Today’s lecture ...

- More on Image Filtering
- Additional transformations
Readings

- Chapter 2 and 3 of Rich Szeliski’s book

- Available online here
Image Sub-Sampling
Image Sub-Sampling

- Throw away every other row and column to create a 1/2 size image

[Source: S. Seitz]
Image Sub-Sampling

- Why does this look so crufty?

![Image Sub-Sampling Example]

1/2 1/4 (2x zoom) 1/8 (4x zoom)

[Source: S. Seitz]
Image Sub-Sampling

[Source: F. Durand]
Even worse for synthetic images

- What’s happening?

[Source: L. Zhang]
Aliasing

- Occurs when your sampling rate is not high enough to capture the amount of detail in your image

- To do sampling right, need to understand the structure of your signal/image
Aliasing

- Occurs when your sampling rate is not high enough to capture the amount of detail in your image.

![Aliasing Graph]

- To do sampling right, need to understand the structure of your signal/image.
- The minimum sampling rate is called the Nyquist rate.
Aliasing

- Occurs when your sampling rate is not high enough to capture the amount of detail in your image

To do sampling right, need to understand the structure of your signal/image

- The minimum sampling rate is called the **Nyquist rate**
Shannon's Sampling Theorem shows that the minimum sampling

\[ f_s \geq 2f_{\text{max}} \]

If you haven’t seen this... take a class on Fourier analysis... everyone should have at least one!

Figure: example of a 1D signal [R. Szeliski et al.]
Nyquist limit 2D example

Good sampling

Bad sampling

[Source: N. Snavely]
Going back to Downsampling ...

- When downsampling by a factor of two, the original image has frequencies that are too high.

- How can we fix this?
Going back to Downsampling ...

- When downsampling by a factor of two, the original image has frequencies that are too high
- How can we fix this?
Gaussian pre-filtering

- Solution: filter the image, then subsample

Gaussian 1/2

G 1/4

G 1/8

[Source: S. Seitz]
Subsampling with Gaussian pre-filtering

G 1/2

G 1/4

G 1/8

[Source: S. Seitz]
Compare with ...

1/2  1/4  (2x zoom)  1/8  (4x zoom)

[Source: S. Seitz]
Figure: (a) Example of a 2D signal. (b–d) downscaled with different filters

[Source: R. Szeliski]
Gaussian pre-filtering

- Solution: filter the image, then subsample

[Source: N. Snavely]
Gaussian pre-filtering

Gaussian pyramid

Diagram showing the process of Gaussian pre-filtering with images denoted as $F_0$, $F_1$, $F_2$, etc., undergoing blur and subsampling steps.
Gaussian Pyramids [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to wavelet transform

How much space does a Gaussian pyramid take compared to the original image?

[Source: S. Seitz]
Gaussian Pyramids [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to wavelet transform

![Diagram of Gaussian Pyramid]

- How much space does a Gaussian pyramid take compared to the original image?

[Source: S. Seitz]
Example of Gaussian Pyramid

[Source: N. Snavely]
Decimation or Sub-sampling

- **Decimation**: reduces resolution

\[ g(i,j) = \sum_{k,l} f(k,l) h(i - k/r, j - l/r) \]

with \( r \) the down-sampling rate.

- Different filters exist to do this.
Decimation or Sub-sampling

- **Decimation**: reduces resolution

\[
g(i, j) = \sum_{k,l} f(k, l) h(i - k/r, j - l/r)
\]

with \( r \) the down-sampling rate.

- Different filters exist to do this.
- What would you use?
Decimation or Sub-sampling

- **Decimation**: reduces resolution

  \[ g(i, j) = \sum_{k,l} f(k, l) h(i - k/r, j - l/r) \]

  with \( r \) the down-sampling rate.

- Different filters exist to do this.

- What would you use?
Image Up-Sampling
Image Up-Sampling

- This image is too small, how can we make it 10 times as big?

- Simplest approach: repeat each row and column 10 times (Nearest neighbor interpolation)

[Source: N. Snavely]
Recall how a digital image is formed

\[ F[x, y] = \text{quantize}\{f(xd, yd)\} \]

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale
Recall how a digital image is formed

\[ F[x, y] = \text{quantize}\{f(xd, yd)\} \]

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

(Source: N. Snavely, S. Seitz)
Recall how a digital image is formed

\[ F[x, y] = \text{quantize}\{f(xd, yd)\} \]

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

[Source: N. Snavely, S. Seitz]
What if we don’t know $f$?

- Guess an approximation: Can be done in a principled way via filtering

$d = 1$ in this example
What if we don’t know $f$?

- Guess an approximation: Can be done in a principled way via filtering
- Convert $F$ to a continuous function

$f_F(x) = \begin{cases} 
F\left(\frac{x}{d}\right) & \text{if } \frac{x}{d} \text{ is an integer} \\
0 & \text{otherwise}
\end{cases}$
What if we don’t know $f$?

