Sampling

Chapter 16

Computer Graphics

- Concerns with all aspects of producing **pictures** or **images** *using a computer*

  rendering process

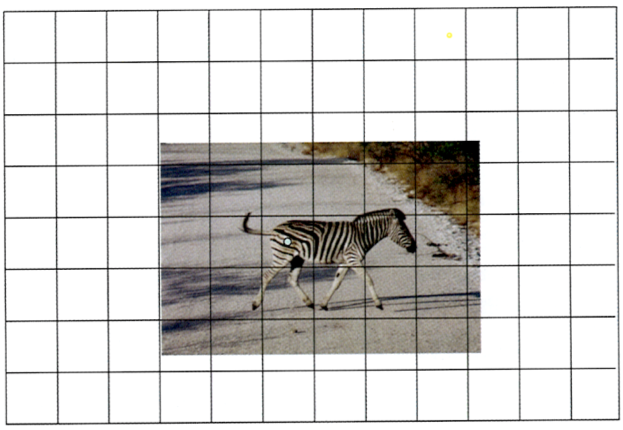
- This chapter deals with the 'image'
  - Picture → collection of pixels
    - '.. is and artifact that depicts or records visual perception'
  - Continuous image $I(x_w, y_w)$: a bivariate function
  - Discrete image `I[i][j]` : two dimensional array of color values
    - We associate each pair of integers `i,j`, with the continuous image coordinates $x_w = i$ and $y_w = j$

Sampling

- Point sampling: continuous vector → discrete pixel
- Our scenes are described with triangles giving a continuous 2D color field.
- Our images are digital/discrete made up of a grid of dots.
- Need to make a bridge between these two worlds (continuous vs. discrete).
- Else we will get some unnecessary artifacts called "aliasing" artifacts.
  - Jaggies, moire patterns, flickering

Sampling

- These occur when there is too much detail to fit in one pixel.
- We can mitigate these artifacts by averaging up the colors within a pixel's square.
- This is called *anti-aliasing.*

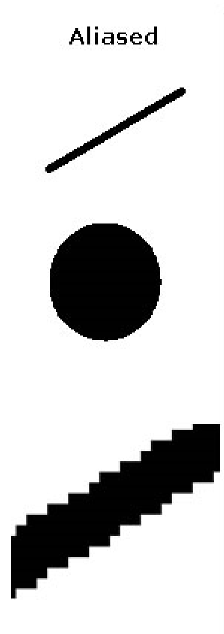


Aliasing

- The simplest and most obvious method to go from a <u>continuous</u> to a <u>discrete</u> image is by *point sampling.*
- To obtain the value of a pixel i, j, we sample the continuous image function at a single integer valued domain location: $\mathtt{I[i][j]} \leftarrow I(i,j)$
- This can results in unwanted artifacts.

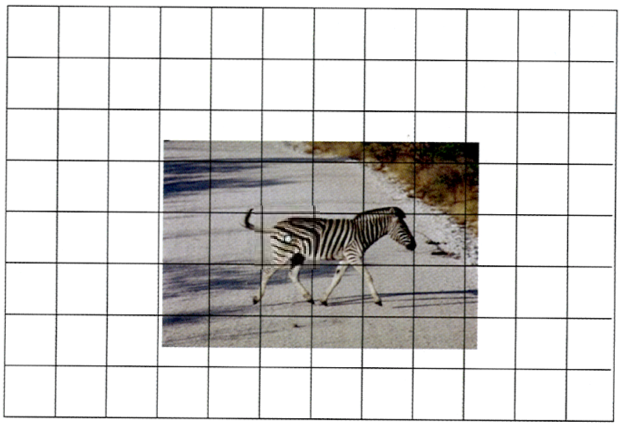
Aliasing

- Scene made up of black and white triangles: jaggies at boundaries
  - Jaggies will crawl during motion
- If triangles are small enough then we get random values or weird patterns
  - Will flicker during motion
- The heart of the problem → too much information in one pixel



Anti-aliasing

- Intuitively: the single sample is a bad value, we would be better off setting the pixel value using some kind of average value over some appropriate region.

