Scaled representations

- Big bars (resp. spots, hands, etc.) and little bars are both interesting
  - Stripes and hairs, say
- Inefficient to detect big bars with big filters
  - And there is superfluous detail in the filter kernel

- Alternative:
  - Apply filters of fixed size to images of different sizes
  - Typically, a collection of images whose edge length changes by a factor of 2 (or root 2)
  - This is a pyramid (or Gaussian pyramid) by visual analogy
Aliasing

- Can’t shrink an image by taking every second pixel
- If we do, characteristic errors appear
  - In the next few slides
  - Typically, small phenomena look bigger; fast phenomena can look slower
  - Common phenomenon
    - Wagon wheels rolling the wrong way in movies
    - Checkerboards misrepresented in ray tracing
    - Striped shirts look funny on color television

Resample the checkerboard by taking one sample at each circle. In the case of the top left board, new representation is reasonable. Top right also yields a reasonable representation. Bottom left is all black (dubious) and bottom right has checks that are too big.
Constructing a pyramid by taking every second pixel leads to layers that badly misrepresent the top layer.

Open questions

- What causes the tendency of differentiation to emphasize noise?
- In what precise respects are discrete images different from continuous images?
- How do we avoid aliasing?
- General thread: a language for fast changes
  The Fourier Transform
The Fourier Transform

- Represent function on a new basis
  - Think of functions as vectors, with many components
  - We now apply a linear transformation to transform the basis
    - dot product with each basis element
- In the expression, u and v select the basis element, so a function of x and y becomes a function of u and v
- basis elements have the form
  \[ e^{i(2\pi uy + v)} \]

The function is a sinusoid with this frequency along the direction, and constant perpendicular to the direction.

\[
F(g(x,y))(u,v) = \int \int_{\mathbb{R}^2} g(x,y) e^{i(2\pi uy + v)} dxdy
\]
Here $u$ and $v$ are larger than in the previous slide.

And larger still...
Phase and Magnitude

- Fourier transform of a real function is complex
  - difficult to plot, visualize
  - instead, we can think of the phase and magnitude of the transform
- Phase is the phase of the complex transform
- Magnitude is the magnitude of the complex transform
- Curious fact
  - all natural images have about the same magnitude transform
  - hence, phase seems to matter, but magnitude largely doesn’t
- Demonstration
  - Take two pictures, swap the phase transforms, compute the inverse - what does the result look like?
This is the magnitude transform of the cheetah pic.

This is the phase transform of the cheetah pic.
This is the magnitude transform of the zebra pic
This is the phase transform of the zebra pic.

Reconstruction with zebra phase, cheetah magnitude
Various Fourier Transform Pairs

- Important facts
  - The Fourier transform is linear
  - There is an inverse FT
  - If you scale the function’s argument, then the transform’s argument scales the other way. This makes sense --- if you multiply a function’s argument by a number that is larger than one, you are stretching the function, so that high frequencies go to low frequencies
  - The FT of a Gaussian is a Gaussian.

- The convolution theorem
  - The Fourier transform of the convolution of two functions is the product of their Fourier transforms
  - The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms

- There’s a table in the book.
Sampling in 1D takes a continuous function and replaces it with a vector of values, consisting of the function’s values at a set of sample points. We’ll assume that these sample points are on a regular grid, and can place one at each integer for convenience.

Sampling in 2D does the same thing, only in 2D. We’ll assume that these sample points are on a regular grid, and can place one at each integer point for convenience.
A continuous model for a sampled function

- We want to be able to approximate integrals sensibly
- Leads to
  - the delta function
  - model on right

\[
\text{Sample}_{2D}(f(x, y)) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(x, y) \delta(x - i, y - j)
\]

\[
= f(x, y) \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \delta(x - i, y - j)
\]

The Fourier transform of a sampled signal

\[
F(\text{Sample}_{2D}(f(x, y))) = F \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(x, y) \delta(x - i, y - j)
\]

\[
= F(f(x, y)) \ast \ast F \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \delta(x - i, y - j)
\]

\[
= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} F(u - i, v - j)
\]
Smoothing as low-pass filtering

- The message of the FT is that high frequencies lead to trouble with sampling.
- Solution: suppress high frequencies before sampling
  - multiply the FT of the signal with something that suppresses high frequencies
  - or convolve with a low-pass filter
- A filter whose FT is a box is bad, because the filter kernel has infinite support
- Common solution: use a Gaussian
  - multiplying FT by Gaussian is equivalent to convolving image with Gaussian.