- Guess an approximation: Can be done in a principled way via filtering
- Convert $F$ to a continuous function

$$f_F(x) = \begin{cases} F\left(\frac{x}{d}\right) & \text{if } \frac{x}{d} \text{ is an integer} \\ 0 & \text{otherwise} \end{cases}$$

- Reconstruct by convolution with a reconstruction filter, $h$

$$\hat{f} = h \ast f_F$$

[Source: N. Snavely, S. Seitz]
Image Interpolation

What if we don’t know $f$?

- Guess an approximation: Can be done in a principled way via filtering
- Convert $F$ to a continuous function
  \[
  f_F(x) = \begin{cases} 
  F \left( \frac{x}{d} \right) & \text{if } \frac{x}{d} \text{ is an integer} \\
  0 & \text{otherwise}
  \end{cases}
  \]

- Reconstruct by convolution with a reconstruction filter, $h$
  \[
  \hat{f} = h \ast f_F
  \]

[Source: N. Snavely, S. Seitz]
Image Interpolation

$sinc(x)$ \(\Rightarrow\) “Ideal” reconstruction

$I(x)$ \(\Rightarrow\) Nearest-neighbor interpolation

$\Lambda(x)$ \(\Rightarrow\) Linear interpolation

gauss($x$) \(\Rightarrow\) Gaussian reconstruction

Source: B. Curless
Reconstruction filters

- What does the 2D version of this hat function look like?

\[ h(x) \]

\[ h(x, y) \]

- Performs linear interpolation
- (tent function) performs \textbf{bilinear interpolation}

- Often implemented without cross-correlation, e.g.,
Reconstruction filters

- What does the 2D version of this hat function look like?

$$h(x)$$ performs linear interpolation

$$h(x, y)$$ (tent function) performs \textit{bilinear interpolation}

- Often implemented without cross-correlation, e.g.,
  \url{http://en.wikipedia.org/wiki/Bilinear_interpolation}

- Better filters give better resampled images: Bicubic is a common choice
Reconstruction filters

- What does the 2D version of this hat function look like?

\[ h(x) \quad \text{performs linear interpolation} \]

\[ h(x, y) \quad \text{(tent function) performs \underline{bilinear interpolation}} \]

- Often implemented without cross-correlation, e.g.,
  http://en.wikipedia.org/wiki/Bilinear_interpolation

- Better filters give better resampled images: Bicubic is a common choice
Image Interpolation

Original image

Interpolation results

Nearest-neighbor interpolation  Bilinear interpolation  Bicubic interpolation

[Source: N. Snavely]
What operation have we done?

Also used for *resampling*

[Source: N. Snavely]
Published by [Kopt et al., SIGGRAPH 2011]
More Examples
When are Pyramids Useful?

- We might want to **change resolution** of an image before processing.
- We might **not know which scale** we want, e.g., when searching for a face in an image.
When are Pyramids Useful?

- We might want to **change resolution** of an image before processing.
- We might **not know which scale** we want, e.g., when searching for a face in an image.
  - In this case, we will generate a full pyramid of different image sizes.
When are Pyramids Useful?

- We might want to **change resolution** of an image before processing.
- We might **not know which scale** we want, e.g., when searching for a face in an image.
  - In this case, we will generate a full pyramid of different image sizes.
- Can also be used to **accelerate the search**, by first finding at the coarser level of the pyramid and then at the full resolution.
When are Pyramids Useful?

- We might want to **change resolution** of an image before processing.
- We might **not know which scale** we want, e.g., when searching for a face in an image.
  - In this case, we will generate a full pyramid of different image sizes.
- Can also be used to **accelerate the search**, by first finding at the coarser level of the pyramid and then at the full resolution.
Image Pyramid

[Source: R. Szeliski]
Interpolation and Decimation

- To **interpolate** (or upsample) an image to a higher resolution, we need to select an **interpolation kernel** with which to convolve the image

\[
g(i, j) = \sum_{k,l} f(k, l) h(i - rk, j - rl)
\]

with \( r \) the up-sampling rate.

- The linear interpolator (corresponding to the tent kernel) produces interpolating piecewise linear curves.
Interpolation and Decimation

- To **interpolate** (or upsample) an image to a higher resolution, we need to select an **interpolation kernel** with which to convolve the image

  \[ g(i, j) = \sum_{k, l} f(k, l) h(i - rk, j - rl) \]

  with \( r \) the up-sampling rate.

- The linear interpolator (corresponding to the tent kernel) produces interpolating piecewise linear curves.

- More complex kernels, e.g., B-splines.
Interpolation and Decimation

- To **interpolate** (or upsample) an image to a higher resolution, we need to select an **interpolation kernel** with which to convolve the image
  \[ g(i, j) = \sum_{k,l} f(k, l) h(i - rk, j - rl) \]
  
  with \( r \) the up-sampling rate.

- The linear interpolator (corresponding to the tent kernel) produces interpolating piecewise linear curves.

- More complex kernels, e.g., B-splines.

- **Decimation**: reduces resolution
  \[ g(i, j) = \sum_{k,l} f(k, l) h(i - k/r, j - l/r) \]
  
  with \( r \) the down-sampling rate.
To **interpolate** (or upsample) an image to a higher resolution, we need to select an **interpolation kernel** with which to convolve the image

\[ g(i, j) = \sum_{k,l} f(k, l) h(i - rk, j - rl) \]

with \( r \) the up-sampling rate.