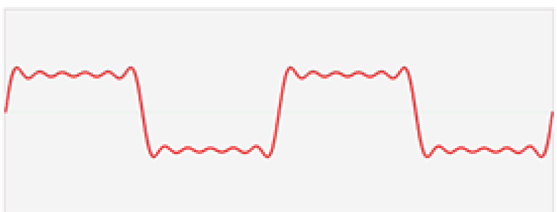


Anti-aliasing

- Mathematically this can be modeled using *Fourier analysis*.
  - Breaks up the data by “frequencies” and figures out what to do with the un-representable high frequencies.

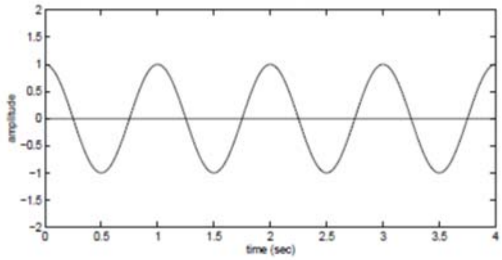
- **Nyquist Sampling Theorem**
  - “The sampling frequency should be at least twice the highest frequency contained in the signal.”

$$f_s \geq 2 f_c$$

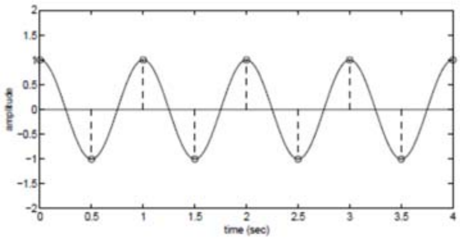


Example

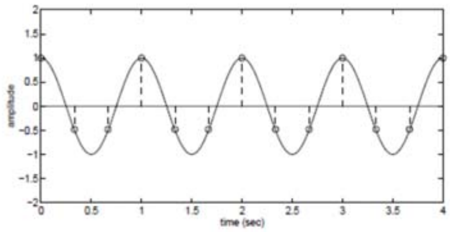
- An input signal with f = 1 Hz



- If we sample by more than 2 Hz, we can reconstruct the shape correctly



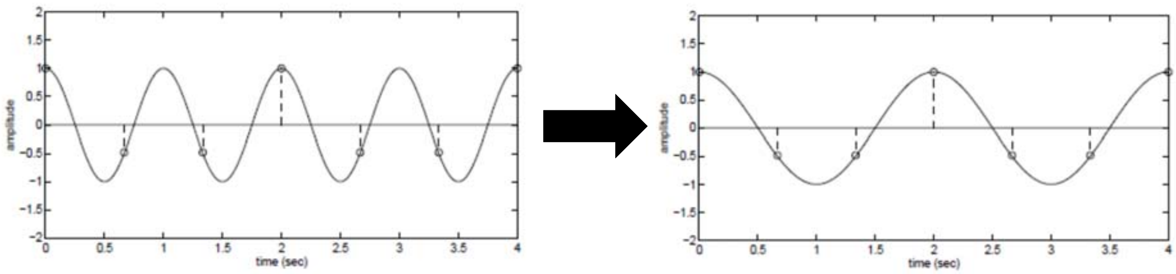
2 Hz



3 Hz

Example

- If we sample by 1.5 Hz (<= 2 Hz), there might be an ambiguity about the signal shape.



1.5 Hz

Anti-aliasing

- We can also model this as an optimization problem.
- These approaches lead to:

$$\texttt{I[i][j]} \leftarrow \int\int_{\Omega} dx\, dy\, I(x,y) F_{i,j}(x,y)$$

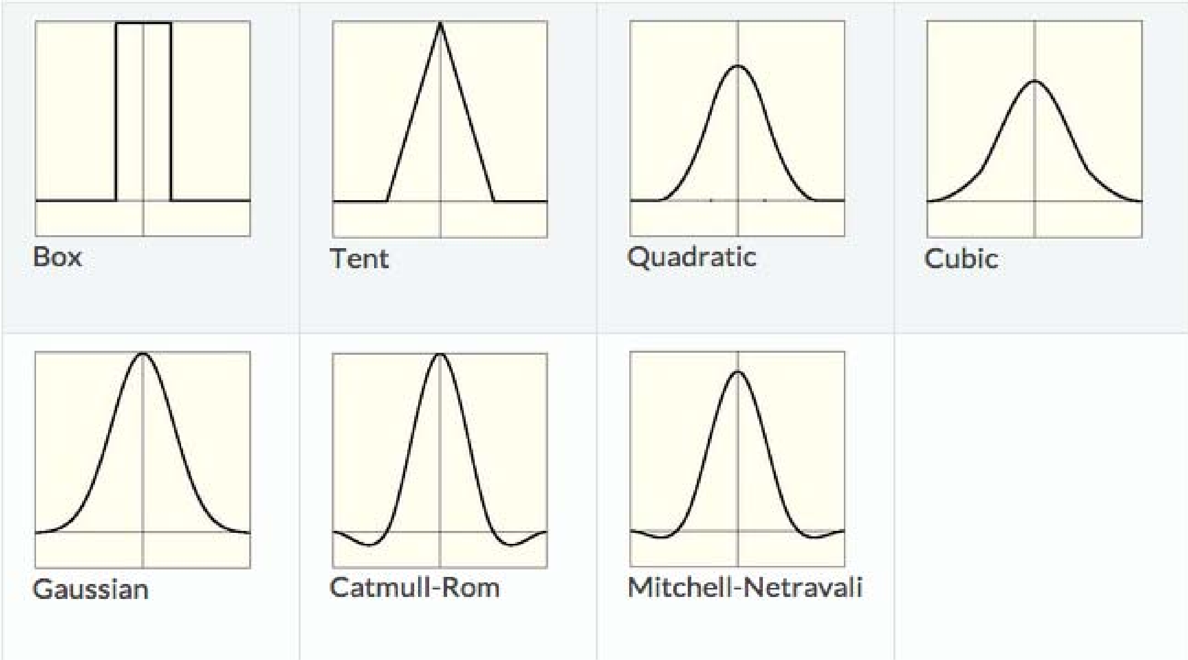
- where $F_{i,j}(x,y)$ is some function that tells us how strongly the continuous image value at $[x,y]^t$ should influence the pixel value `i.j`
- In this setting, the function $F_{i,j}(x,y)$ is called *a* *filter*.
  - In other words, the best pixel value is determined by performing some continuous weighted averaging near the pixel's location.
  - Effectively, this is like blurring the continuous image before point sampling it.

Box Filter

- We often choose the filters $F_{i,j}(x,y)$ to be something non-optimal, but that can more easily computed with.
- The simplest such choice is a *box filter*, where $F_{i,j}(x,y)$ is zero everywhere except over the 1-by-1 square center at $x = i, y = j$.

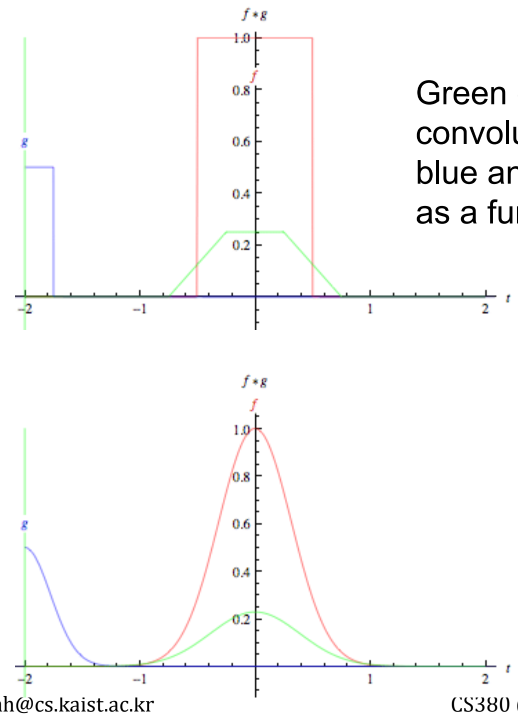
- Calling this square $\Omega_{i,j}$, we arrive at $$\texttt{I[i][j]} \leftarrow \int\int_{\Omega_{i,j}} dx\, dy\, I(x,y)$$
- In this case, the desired pixel value is simply the average of the continuous image over the pixel's square domain.

