Midterm Review

(unless otherwise stated slides are taken or adopted from Bob Woodham, Jim Little and Fred Tung)
Midterm Details

Closed book, (simple) calculators allowed

Format similar to posted practice problems
  — Part A: Multiple-part true/false
  — Part B: Short answer

No coding questions

No complex math questions
Midterm Review: Study materials

**Lectures** 1–12 slides

Assigned **readings** from Szeliski / Forsyth and Ponce

**Assignments** 1–2

**iClicker** questions

**Practice quizzes** on Canvas

**Practice problems / solutions** on Canvas
The image formation process that produces a particular image depends on

- Lightening condition
- Scene geometry
- Surface properties
- Camera optics

Sensor (or eye) captures amount of light reflected from the object
Diffuse vs Specular Surfaces
Surface reflection depends on both the viewing \((\theta_v, \phi_v)\) and illumination \((\theta_i, \phi_i)\) direction, with Bidirectional Reflection Distribution Function: \(\text{BRDF}(\theta_i, \phi_i, \theta_v, \phi_v)\)

**Lambertian** surface:

\[
\text{BRDF}(\theta_i, \phi_i, \theta_v, \phi_v) = \frac{\rho d}{\pi}
\]

\[
L = \frac{\rho d}{\pi} I(\hat{\mathbf{i}} \cdot \hat{\mathbf{n}})
\]

**Mirror** surface: all incident light reflected in one directions \((\theta_v, \phi_v) = (\theta_r, \phi_r)\)

*Slide adopted from: Ioannis (Yannis) Gkioulekas (CMU)*
Camera Obscura (Latin for “dark chamber”)

Reinerus Gemma-Frisius observed an eclipse of the sun at Louvain on January 24, 1544. He used this illustration in his book, “De Radio Astronomica et Geometrica,” 1545. It is thought to be the first published illustration of a camera obscura.

Credit: John H., Hammond, “The Camera Obscure, A Chronicle”
Camera Obscura (latin for “dark chamber”)

Reinerus Gemma-Frisius observed an eclipse of the sun at Louvain on January 24, 1544. He used this illustration in his book, “De Radio Astronomica et Geometrica,” 1545. It is thought to be the first published illustration of a camera obscura.

principles behind the pinhole camera or camera obscura were first mentioned by Chinese philosopher Mozi (Mo-Ti) (470 to 390 BCE)

Credit: John H., Hammond, “Th Camera Obscure, A Chronicle”
Pinhole Camera (Simplified)

\( f' \) is the **focal length** of the camera

**Image plane**  **pinhole**  **object**

**Note:** In a pinhole camera we can adjust the focal length, all this will do is change the **size** of the resulting image
Pinhole Camera (Simplified)

It is convenient to think of the image plane which is in front of the pinhole.

What happens if object moves towards the camera? Away from the camera?
Perspective Projection

3D object point

Forsyth & Ponce (1st ed.) Figure 1.4

\[ P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \] projects to 2D image point \[ P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \]

where

\[ x' = f' \frac{x}{z} \]
\[ y' = f' \frac{y}{z} \]

Note: this assumes world coordinate frame at the optical center (pinhole) and aligned with the image plane, image coordinate frame aligned with the camera coordinate frame
Summary of **Projection Equations**

3D object point \( P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \) projects to 2D image point \( P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \) where

<table>
<thead>
<tr>
<th>Type</th>
<th>Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Perspective</strong></td>
<td>( x' = f' \frac{x}{z} )  ( y' = f' \frac{y}{z} )</td>
</tr>
<tr>
<td><strong>Weak Perspective</strong></td>
<td>( x' = m x )  ( y' = m y )  ( m = \frac{f'}{z_0} )</td>
</tr>
<tr>
<td><strong>Orthographic</strong></td>
<td>( x' = x )  ( y' = y )</td>
</tr>
</tbody>
</table>
Sample Question: Image Formation

**True** of **false**: A pinhole camera uses an orthographic projection.
Why **Not** a Pinhole Camera?

— If pinhole is **too big** then many directions are averaged, blurring the image

— If pinhole is **too small** then diffraction becomes a factor, also blurring the image

— Generally, pinhole cameras are **dark**, because only a very small set of rays from a particular scene point hits the image plane

— Pinhole cameras are **slow**, because only a very small amount of light from a particular scene point hits the image plane per unit time

Pinhole Model (Simplified) with Lens
Vignetting

Image Credit: Cambridge in Colour
Chromatic Aberration

— Index of **refraction depends on wavelength**, $\lambda$, of light
— Light of different colours follows different paths
— Therefore, not all colours can be in equal focus