- The linear interpolator (corresponding to the tent kernel) produces interpolating piecewise linear curves.
- More complex kernels, e.g., B-splines.
- **Decimation**: reduces resolution

\[ g(i, j) = \sum_{k,l} f(k, l) h(i - k/r, j - l/r) \]

with \( r \) the down-sampling rate.

- Different filters exist as well.
To **interpolate** (or upsample) an image to a higher resolution, we need to select an **interpolation kernel** with which to convolve the image

\[ g(i, j) = \sum_{k,l} f(k, l) h(i - rk, j - rl) \]

with \( r \) the up-sampling rate.

The linear interpolator (corresponding to the tent kernel) produces interpolating piecewise linear curves.

More complex kernels, e.g., B-splines.

**Decimation**: reduces resolution

\[ g(i, j) = \sum_{k,l} f(k, l) h(i - k/r, j - l/r) \]

with \( r \) the down-sampling rate.

Different filters exist as well.
Multi-Resolution Representations

The most used one is the **Laplacian pyramid**:

- We first **blur** and **subsample** the original image by a factor of two and store this in the next level of the pyramid.
- Subtract then this low-pass version from the original to yield the **band-pass Laplacian image**.
The most used one is the **Laplacian pyramid**:  
- We first **blur** and **subsample** the original image by a factor of two and store this in the next level of the pyramid.  
- Subtract then this low-pass version from the original to yield the **band-pass Laplacian image**.  
- The pyramid has **perfect reconstruction**: the Laplacian images plus the base-level Gaussian are sufficient to exactly reconstruct the original image.
Multi-Resolution Representations

The most used one is the **Laplacian pyramid**:

- We first **blur** and **subsample** the original image by a factor of two and store this in the next level of the pyramid.

- Subtract then this low-pass version from the original to yield the **band-pass Laplacian image**.

- The pyramid has **perfect reconstruction**: the Laplacian images plus the base-level Gaussian are sufficient to exactly reconstruct the original image.

- Wavelets are alternative pyramids. We will not see them here.

[Source: R. Szeliski]
The most used one is the **Laplacian pyramid**:

- We first **blur** and **subsample** the original image by a factor of two and store this in the next level of the pyramid.

- Subtract then this low-pass version from the original to yield the **band-pass Laplacian image**.

- The pyramid has **perfect reconstruction**: the Laplacian images plus the base-level Gaussian are sufficient to exactly reconstruct the original image.

- Wavelets are alternative pyramids. We will not see them here.

---

[Source: R. Szeliski]
How do we reconstruct back?
Laplacian Pyramid Construction

How do we reconstruct back?

- $f_0$
- $l_0$
- $f_1$
- $l_1$
- $f_2$
- $h_0$
- $h_1$
Laplacian Pyramid Re-construction

When is this useful?
Laplacian Pyramid Re-construction

When is this useful?
More Complex Filters
Steerable Filters

- **Oriented filters** are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.

- One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.
**Oriented filters** are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.

One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.

More **efficient** is to apply a few filters corresponding to a few angles and **interpolate** between the responses.
Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.

One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.

More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.

One then needs to know how many filters are required and how to properly interpolate between the responses.
Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.

One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.

More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.

One then needs to know how many filters are required and how to properly interpolate between the responses.

With the correct filter set and the correct interpolation rule, it is possible to determine the response of a filter of arbitrary orientation without explicitly applying that filter.
Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.

One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.

More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.

One then needs to know how many filters are required and how to properly interpolate between the responses.

With the correct filter set and the correct interpolation rule, it is possible to determine the response of a filter of arbitrary orientation without explicitly applying that filter.

Steerable filters are a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of basis filters.
Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.

One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.

More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.

One then needs to know how many filters are required and how to properly interpolate between the responses.

With the correct filter set and the correct interpolation rule, it is possible to determine the response of a filter of arbitrary orientation without explicitly applying that filter.

Steerable filters are a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of basis filters.
Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1
  $$G(x, y, \sigma) = \exp(-x^2 + y^2)$$

- The directional derivative operator is steerable.
Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1
  \[
  G(x, y, \sigma) = \exp(-x^2 + y^2)
  \]

- The directional derivative operator is steerable.

  - The first derivative
  \[
  G^0_1 = \frac{\partial}{\partial x} \exp(-x^2 + y^2) = -2x \exp(-x^2 + y^2)
  \]

  and the same function rotated 90 degrees is
  \[
  G^{90}_1 = \frac{\partial}{\partial y} \exp(-x^2 + y^2) = -2y \exp(-x^2 + y^2)
  \]
Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1
  
  \[ G(x, y, \sigma) = \exp(-x^2 + y^2) \]

- The directional derivative operator is steerable.