Filters



convolution

http://mathworld.wolfram.com/Convolution.html



Green curve is the convolution of the blue and red curves as a function of t.

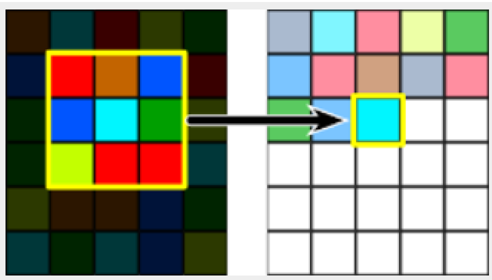


Diagram 1: The source pixel and its surrounding pixels are all mathematically merged to produce a single destination pixel. The matrix slides across the surface of the source image, producing pixels for the destination image.

https://en.wikipedia.org/wiki/Kernel_(image_processing)

convolution

- identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- box blur

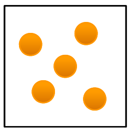
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Over-sampling

- Even that integral $\texttt{I[i][j]} \leftarrow \int\int_{\Omega_{i,j}} dx\, dy\, I(x,y)$ is not really reasonable to compute.
- Instead, it is approximated by some sum of the form:

$$\texttt{I[i][j]} \leftarrow \frac{1}{n}\sum_{k=1}^{n} I(x_k, y_k)$$

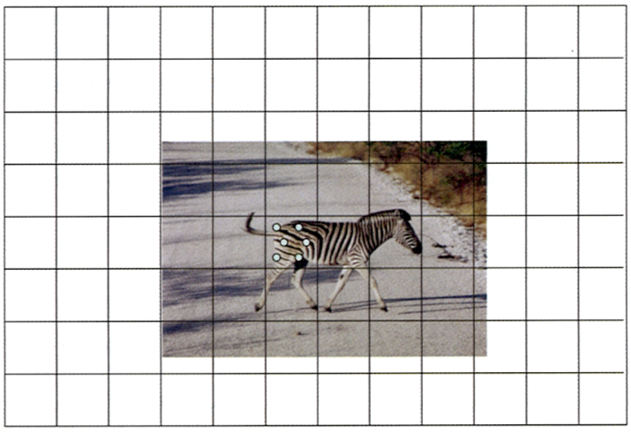
a pixel



where *k* indexes some set of locations (x_k, y_k) called the sample locations.
- The renderer first produces a “high resolution” color and z-buffer “image”,
  - where we will use the term *sample* to refer to each of these high resolution pixels.

Over-sampling

- Then, once rasterization is complete, groups of these samples are averaged together, to create the final lower resolution image.



Super-sampling

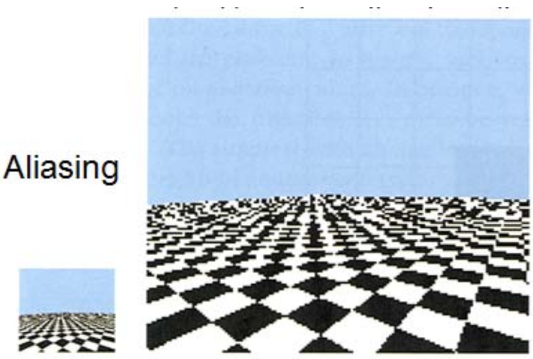
- If the sample locations for the high resolution image form a regular, high resolution grid, then this is called *super sampling.*
- We can also choose other sampling patterns for the high resolution "image",
  - Such less regular patterns can help us avoid systematic errors that can arise when using the sum to replace the integral.