*Image Credit: Trevor Darrell*
Lines in the world are no longer lines on the image, they are curves!
Sample Question: Cameras and Lenses

True or false: Snell’s Law describes how much light is reflected and how much passes through the boundary between two materials.
For a given $X$ and $Y$, superimpose the filter on the image centered at $(X, Y)$

Compute the new pixel value, $I'(X, Y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(X, Y)$ and the corresponding values in the filter.

$$I'(X, Y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(I, J) I(X + i, Y + j)$$
Linear Filter Example

\[ F(X, Y) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \]

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j) \]

Slide Credit: Ioannis (Yannis) Gkioulakas (CMU)
Linear Filter Example

\[ I(X, Y) \]

\[ F(X, Y) \]

\[ \frac{1}{9} \]

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j) \]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j) \]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(I, J) I(X + i, Y + j) \]

**Slide Credit:** Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[
F(X, Y) = \frac{1}{9}
\]

\[
I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j)
\]

output image

Slide Credit: Ioannis (Yannis) Gkiouldekas (CMU)
### Linear Filter Example

The linear filter example illustrates how an input image is processed to produce an output image. The filter operation is defined as

$$I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j)$$

where $F(X, Y)$ is the filter kernel, $I(X, Y)$ is the input image, and $I'(X, Y)$ is the output image. The filter kernel used in the example is a 3x3 average filter, shown as

$$F(X, Y) = \frac{1}{9}$$

The input image is

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 90 90 90 90 90 0 0
0 0 90 0 90 90 90 0 0
0 0 0 90 90 90 90 0 0
0 0 0 90 90 90 90 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

The output image is

```
0 10 20
```

**Slide Credit:** Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

$$I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j)$$

Filter: $F(X, Y)$

Image: $I(X, Y)$

Output: $I'(X, Y)$

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ F(X, Y) \]

\[ \frac{1}{9} \]

\[ I(X, Y) \]

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j) \]

output

filter

image (signal)

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I(X,Y) \]

\[ F(X,Y) \]

\[ \frac{1}{9} \]

\[ I'(X,Y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(I,J) I(X+i,Y+j) \]

**Slide Credit:** Ioannis (Yannis) Gkioulékas (CMU)
## Linear Filter Example

A linear filter `F(X, Y)` is applied to an image `I(X, Y)` to produce an output image `I'(X, Y)`. The filter is defined as:

\[
F(X, Y) = \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

The output image `I'(X, Y)` is calculated as:

\[
I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j)
\]

where `F(I, J)` is the filter and `I(X + i, Y + j)` is the image (signal) at each pixel location.

---

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j) \]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(I, J) I(X + i, Y + j) \]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j) \]

Slide Credit: Ioannis (Yannis) Gkioullekas (CMU)
Linear Filter Example

\[ I(X, Y) \]

\[ F(X, Y) \]

\[ \frac{1}{9} \]

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j) \]
Linear Filter Example

The equation for the filtered image output is given by:

$$I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j)$$

where $F(X, Y)$ is the filter, $I(X, Y)$ is the image, and $I'(X, Y)$ is the output image.
Linear Filter Example

$$F(X, Y)$$

filter

$$I(X, Y)$$

image

$$I'(X, Y)$$

output

$$I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j)$$

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j) \]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j) \]

Output \( I'(X, Y) \)

Filter \( F(X, Y) \)

Image \( I(X, Y) \)

Filter image (signal)

Image output

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[ I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X + i, Y + j) \]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Linear Filter Example

\[
F(X, Y) = \frac{1}{9}
\]

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
I(X, Y)
\]

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j)
\]

\[
\begin{array}{cccccc}
0 & 10 & 20 & 30 & 30 & 30 & 20 & 10 \\
0 & 20 & 40 & 60 & 60 & 60 & 40 & 20 \\
0 & 30 & 50 & 80 & 80 & 90 & 60 & 30 \\
0 & 30 & 50 & 80 & 80 & 90 & 60 & 30 \\
0 & 20 & 30 & 50 & 50 & 60 & 40 & 20 \\
0 & 10 & 20 & 30 & 30 & 30 & 20 & 10 \\
10 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
10 & 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
\end{array}
\]

output

Slide Credit: Ioannis (Yannis) Gkioulkas (CMU)
Linear Filters: **Boundary Effects**

Three standard ways to deal with boundaries:

1. **Ignore these locations**: Make the computation undefined for the top and bottom $k$ rows and the leftmost and rightmost $k$ columns

2. **Pad the image with zeros**: Return zero whenever a value of $I$ is required at some position outside the defined limits of $X$ and $Y$

3. **Assume periodicity**: The top row wraps around to the bottom row; the leftmost column wraps around to the rightmost column
— The **correlation** of $F(X, Y)$ and $I(X, Y)$ is:

$$I'(X, Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(I, J) I(X+i, Y+j)$$

**Visual interpretation**: Superimpose the filter $F$ on the image $I$ at $(X, Y)$, perform an element-wise multiply, and sum up the values.