- The first derivative
  
  \[ G^0_1 = \frac{\partial}{\partial x} \exp(-x^2 + y^2) = -2x \exp(-x^2 + y^2) \]

  and the same function rotated 90 degrees is
  
  \[ G^{90}_1 = \frac{\partial}{\partial y} \exp(-x^2 + y^2) = -2y \exp(-x^2 + y^2) \]

- A filter of arbitrary orientation $\theta$ can be synthesized by taking a linear combination of $G^0_1$ and $G^{90}_1$
  
  \[ G^\theta_1 = \cos \theta G^0_1 + \sin \theta G^{90}_1 \]

  $G^0_1$ and $G^{90}_1$ are the basis filters and $\cos \theta$ and $\sin \theta$ are the interpolation functions
Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1
  \[ G(x, y, \sigma) = \exp(-x^2 + y^2) \]

- The directional derivative operator is steerable.

- The first derivative
  \[ G^0_1 = \frac{\partial}{\partial x} \exp(-x^2 + y^2) = -2x \exp(-x^2 + y^2) \]
  and the same function rotated 90 degrees is
  \[ G^{90}_1 = \frac{\partial}{\partial y} \exp(-x^2 + y^2) = -2y \exp(-x^2 + y^2) \]

- A filter of arbitrary orientation $\theta$ can be synthesized by taking a linear combination of $G^0_1$ and $G^{90}_1$
  \[ G^\theta_1 = \cos \theta G^0_1 + \sin \theta G^{90}_1 \]
  $G^0_1$ and $G^{90}_1$ are the basis filters and $\cos \theta$ and $\sin \theta$ are the interpolation functions
More on steerable filters

Because convolution is a linear operation, we can synthesize an image filtered at an arbitrary orientation by taking linear combinations of the images filtered with $G_1^0$ and $G_1^{90}$

$$R_1^0 = G_1^0 * I \quad \text{and} \quad R_1^{90} = G_1^{90} * I \quad \text{then} \quad R_1^\theta = \cos \theta R_1^0 + \sin \theta R_1^{90}$$

Check yourself that this is the case.
More on steerable filters

- Because convolution is a linear operation, we can synthesize an image filtered at an arbitrary orientation by taking linear combinations of the images filtered with $G_1^0$ and $G_1^{90}$

\[
\text{if } R_1^0 = G_1^0 \ast I \text{ and } R_1^{90} = G_1^{90} \ast I \text{ then } R_1^\theta = \cos \theta R_1^0 + \sin \theta R_1^{90}
\]

- Check yourself that this is the case.

- See [Freeman & Adelson, 91] for the conditions on when a filter is steerable and how many basis are necessary.
Because convolution is a linear operation, we can synthesize an image filtered at an arbitrary orientation by taking linear combinations of the images filtered with $G^{0}_1$ and $G^{90}_1$

$$R^{0}_1 = G^{0}_1 \star I \quad \text{and} \quad R^{90}_1 = G^{90}_1 \star I$$

then

$$R^{\theta}_1 = \cos \theta R^{0}_1 + \sin \theta R^{90}_1$$

Check yourself that this is the case.

See [Freeman & Adelson, 91] for the conditions on when a filter is steerable and how many basis are necessary.
Figure 2-1: Example of steerable filters. (a) $G_1^{0^\circ}$, first derivative with respect to $x$ (horizontal) of a Gaussian. (b) $G_1^{90^\circ}$, which is $G_1^{0^\circ}$, rotated by $90^\circ$. From a linear combination of these two filters, one can create $G_1^\theta$, an arbitrary rotation of the first derivative of a Gaussian. (c) $G_1^{30^\circ}$, formed by $\frac{1}{2}G_1^{0^\circ} + \frac{\sqrt{3}}{2}G_1^{90^\circ}$. The same linear combinations used to synthesize $G_1^\theta$ from the basis filters will also synthesize the response of an image to $G_1^\theta$ from the responses of the image to the basis filters: (d) Image of circular disk. (e) $G_1^{0^\circ}$ (at a smaller scale than pictured above) convolved with the disk, (d). (f) $G_1^{90^\circ}$ convolved with (d). (g) $G_1^{30^\circ}$ convolved with (d), obtained from $\frac{1}{2}$ [image e] + $\frac{\sqrt{3}}{2}$ [image f].

[Source: W. Freeman 91]
More complex filters

What about the second order derivative?
More complex filters

What about the second order derivative?

- Only three basis are required
More complex filters

What about the second order derivative?

- Only three basis are required

\[ G_{\hat{u}\hat{u}} = u^2 G_{xx} + 2uvG_{x,y} + v^2 G_{y,y} \]

with \( \hat{u} = (u, v) \)
Other transformations
Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.

- It is the running sum of all the pixel values from the origin

\[
s(i, j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k, l)
\]

Summed area tables have been used in face detection [Viola & Jones, 04]
Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin
  \[
  s(i, j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k, l)
  \]
- This can be efficiently computed using a recursive (raster-scan) algorithm
  \[
  s(i, j) = s(i - 1, j) + s(i, j - 1) - s(i - 1, j - 1) + f(i, j)
  \]
If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.

