### Sampling and Antialiasing

#### Undersampling
- What if we “missed” things between the samples?
- Simple example: undersampling a sine wave
  - unsurprising result: information is lost
  - surprising result: indistinguishable from lower frequency
  - also was always indistinguishable from higher frequencies
- aliasing: signals “traveling in disguise” as other frequencies

#### Preventing aliasing
- Introduce lowpass filters:
  - remove high frequencies leaving only safe, low frequencies
  - choose lowest frequency in reconstruction (disambiguate)

#### Filters
- Sequence of weights $a[j]$ is called a filter
- Filter is nonzero over its region of support
  - usually centered on zero: support radius $r$
- Filter is normalized so that it sums to 1.0
  - this makes for a weighted average, not just any old weighted sum
- Most filters are symmetric about 0
  - since for images we usually want to treat left and right the same

#### Convolution and filtering
- Can express sliding average as convolution with a box filter
- $a_{box} = [..., 0, 1, 1, 1, 1, 1, 0, ...]$

#### Example: box and step

**Convolution and filtering**

- Convolution applies with any sequence of weights
- Example: Bell curve (Gaussian-like) \([..., 1, 4, 6, 4, 1, ...]/16\)

**A gallery of filters**

- **Box filter**
  - Simple and cheap
- **Tent filter**
  - Linear interpolation
- **Gaussian filter**
  - Very smooth antialiasing filter
- **B-spline cubic**
  - Very smooth
- ...

**Box filter**

\[
o_{\text{Box}, r}[i] = \begin{cases} 
1/(2r+1) & |i| \leq r, \\
0 & \text{otherwise}
\end{cases}
\]

\[
f_{\text{Box}, r}(x) = \begin{cases} 
1/(2r) & -r \leq x < r, \\
0 & \text{otherwise}
\end{cases}
\]

**Tent filter**

\[
f_{\text{Tent}}(x) = \begin{cases} 
1 - |x| & |x| < 1, \\
0 & \text{otherwise}
\end{cases}
\]

\[
f_{\text{Tent}, r}(x) = \frac{f_{\text{Tent}}(x/r)}{r}
\]

**Gaussian filter**

\[
f_{\text{G}}(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}
\]

**Resampling**

- Changing the sample rate
  - in images, this is enlarging and reducing
- Creating more samples:
  - increasing the sample rate
  - “upsampling”
  - “enlarging”
- Ending up with fewer samples:
  - decreasing the sample rate
  - “downsampling”
  - “reducing”
Reducing and enlarging

- Very common operation
  - devices have differing resolutions
  - applications have different memory/quality tradeoffs
- Also very commonly done poorly
- Simple approach: drop/replicate pixels
- Correct approach: use resampling

Types of artifacts

- Garden variety
  - what we saw in this natural image
  - fine features become jagged or sparkle
- Moiré patterns
  - caused by repetitive patterns in input
  - produce large-scale artifacts; highly visible
- These artifacts are aliasing just like in the audio example earlier

How do I know what filter is best at preventing aliasing?

- practical answer: experience
- theoretical answer: there is another layer of cool math behind all this

Antialiasing
**Aliasing**

Point sampling a continuous image:

- Continuous image defined by ray tracing procedure
- Continuous image defined by a bunch of black rectangles

**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
  - 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

Antialiasing

- 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

Gaussian filtering in action

Filter comparison

Point sampling  Box filtering  Gaussian filtering

Antialiasing in ray tracing

- aliased image

Antialiasing in ray tracing

- aliased image
- one sample per pixel
Antialiasing in ray tracing

![Antialiased image with four samples per pixel]

Details of supersampling

- For image coordinates with integer pixel centers:

```c
// 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));
```

```c
// 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));
```

Mipmap image pyramid

![Mipmap image pyramid with d axis]

Antialiasing in textures

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

Texture minification

![Texture minification with point sampled minification and mipmap minification]

Antialias in ray tracing

![Antialiased image with one sample per pixel and nine samples per pixel]
Texture minification

mipmap minification

higher quality minification