Multi-sampling

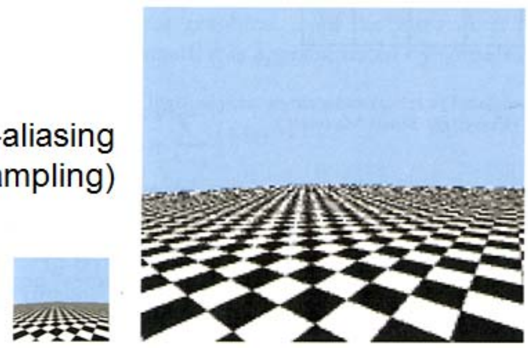
- In OpenGL, we can also choose to do *multisampling*.
- Render to a "high resolution" color and z-buffer
- *During the rasterization* of each triangle, "coverage" and z-values are computed at this (high)sample level.
  - But for efficiency, the fragment shader is only called only once per final resolution pixel.
  - This color data is shared between all of the samples hit by the triangle in a single (final resolution) pixel.
  - Once rasterization is complete, groups of these high resolution samples are averaged together.
- Multisampling can be an effective anti-aliasing method since, without texture mapping, colors tend to vary quite slowly over each triangle, and thus they do not need to be computed at high spatial resolution.
  - → Mipmapping (next chapter)

Aliasing vs. anti-aliasing

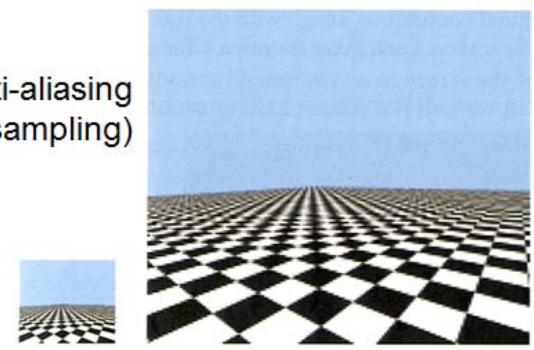
Aliasing



Anti-aliasing (multi-sampling)



Anti-aliasing (super-sampling)