It is the running sum of all the pixel values from the origin

\[ s(i, j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k, l) \]

This can be efficiently computed using a recursive (raster-scan) algorithm

\[ s(i, j) = s(i - 1, j) + s(i, j - 1) - s(i - 1, j - 1) + f(i, j) \]

The image \( s(i, j) \) is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.
Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin:

\[
s(i, j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k, l)
\]

- This can be efficiently computed using a recursive (raster-scan) algorithm:

\[
s(i, j) = s(i - 1, j) + s(i, j - 1) - s(i - 1, j - 1) + f(i, j)
\]

- The image \(s(i, j)\) is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.

- To find the summed area (integral) inside a rectangle \([i_0, i_1] \times [j_0, j_1]\) we simply combine four samples from the summed area table:

\[
S([i_0, i_1] \times [j_0, j_1]) = s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1)
\]
Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin

\[ s(i, j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k, l) \]

- This can be efficiently computed using a recursive (raster-scan) algorithm

\[ s(i, j) = s(i - 1, j) + s(i, j - 1) - s(i - 1, j - 1) + f(i, j) \]

- The image \( s(i, j) \) is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.

- To find the summed area (integral) inside a rectangle \([i_0, i_1] \times [j_0, j_1]\) we simply combine four samples from the summed area table.

\[ S([i_0, i_1] \times [j_0, j_1]) = s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1) \]

- Summed area tables have been used in face detection [Viola & Jones, 04]
Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin
  \[
  s(i, j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k, l)
  \]
- This can be efficiently computed using a recursive (raster-scan) algorithm
  \[
  s(i, j) = s(i - 1, j) + s(i, j - 1) - s(i - 1, j - 1) + f(i, j)
  \]
- The image \(s(i, j)\) is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.
- To find the summed area (integral) inside a rectangle \([i_0, i_1] \times [j_0, j_1]\) we simply combine four samples from the summed area table.
  \[
  S([i_0, i_1] \times [j_0, j_1]) = s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1)
  \]
- Summed area tables have been used in face detection [Viola & Jones, 04]
Example of Integral Images

Figure 3.17  Summed area tables: (a) original image; (b) summed area table; (c) computation of area sum. Each value in the summed area table $s(i, j)$ (red) is computed recursively from its three adjacent (blue) neighbors (3.31). Area sums $S$ (green) are computed by combining the four values at the rectangle corners (purple) (3.32). Positive values are shown in **bold** and negative values in *italics.*
Non-linear filters: Median filter

- We have seen **linear filters**, i.e., their response to a sum of two signals is the same as the sum of the individual responses.

\[ h \circ (f + g) = h \circ f + h \circ g \]

- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.
Non-linear filters: Median filter

- We have seen **linear filters**, i.e., their response to a sum of two signals is the same as the sum of the individual responses.

\[ h \circ (f + g) = h \circ f + h \circ g \]

- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.

- Robust to **outliers**, but not good for Gaussian noise.
We have seen **linear filters**, i.e., their response to a sum of two signals is the same as the sum of the individual responses.

\[ h \circ (f + g) = h \circ f + h \circ g \]

**Median filter**: Non linear filter that selects the median value from each pixels neighborhood.

Robust to **outliers**, but not good for Gaussian noise.
Example of non-linear filters

(Median filter)  (\(\alpha\)-trimmed mean)
We have seen **linear filters**, i.e., their response to a sum of two signals is the same as the sum of the individual responses.

\[ h \circ (f + g) = h \circ f + h \circ g \]

- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.

- Robust to **outliers**, but not good for Gaussian noise.

- **α-trimmed mean**: averages together all of the pixels except for the \( \alpha \) fraction that are the smallest and the largest.
Non-linear filters: Median filter

- We have seen **linear filters**, i.e., their response to a sum of two signals is the same as the sum of the individual responses.

\[ h \circ (f + g) = h \circ f + h \circ g \]

- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.

- Robust to **outliers**, but not good for Gaussian noise.

- **\( \alpha \)-trimmed mean**: averages together all of the pixels except for the \( \alpha \) fraction that are the smallest and the largest.
Example of non-linear filters

(Median filter)  
(\(\alpha\)-trimmed mean)
Bilateral Filtering

- Weighted filter kernel with a **better outlier rejection**.
- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.
Bilateral Filtering

- Weighted filter kernel with a **better outlier rejection**.

- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.

- The output pixel value depends on a weighted combination of neighboring pixel values

\[
g(i,j) = \frac{\sum_{k,l} f(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}
\]
Bilateral Filtering

- Weighted filter kernel with a **better outlier rejection**.
- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.
- The output pixel value depends on a weighted combination of neighboring pixel values
  \[ g(i, j) = \frac{\sum_{k, l} f(k, l) w(i, j, k, l)}{\sum_{k, l} w(i, j, k, l)} \]

- Data-dependent bilateral weight function
  \[ w(i, j, k, l) = \exp\left( -\frac{(i - k)^2 + (j - l)^2}{2\sigma_d^2} - \frac{||f(i, j) - f(k, l)||^2}{2\sigma_r^2} \right) \]
  composed of the **domain kernel** and the **range kernel**.
Bilateral Filtering

- Weighted filter kernel with a **better outlier rejection**.
- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.
- The output pixel value depends on a weighted combination of neighboring pixel values
  \[
  g(i, j) = \frac{\sum_{k,l} f(k, l)w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)}
  \]