— **Convolution** is like **correlation** except filter “flipped”

$$\text{if } F(X, Y) = F(-X, -Y) \text{ then correlation } = \text{ convolution.}$$
Linear System: Characterization Theorem

Any linear, shift invariant operation can be expressed as a convolution
Linear System: **Characterization** Theorem

**Any** linear, shift invariant operation can be expressed as a convolution

(if and only if’ result)
Low-pass Filtering = “Smoothing”

**Gaussian Filter**

\[
\begin{array}{ccccc}
1 & 4 & 6 & 4 & 1 \\
4 & 16 & 24 & 16 & 4 \\
6 & 24 & 36 & 24 & 6 \\
4 & 16 & 24 & 16 & 4 \\
1 & 4 & 6 & 4 & 1 \\
\end{array}
\]

\[
\frac{1}{256}
\]

All of these filters are **Low-pass Filters**

**Low-pass filter:** Low pass filter filters out all of the high frequency content of the image, only low frequencies remain
Other smoothing filters

Box Filter

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\frac{1}{9}
\]

Pillbox Filter
Smoothing with a Gaussian

Idea: Weight contributions of pixels by spatial proximity (nearness)

2D **Gaussian** (continuous case):

\[ G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \]

Forsyth & Ponce (2nd ed.)
Figure 4.2
Gaussian: Area Under the Curve

- 68%
- 95%
- 99.7%
- 99.99%
Efficient Implementation: Separability

A 2D function of $x$ and $y$ is **separable** if it can be written as the product of two functions, one a function only of $x$ and the other a function only of $y$

Both the 2D box filter and the 2D Gaussian filter are separable.

Both can be implemented as two 1D convolutions:

- First, convolve each row with a 1D filter
- Then, convolve each column with a 1D filter
- Aside: or vice versa

The **2D Gaussian** is the only (non trivial) 2D function that is both separable and rotationally invariant.
Linear Filters: Additional Properties

Let $\otimes$ denote convolution. Let $I(X,Y)$ be a digital image. Let $F$ and $G$ be digital filters

— Convolution is **associative**. That is,

$$G \otimes (F \otimes I(X,Y)) = (G \otimes F) \otimes I(X,Y)$$

— Convolution is **symmetric**. That is,

$$(G \otimes F) \otimes I(X,Y) = (G \otimes F) \otimes I(X,Y)$$

Convolving $I(X,Y)$ with filter $F$ and then convolving the result with filter $G$ can be achieved in single step, namely convolving $I(X,Y)$ with filter $G \otimes F = F \otimes G$
Bilateral Filter

An edge-preserving non-linear filter

**Like** a Gaussian filter:

— The filter weights depend on spatial distance from the center pixel
— Pixels nearby (in space) should have greater influence than pixels far away

**Unlike** a Gaussian filter:

— The filter weights also depend on range distance from the center pixel
— Pixels with similar brightness value should have greater influence than pixels with dissimilar brightness value
**Bilateral Filter**

**Gaussian** filter: weights of neighbor at a spatial offset \((x, y)\) away from the center pixel \(I(X, Y)\) given by:

\[
G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)
\]

(with appropriate normalization)

**Bilateral** filter: weights of neighbor at a spatial offset \((x, y)\) away from the center pixel \(I(X, Y)\) given by a product:

\[
\exp\left(-\frac{x^2+y^2}{2\sigma_d^2}\right) \exp\left(-\frac{(I(X+x, Y+y)-I(X,Y))^2}{2\sigma_r^2}\right)
\]

(with appropriate normalization)
Bilateral Filter Application: Denoising

Noisy Image  Gaussian Filter  Bilateral Filter

Slide Credit: Alexander Wong
Sample Question: Filters

What does the following $3 \times 3$ linear, shift invariant filter compute when applied to an image?

\[
\begin{bmatrix}
-1 & -1 & -1 \\
0 & 0 & 0 \\
1 & 1 & 1
\end{bmatrix}
\]
Resampling Images

- Naive method: form new image by selecting every $n$th pixel
Aliasing Example

- Sampling every 5th pixel, while shifting rightwards 1 pixel at a time
Aliasing Example

• Sampling every 5th pixel, while shifting rightwards 1 pixel at a time
Example: A Simple Sine Wave

How do we discretize the signal?