Sampling without smoothing. Top row shows the images, sampled at every second pixel to get the next; bottom row shows the magnitude spectrum of these images.
Sampling with smoothing. Top row shows the images. We get the next image by smoothing the image with a Gaussian with sigma 1 pixel, then sampling at every second pixel to get the next; bottom row shows the magnitude spectrum of these images.

<table>
<thead>
<tr>
<th>Image Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>256x256</td>
</tr>
<tr>
<td>128x128</td>
</tr>
<tr>
<td>64x64</td>
</tr>
<tr>
<td>32x32</td>
</tr>
<tr>
<td>16x16</td>
</tr>
</tbody>
</table>

Sampling with smoothing. Top row shows the images. We get the next image by smoothing the image with a Gaussian with sigma 1.4 pixels, then sampling at every second pixel to get the next; bottom row shows the magnitude spectrum of these images.

<table>
<thead>
<tr>
<th>Image Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>256x256</td>
</tr>
<tr>
<td>128x128</td>
</tr>
<tr>
<td>64x64</td>
</tr>
<tr>
<td>32x32</td>
</tr>
<tr>
<td>16x16</td>
</tr>
</tbody>
</table>
Applications of scaled representations

- Search for correspondence
  - look at coarse scales, then refine with finer scales
- Edge tracking
  - a “good” edge at a fine scale has parents at a coarser scale
- Control of detail and computational cost in matching
  - e.g. finding stripes
  - terribly important in texture representation

The Gaussian pyramid

- Smooth with gaussians, because
  - a gaussian*gaussian=another gaussian
- Synthesis
  - smooth and sample
- Analysis
  - take the top image
- Gaussians are low pass filters, so representation is redundant
Texture

- Key issue: representing texture
  - Texture based matching
    - little is known
  - Texture segmentation
    - key issue: representing texture
  - Texture synthesis
    - useful; also gives some insight into quality of representation
  - Shape from texture
    - cover superficially
Representing textures

- Textures are made up of quite stylized subelements, repeated in meaningful ways
- Representation:
  - find the subelements, and represent their statistics
- But what are the subelements, and how do we find them?
  - recall normalized correlation
  - find subelements by applying filters, looking at the magnitude of the response

- What filters?
  - experience suggests spots and oriented bars at a variety of different scales
  - details probably don’t matter
- What statistics?
  - within reason, the more the merrier.
  - At least, mean and standard deviation
  - better, various conditional histograms.
Gabor filters at different scales and spatial frequencies

top row shows anti-symmetric (or odd) filters, bottom row the symmetric (or even) filters.

The Laplacian Pyramid

- **Synthesis**
  - preserve difference between upsampled Gaussian pyramid level and Gaussian pyramid level
  - band pass filter - each level represents spatial frequencies (largely) unrepresented at other levels
- **Analysis**
  - reconstruct Gaussian pyramid, take top layer
Oriented pyramids

- Laplacian pyramid is orientation independent
- Apply an oriented filter to determine orientations at each layer
  - by clever filter design, we can simplify synthesis
  - this represents image information at a particular scale and orientation
Filter Kernels

Image

Coarsest scale

Finest scale

Final texture representation

- Form an oriented pyramid (or equivalent set of responses to filters at different scales and orientations).
- Square the output
- Take statistics of responses
  - e.g. mean of each filter output (are there lots of spots)
  - std of each filter output
  - mean of one scale conditioned on other scale having a particular range of values (e.g. are the spots in straight rows?)
Texture synthesis

- Use image as a source of probability model
- Choose pixel values by matching neighborhood, then filling in
- Matching process
  - look at pixel differences
  - count only synthesized pixels

Variations

- Texture synthesis at multiple scales
- Texture synthesis on surfaces
- Texture synthesis by tiles
- “Analogous” texture synthesis