- Data-dependent bilateral weight function
  \[
  w(i, j, k, l) = \exp\left( -\frac{(i - k)^2 + (j - l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2} \right)
  \]
  composed of the **domain kernel** and the **range kernel**.
Example Bilateral Filtering

**Figure:** Bilateral filtering [Durand & Dorsey, 02]. (a) noisy step edge input. (b) domain filter (Gaussian). (c) range filter (similarity to center pixel value). (d) bilateral filter. (e) filtered step edge output. (f) 3D distance between pixels

[Source: R. Szeliski]
Distance Transform

- Useful to quickly precomputing the distance to a curve or a set of points.
- Let $d(k, l)$ be some distance metric between pixel offsets, e.g., Manhattan distance
  $$d(k, l) = |k| + |l|$$
optorynchusor Euclidean distance
  $$d(k, l) = \sqrt{k^2 + l^2}$$
Distance Transform

- Useful to quickly precomputing the distance to a curve or a set of points.
- Let $d(k, l)$ be some distance metric between pixel offsets, e.g., Manhattan distance
  \[ d(k, l) = |k| + |l| \]
  or Euclidean distance
  \[ d(k, l) = \sqrt{k^2 + l^2} \]
- The distance transform $D(i, j)$ of a binary image $b(i, j)$ is defined as
  \[ D(i, j) = \min_{k,l; b(k,l)=0} d(i - k, j - l) \]
  it is the distance to the nearest pixel whose value is 0.
Distance Transform

- Useful to quickly precomputing the distance to a curve or a set of points.
- Let $d(k, l)$ be some distance metric between pixel offsets, e.g., Manhattan distance
  
  $$d(k, l) = |k| + |l|$$

  or Euclidean distance
  
  $$d(k, l) = \sqrt{k^2 + l^2}$$

- The distance transform $D(i, j)$ of a binary image $b(i, j)$ is defined as
  
  $$D(i, j) = \min_{k,l; b(k,l)=0} d(i - k, j - l)$$

  it is the distance to the nearest pixel whose value is 0.
Distance Transform Algorithm

- The **Manhattan distance** can be computed using a forward and backward pass of a simple raster-scan algorithm.

  - **Forward pass**: each non-zero pixel in \( b \) is replaced by the minimum of \( 1 + \) the distance of its north or west neighbor.

  - **Backward pass**: the same, but the minimum is both over the current value \( D \) and \( 1 + \) the distance of the south and east neighbors.
The **Manhattan distance** can be computed using a forward and backward pass of a simple raster-scan algorithm.

**Forward pass**: each non-zero pixel in b is replaced by the minimum of 1 + the distance of its north or west neighbor.

**Backward pass**: the same, but the minimum is both over the current value D and 1 + the distance of the south and east neighbors.

---

**Figure**: City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange value; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.

[Source: R. Szeliski]
The **Manhattan distance** can be computed using a forward and backward pass of a simple raster-scan algorithm.

**Forward pass**: each non-zero pixel in $b$ is replaced by the minimum of $1 +$ the distance of its north or west neighbor.

**Backward pass**: the same, but the minimum is both over the current value $D$ and $1 +$ the distance of the south and east neighbors.

![City block distance transform](source)

**Figure**: City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange value; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.

[Source: R. Szeliski]
Example of Distance Transform

- More complicated in the Euclidean case.
- Example of a distance transform

The ridges is the skeleton or medial axis.

Extension: Signed distance transform.

[Source: P. Felzenszwalb]
Fourier Transform

- Fourier analysis could be used to analyze the frequency characteristics of various filters.
- How can we analyze what a given filter does to high, medium, and low frequencies?

```latex
\text{s}(x) = \sin(2\pi f x + \phi_i) = \sin(\omega x + \phi_i)
```

If we convolve the sinusoidal signal \( s(x) \) with a filter whose impulse response is \( h(x) \), we get another sinusoid of the same frequency but different magnitude and phase \( o(x) = h(x) * s(x) = A \sin(\omega x + \phi_o) \).
Fourier Transform

- Fourier analysis could be used to analyze the frequency characteristics of various filters.

- How can we analyze what a given filter does to high, medium, and low frequencies?

- Pass a sinusoid of known frequency through the filter and observe by how much it is attenuated.

\[ s(x) = \sin(2\pi f x + \phi_i) = \sin(\omega x + \phi_i) \]

with frequency \( f \), angular frequency \( \omega \) and phase \( \phi_i \).
Fourier analysis could be used to analyze the frequency characteristics of various filters.

How can we analyze what a given filter does to high, medium, and low frequencies?

Pass a sinusoid of known frequency through the filter and to observe by how much it is attenuated

\[ s(x) = \sin(2\pi fx + \phi_i) = \sin(\omega x + \phi_i) \]

with frequency \( f \), angular frequency \( \omega \) and phase \( \phi_i \).