Signal can be confused with one at lower frequency
— This is called “Aliasing”
Nyquist Sampling Theorem

To avoid aliasing a signal must be sampled at twice the maximum frequency:

\[ f_s > 2 \times f_{max} \]

where \( f_s \) is the sampling frequency, and \( f_{max} \) is the maximum frequency present in the signal

Furthermore, Nyquist’s theorem states that a signal is exactly recoverable from its samples if sampled at the Nyquist rate (or higher)

Note: that a signal must be bandlimited for this to apply (i.e., it has a maximum frequency)
Template Matching

A toy example

Slide Credit: Kristen Grauman
Template Matching

Detected template

Correlation map

Slide Credit: Kristen Grauman
Template Matching

Similarity measures between a filter $J$ and local image region $I$

Correlation, $\text{CORR} = I \cdot J = I^T J$

Normalised Correlation, $\text{NCORR} = \frac{I^T J}{||I||\cdot||J||} = \cos \theta$

Sum Squared Difference, $\text{SSD} = |I - J|^2$

Normalized correlation varies between $-1$ and $1$, attains the value $1$ when the filter and image region are identical (up to a scale factor)

Minimising SSD and maximizing Normalized Correlation are equivalent if $||I|| = ||J|| = 1$
Template Matching

Convolve image with template, find local maxima
Template Matching

Convolve image with template, find local maxima
Template Matching

Convolve image with template, find local maxima
Template Matching

Convolve image with template, find local maxima

* → Non-max suppress
**Template Matching**

Convolve image with template, find local maxima
Template Matching

Convolve image with template, find local maxima

* Non-max suppress + threshold
When might template matching fail?

- Different scales
- Different orientation
- Lighting conditions
- Left vs. Right hand
- Partial Occlusions
- Different Perspective
- Motion / blur
Example 1:

Template (left), image (middle), normalized correlation (right)

Note peak value at the true position of the hand

Sample Question: Template Matching

**True or false:** Normalized correlation is robust to a constant scaling in the image brightness.
Scaled Representations: Goals

to find **template matches** at all scales
  — template size constant, image scale varies
  — finding hands or faces when we don’t know what size they are in the image

efficient search for image–to–image correspondences
  — look first at coarse scales, refine at finer scales
  — much less cost (but may miss best match)

to examine all **levels of detail**
  — find edges with different amounts of blur
  — find textures with different spatial frequencies (i.e., different levels of detail)
Blur with a Gaussian kernel, then select every 2nd pixel

\[ I_s(x, y) = I(x, y) \ast g_\sigma(x, y) \]
Blur with a Gaussian kernel, then select every 2nd pixel

\[ I_s(x, y) = I(x, y) \ast g_\sigma(x, y) \]
Blur with a Gaussian kernel, then select every 2nd pixel

\[ I_s(x, y) = I(x, y) * g_\sigma(x, y) \]
Blur with a Gaussian kernel, then select every 2nd pixel

\[ I_s(x, y) = I(x, y) * g_\sigma(x, y) \]
Blur with a Gaussian kernel, then select every 2nd pixel

\[ I_s(x, y) = I(x, y) * g_\sigma(x, y) \]
Blur with a Gaussian kernel, then select every 2nd pixel

\[ I_s(x, y) = I(x, y) \ast g_\sigma(x, y) \]
Blur with a Gaussian kernel, then select every 2nd pixel

\[ I_s(x, y) = I(x, y) \ast g_\sigma(x, y) \]
Gaussian Pyramid | Laplacian Pyramid

G1

G2

G3

G4
Gaussian Pyramid  \rightarrow \text{Laplacian Pyramid}

$G1$

$G2$

$G3$

$G4 \rightarrow L4$
Gaussian Pyramid

Laplacian Pyramid

G1

G2

G3

G4

L4

↑2
Gaussian Pyramid

Laplacian Pyramid

G1

G2

G3

G4

L3

L4
Gaussian Pyramid

Laplacian Pyramid

G1

G2

G3

G4

L3

L4
Gaussian Pyramid

Laplacian Pyramid

\[ G_1 \rightarrow \uparrow 2 \rightarrow G_2 \rightarrow \uparrow 2 \rightarrow G_3 \rightarrow \uparrow 2 \rightarrow G_4 \rightarrow \square \rightarrow L_1 \]

\[ G_1 \rightarrow \oplus \rightarrow G_2 \rightarrow \oplus \rightarrow G_3 \rightarrow \oplus \rightarrow G_4 \rightarrow \square \rightarrow L_2 \]

\[ G_1 \rightarrow \oplus \rightarrow G_2 \rightarrow \oplus \rightarrow G_3 \rightarrow \oplus \rightarrow G_4 \rightarrow \square \rightarrow L_3 \]

\[ G_1 \rightarrow \oplus \rightarrow G_2 \rightarrow \oplus \rightarrow G_3 \rightarrow \oplus \rightarrow G_4 \rightarrow \square \rightarrow L_4 \]
Pyramid Blending
**Step 2:** blend lower frequency bands over larger spatial ranges, high frequency bands over small spatial ranges.
From Template Matching to **Local Feature Detection**