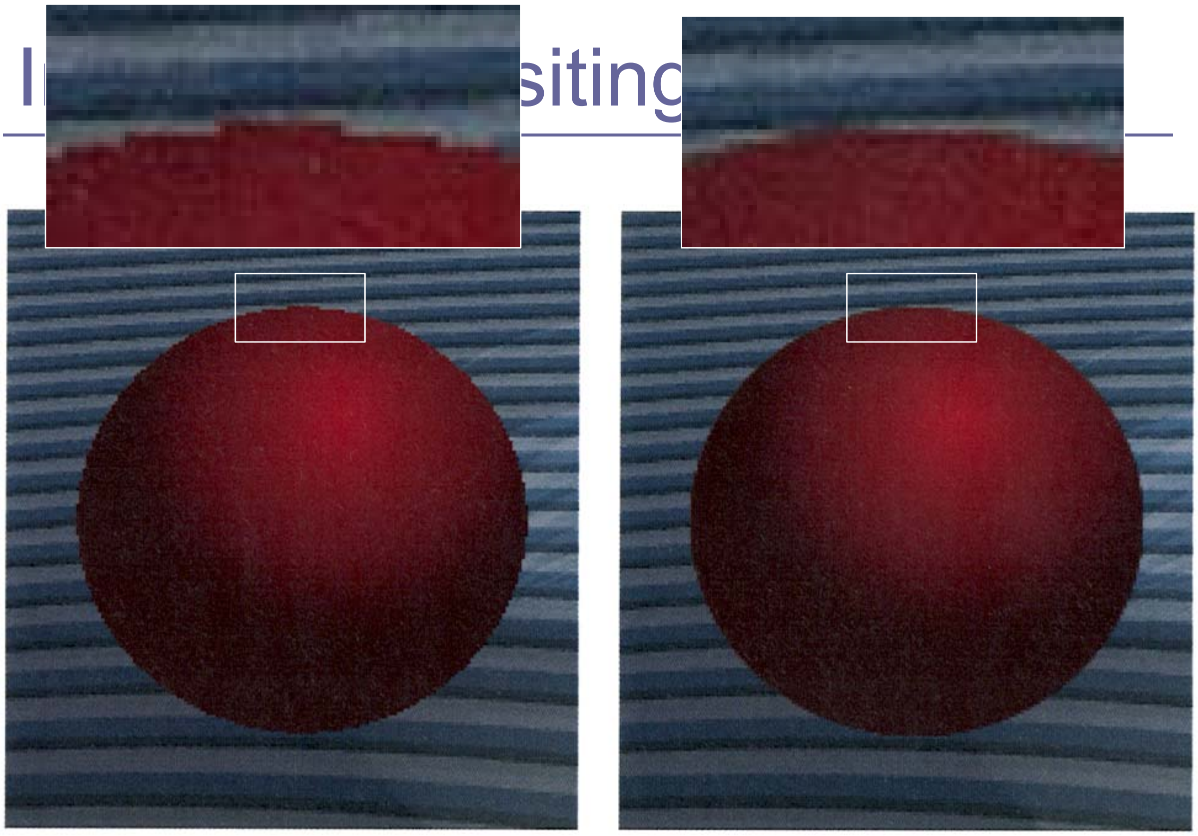
Camera

- In digital cameras, anti-aliasing is accomplished by a combination of the spatial integration that happens over the extent of *each pixel sensor*, as well as by the *optical blurring* that happens at due to the lens.
- Some cameras also include additional optical elements specifically to blur the continuous image data before it is sampled at the sensors.

Image compositing

- Given two discrete images, a foreground, I^f, and background, I^b, that we want to combine into one image I^c.
- Simple: in composite, use foreground pixels where they are defined. Else use background pixels.
  - This will give us a jagged boundary.
- Real image would have "boundary" pixels with blended colors.
  - But this requires using "sub-pixel" information.

I siting



Alpha blending

- Associate with each pixel in each image layer, a value $\alpha[\mathtt{i}][\mathtt{j}]$ that describes the overall *opacity* or *coverage* of the image layer at that pixel.
  - An alpha value of 1 represents a fully opaque/occupied pixel, while a value of 0 represents a fully transparent/empty one.
  - A fractional value represents a partially transparent (partially occupied) pixel.
- Alpha will be used during compositing.

- $I(x,y)$ continuous image
- $C(x,y)$ binary valued covera (x,y) domain (1 “occupie

$$\mathtt{I}[\mathtt{i}][\mathtt{j}] \leftarrow \int\int_{\Omega_{i,j}} dx\, dy\, I(x,y)C(x,y)$$

$$\alpha[\mathtt{i}][\mathtt{j}] \leftarrow \int\int_{\Omega_{i,j}} dx\, dy\, C(x,y)$$

Over operation

- To compose $I^f[i][j]$ *over* $I^b[i][j]$
  we compute the composite image colors $I^c[i][j]$ using

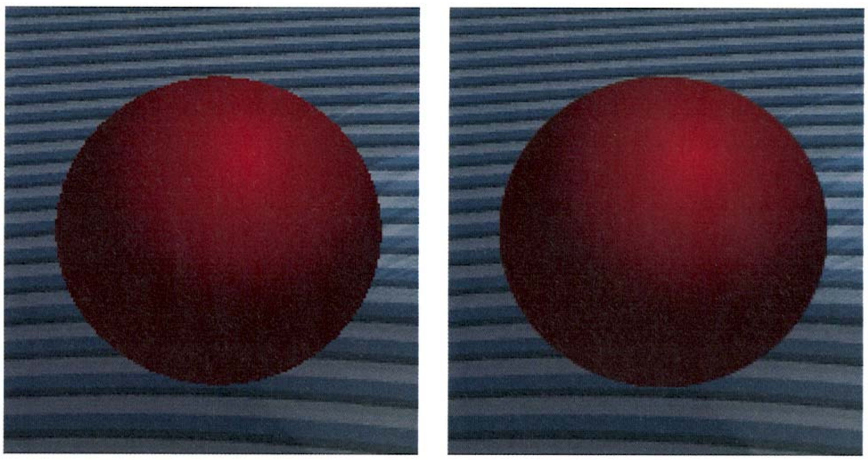
$$I^c[i][j] \leftarrow I^f[i][j] + I^b[i][j](1-\alpha^f[i][j])$$

  - the amount of observed background color at a pixel is proportional to the transparency of the foreground layer at that pixel.
  - Alpha for the composite image is computed as

$$\alpha^c[i][j] \leftarrow \alpha^f[i][j] + \alpha^b[i][j](1-\alpha^f[i][j])$$

Over operation

- If background is opaque, so the composite pixel is opaque.
- But we can model more general case as part of blending multiple layers.
  - a reasonable approximation to the correctly rendered image.