If we convolve the sinusoidal signal \( s(x) \) with a filter whose impulse response is \( h(x) \), we get another sinusoid of the same frequency but different magnitude and phase

\[ o(x) = h(x) \ast s(x) = A \sin(\omega x + \phi_o) \]
Fourier Transform

- Fourier analysis could be used to analyze the frequency characteristics of various filters.
- How can we analyze what a given filter does to high, medium, and low frequencies?
- Pass a sinusoid of known frequency through the filter and to observe by how much it is attenuated

\[ s(x) = \sin(2\pi fx + \phi_i) = \sin(\omega x + \phi_i) \]

with frequency \( f \), angular frequency \( \omega \) and phase \( \phi_i \).

- If we convolve the sinusoidal signal \( s(x) \) with a filter whose impulse response is \( h(x) \), we get another sinusoid of the same frequency but different magnitude and phase

\[ o(x) = h(x) \ast s(x) = A \sin(\omega x + \phi_o) \]
Convolution can be expressed as a weighted summation of shifted input signals (sinusoids); so it is just a single sinusoid at that frequency.

\[ o(x) = h(x) * s(x) = A \sin(\omega x + \phi_o) \]

\( A \) is the **gain** or **magnitude** of the filter, while the phase difference \( \Delta \phi = \phi_o - \phi_i \) is the **shift** or **phase**

**Figure 3.24** The Fourier Transform as the response of a filter \( h(x) \) to an input sinusoid \( s(x) = e^{j \omega x} \) yielding an output sinusoid \( o(x) = h(x) * s(x) = A e^{j \omega x + \phi} \).
The sinusoid is expressed as \( s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x \) and the filter sinusoid as

\[
o(x) = h(x) \ast s(x) = Ae^{j\omega x + \phi}
\]

The Fourier transform pair is

\[
h(x) \longleftrightarrow H(\omega)
\]
Complex notation

- The sinusoid is expressed as $s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x$ and the filter sinusoid as
  \[ o(x) = h(x) \ast s(x) = Ae^{j\omega x + \phi} \]

- The Fourier transform pair is
  \[ h(x) \longleftrightarrow H(\omega) \]

- The Fourier transform in continuous domain
  \[ H(\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x} \, dx \]
Complex notation

- The sinusoid is express as \( s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x \) and the filter sinusoid as

\[
o(x) = h(x) \ast s(x) = A e^{j\omega x + \phi}
\]

- The Fourier transform pair is

\[
h(x) \leftrightarrow H(\omega)
\]

- The Fourier transform in continuous domain

\[
H(\omega) = \int_{-\infty}^{\infty} h(x) e^{-j\omega x} dx
\]

- The Fourier transform in discrete domain

\[
H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x) e^{-j\frac{2\pi kx}{N}}
\]

where \( N \) is the length of the signal.
The sinusoid is expressed as
\[ s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x \]
and the filter sinusoid as
\[ o(x) = h(x) \ast s(x) = Ae^{j\omega x + \phi} \]

The Fourier transform pair is
\[ h(x) \longleftrightarrow H(\omega) \]

The Fourier transform in continuous domain
\[ H(\omega) = \int_{-\infty}^{\infty} h(x) e^{-j\omega x} dx \]

The Fourier transform in discrete domain
\[ H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x) e^{-j \frac{2\pi k x}{N}} \]

where \( N \) is the length of the signal.

The discrete form is known as the Discrete Fourier Transform (DFT).
The sinusoid is express as \( s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x \) and the filter sinusoid as
\[
o(x) = h(x) \ast s(x) = Ae^{j\omega x + \phi}
\]
The Fourier transform pair is
\[
h(x) \longleftrightarrow H(\omega)
\]
The Fourier transform in continuous domain
\[
H(\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x} \, dx
\]
The Fourier transform in discrete domain
\[
H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x)e^{-j\frac{2\pi k x}{N}}
\]
where \( N \) is the length of the signal.
The discrete form is known as the Discrete Fourier Transform (DFT).
## Properties Fourier Transform

<table>
<thead>
<tr>
<th>Property</th>
<th>Signal</th>
<th>Transform</th>
</tr>
</thead>
<tbody>
<tr>
<td>superposition</td>
<td>$f_1(x) + f_2(x)$</td>
<td>$F_1(\omega) + F_2(\omega)$</td>
</tr>
<tr>
<td>shift</td>
<td>$f(x - x_0)$</td>
<td>$F(\omega)e^{-j\omega x_0}$</td>
</tr>
<tr>
<td>reversal</td>
<td>$f(-x)$</td>
<td>$F^*(\omega)$</td>
</tr>
<tr>
<td>convolution</td>
<td>$f(x) * h(x)$</td>
<td>$F(\omega)H(\omega)$</td>
</tr>
<tr>
<td>correlation</td>
<td>$f(x) \otimes h(x)$</td>
<td>$F(\omega)H^*(\omega)$</td>
</tr>
<tr>
<td>multiplication</td>
<td>$f(x)h(x)$</td>
<td>$F(\omega) * H(\omega)$</td>
</tr>
<tr>
<td>differentiation</td>
<td>$f'(x)$</td>
<td>$j\omega F(\omega)$</td>
</tr>
<tr>
<td>domain scaling</td>
<td>$f(ax)$</td>
<td>$1/a F(\omega/a)$</td>
</tr>
<tr>
<td>real images</td>
<td>$f(x) = f^*(x)$</td>
<td>$F(\omega) = F(-\omega)$</td>
</tr>
<tr>
<td>Parseval’s Theorem</td>
<td>$\sum_x</td>
<td>f(x)</td>
</tr>
</tbody>
</table>

