	1	$1-\alpha_A$
B over A	$1-\alpha_B$	1
A in B	α_B	0
B in A	0	α_A
A out B	$1-\alpha_B$	0
B out A	0	$1-\alpha_A$
A atop B	α_B	$1-\alpha_A$
B atop A	$1-\alpha_B$	α_A
A xor B	$1-\alpha_B$	$1-\alpha_A$

Example usage:

- $C = A$ over B
- $\mathbf{c}'_C = F_A\, \mathbf{c}'_A + F_B\, \mathbf{c}'_B = \mathbf{c}'_A + (1-\alpha_A)\, \mathbf{c}'_B$
- $\alpha_C = F_A\, \alpha_A + F_B\, \alpha_B = \alpha_A + (1-\alpha_A)\, \alpha_B$

Porter&Duff84

Blending and Compositing in OpenGL

First allocate space for the alpha component:

```
glutInitDisplayMode( ... | GLUT_ALPHA);
```

Then enable blending:

$$\mathbf{c}_C = \alpha_s \mathbf{c}_s + (1-\alpha_s)\mathbf{c}_d,\ \alpha_C = \alpha_s + (1-\alpha_s)\alpha_d$$

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

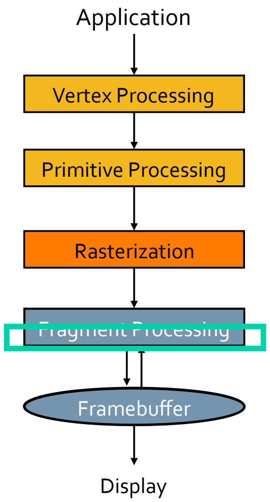
OpenGL blending factors:

- `GL_ONE, GL_ZERO`
- `GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA`
- `GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA`

Fragment Tests

Before a fragment becomes a pixel, there are several fragment tests/ merging operations to check whether the fragment belongs in the framebuffer

- a fragment may be discarded or merged with another as the result these tests



Application

Vertex Processing

Primitive Processing

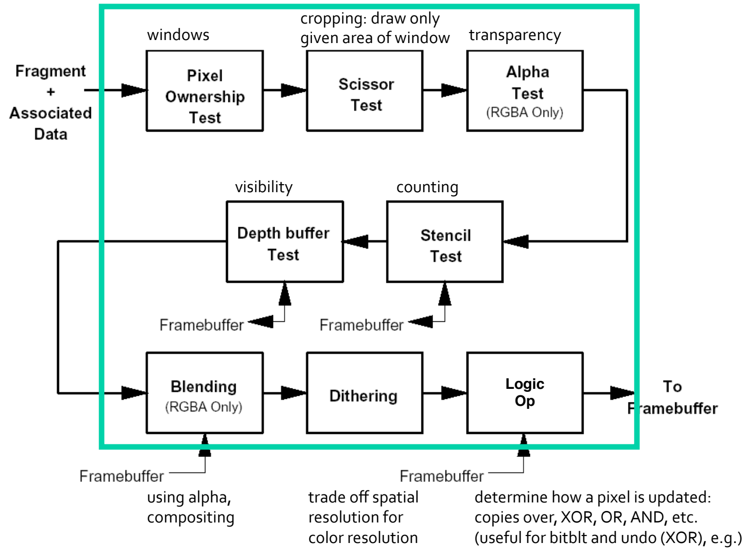
Rasterization

Fragment Processing

Framebuffer

Display

Fragment Tests



Fragment + Associated Data

windows

Pixel Ownership Test

cropping: draw only given area of window

Scissor Test

transparency

Alpha Test (RGBA Only)

counting

Stencil Test

Framebuffer

visibility

Depth buffer Test

Framebuffer

Blending (RGBA Only)

Dithering

Logic Op

To Framebuffer

Framebuffer

using alpha, compositing

trade off spatial resolution for color resolution

Framebuffer

determine how a pixel is updated: copies over, XOR, OR, AND, etc. (useful for bitblt and undo (XOR), e.g.)

Anti-aliasing vs. Visibility Culling

Coupling of visibility determination and sampling (aliasing)

- anti-aliasing blurs geometry
- accurate occlusion culling requires sampling of exact (sharply defined) geometry
- can't have both ☹

A partial solution:

- anti-aliased geometry does not occlude completely
  - occluded objects bleed through the seams
- but anti-aliased geometry can be occluded by exact geometry

Akeley07