Over properties

- over operation is associative

$$I^a \text{ over } (I^b \text{ over } I^c) = (I^a \text{ over } I^b) \text{ over } I^c$$

but not commutative.

$$I^a \text{ over } I^b \neq I^b \text{ over } I^a$$

In Practice

- In OpenGL, alpha is used not just for image compositing, but in more general sense as tool for modeling transparency and for blending color values.
- (R,G, B, A) → `fragColor` in fagment shader
  - `glEnable(GL_BLEND)`
  - `glBlendFunc`

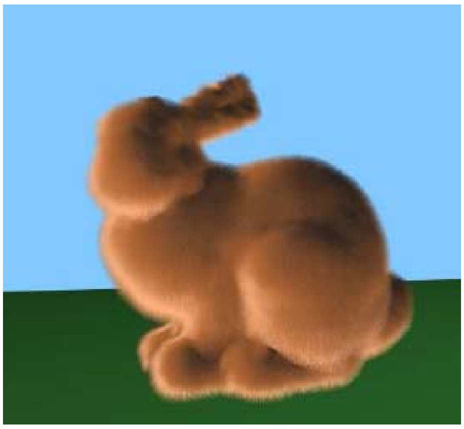
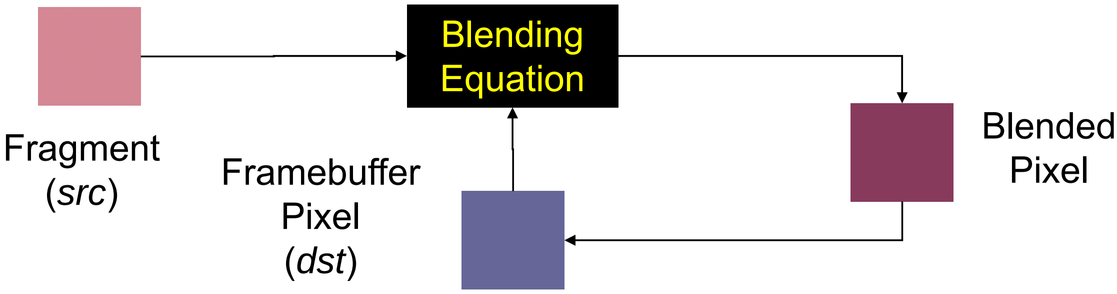


Figure 16.5:
A furry bunny is drawn as a series of larger and larger concentric bunnies. each is partially transparent. Amazingly, this looks like fur.

Blending

- Combine pixels with what's in already in the framebuffer
- In OpenGL
  - glEnable(GL_BLEND)
  - glBlendFunc(source_factor, destination_factor)

    GL_ONE, GL_ZERO, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA

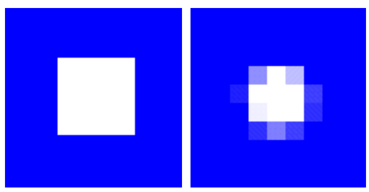


Compositing Techniques

- **Alpha blending**

Some Applications

- **Multipass rendering**
  - Blending allows results from multiple drawing passes to be combined together
- **Image compositing**
  - merge a set of images into a single image
- **Antialiasing**
  - alpha value computed by computing sub-pixel coverage
- **Depth Cueing and Fog**
  - C: color, f: fog factor



$$C_{s'} = f\, C_s + (1-f)\, C_f$$

<fog demo>

Multipass Rendering

- Motion blur

- Depth of field

Today's animation

- PGi-13
  - Siggraph *2004*
  - Parental Guidance for Certain Imaginations for Children under the age of 13….
    A scary imagination comes from a sudden curiosity about the materials of a tea bag before put into a cup of hot water. …
  - Average CPU time for rendering per frame: 18 minutes (10 min ~ 3 hours).
  - Many use of alpha-channel sequences were needed.