[Source: R. Szeliski]
<table>
<thead>
<tr>
<th>Name</th>
<th>Signal</th>
<th>Transform</th>
</tr>
</thead>
<tbody>
<tr>
<td>impulse</td>
<td>$\delta(x)$</td>
<td>$1$</td>
</tr>
<tr>
<td>shifted impulse</td>
<td>$\delta(x - u)$</td>
<td>$e^{-j\omega u}$</td>
</tr>
<tr>
<td>box filter</td>
<td>box($x/a$)</td>
<td>$a\text{sinc}(a\omega)$</td>
</tr>
<tr>
<td>tent</td>
<td>tent($x/a$)</td>
<td>$a\text{sinc}^2(a\omega)$</td>
</tr>
<tr>
<td>Gaussian</td>
<td>$G(x; \sigma)$</td>
<td>$\frac{\sqrt{2\pi}}{\sigma} G(\omega; \sigma^{-1})$</td>
</tr>
<tr>
<td>Laplacian of Gaussian</td>
<td>$(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})G(x; \sigma)$</td>
<td>$-\frac{\sqrt{2\pi}}{\sigma} \omega^2 G(\omega; \sigma^{-1})$</td>
</tr>
<tr>
<td>Gabor</td>
<td>$\cos(\omega_0 x)G(x; \sigma)$</td>
<td>$\frac{\sqrt{2\pi}}{\sigma} G(\omega \pm \omega_0; \sigma^{-1})$</td>
</tr>
<tr>
<td>unsharp mask</td>
<td>$(1 + \gamma)\delta(x) - \gamma G(x; \sigma)$</td>
<td>$\frac{(1 + \gamma)}{\sqrt{2\pi\sigma}} G(\omega; \sigma^{-1})$</td>
</tr>
<tr>
<td>windowed sinc</td>
<td>$r\cos(x/(aW))$</td>
<td>(see Figure 3.29)</td>
</tr>
<tr>
<td>Name</td>
<td>Kernel</td>
<td>Transform</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| box-3  | \(\frac{1}{3}\) \[
\begin{array}{ccc}
1 & 1 & 1 \\
\end{array}\]
| \(\frac{1}{3}(1 + 2\cos \omega)\) | ![Plot](image1.png) |
| box-5  | \(\frac{1}{5}\) \[
\begin{array}{ccccc}
1 & 1 & 1 & 1 & 1 \\
\end{array}\]
| \(\frac{1}{5}(1 + 2\cos \omega + 2\cos 2\omega)\) | ![Plot](image2.png) |
| linear | \(\frac{1}{4}\) \[
\begin{array}{ccc}
1 & 2 & 1 \\
\end{array}\]
| \(\frac{1}{2}(1 + \cos \omega)\) | ![Plot](image3.png) |
| binomial | \(\frac{1}{16}\) \[
\begin{array}{cccc}
1 & 4 & 6 & 4 & 1 \\
\end{array}\]
| \(\frac{1}{4}(1 + \cos \omega)^2\) | ![Plot](image4.png) |
| Sobel  | \(\frac{1}{2}\) \[
\begin{array}{ccc}
-1 & 0 & 1 \\
\end{array}\]
| \(\sin \omega\) | ![Plot](image5.png) |
| corner | \(\frac{1}{2}\) \[
\begin{array}{ccc}
-1 & 2 & -1 \\
\end{array}\]
| \(\frac{1}{2}(1 - \cos \omega)\) | ![Plot](image6.png) |

[Source: R. Szeliski]
Same as 1D, but in 2D. Now the sinusoid is

\[ s(x, y) = \sin(\omega_x x + \omega_y y) \]

The 2D Fourier in continuous domain is then

\[ H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} \, dx \, dy \]

and in the discrete domain

\[ H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-2\pi j \frac{k_x x + k_y y}{MN}} \]

where \( M \) and \( N \) are the width and height of the image.
2D Fourier Transform

- Same as 1D, but in 2D. Now the sinusoid is
  \[ s(x, y) = \sin(\omega_x x + \omega_y y) \]

- The 2D Fourier in continuous domain is then
  \[ H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} \, dx \, dy \]
  
  and in the discrete domain
  \[ H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-2\pi j \frac{k_x x + k_y y}{MN}} \]
  
  where M and N are the width and height of the image.

- All the properties carry over to 2D.
2D Fourier Transform

- Same as 1D, but in 2D. Now the sinusoid is
  \[ s(x, y) = \sin(\omega_x x + \omega_y y) \]

- The 2D Fourier in continuous domain is then
  \[ H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} \, dx \, dy \]

  and in the discrete domain

  \[ H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-2\pi j \frac{k_x x + k_y y}{MN}} \]

  where M and N are the width and height of the image.

- All the properties carry over to 2D.
Example of 2D Fourier Transform

[Source: A. Jepson]
Next class ... image features