Find the chair in this image

Pretty much garbage
Simple template matching is not going to make it

**Slide Credit:** Li Fei-Fei, Rob Fergus, and Antonio Torralba
Estimating Derivatives

Recall, for a 2D (continuous) function, $f(x,y)$

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x + \varepsilon, y) - f(x, y)}{\varepsilon}$$

Differentiation is linear and shift invariant, and therefore can be implemented as a convolution

A (discrete) approximation is

$$\frac{\partial f}{\partial x} \approx \frac{F(X + 1, y) - F(x, y)}{\Delta x}$$
Estimating **Derivatives**

A similar definition (and approximation) holds for \( \frac{\partial f}{\partial y} \)

Image **noise** tends to result in pixels not looking exactly like their neighbours, so simple “finite differences” are sensitive to noise.

The usual way to deal with this problem is to **smooth** the image prior to derivative estimation.
What Causes **Edges**?

- Depth discontinuity
- Surface orientation discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)
- Illumination discontinuity (e.g., shadow)

*Slide Credit: Christopher Rasmussen*
**Smoothing and Differentiation**

**Edge:** a location with high gradient (derivative)

Need smoothing to reduce noise prior to taking derivative

Need two derivatives, in x and y direction

We can use **derivative of Gaussian** filters

— because differentiation is convolution, and
— convolution is associative

Let \( \otimes \) denote convolution

\[
D \otimes (G \otimes I(X, Y)) = (D \otimes G) \otimes I(X, Y)
\]
Let $I(X, Y)$ be a (digital) image

Let $I_x(X, Y)$ and $I_y(X, Y)$ be estimates of the partial derivatives in the $x$ and $y$ directions, respectively.

Call these estimates $I_x$ and $I_y$ (for short) The vector $[I_x, I_y]$ is the \textbf{gradient}

The scalar $\sqrt{I_x^2 + I_y^2}$ is the \textbf{gradient magnitude}

The \textbf{gradient direction} is given by: $\theta = \tan^{-1} \left( \frac{I_y}{I_x} \right)$
Two generic approaches to edge point detection:

- (significant) local extrema of a first derivative operator
- zero crossings of a second derivative operator

Two Generic Approaches for Edge Detection
Marr / Hildreth **Laplacian of Gaussian**

A “zero crossings” of a second derivative operator” approach

**Steps:**

1. Gaussian for smoothing

2. Laplacian ($\nabla^2$) for differentiation where

\[
\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}
\]

3. Locate zero-crossings in the Laplacian of the Gaussian ($\nabla^2 G$) where

\[
\nabla^2 G(x, y) = \frac{-1}{2\pi\sigma^4} \left[ 2 - \frac{x^2 + y^2}{\sigma^2} \right] \exp\left(\frac{-x^2+y^2}{2\sigma^2}\right)
\]
Here’s a 3D plot of the Laplacian of the Gaussian (\( \nabla^2 G \))

... with its characteristic “Mexican hat” shape
Canny Edge Detector

Steps:

1. Apply **directional derivatives** of Gaussian

2. Compute **gradient magnitude** and **gradient direction**

3. **Non-maximum** suppression
   - thin multi-pixel wide “ridges” down to single pixel width

4. **Linking** and thresholding
   - Low, high edge-strength thresholds
   - Accept all edges over low threshold that are connected to edge over high threshold
Why is non-maximum suppression applied in the Canny edge detector?
What is a corner?

We can think of a corner as any locally distinct 2D image feature that (hopefully) corresponds to a distinct position on an 3D object of interest in the scene.

Image Credit: John Shakespeare, Sydney Morning Herald
Why are corners **distinct**?

A corner can be **localized reliably**.

Thought experiment:

— Place a small window over a patch of constant image value. If you slide the window in any direction, the image in the window will not change.

— Place a small window over an edge. If you slide the window in the direction of the edge, the image in the window will not change
  
  → Cannot estimate location along an edge (a.k.a., **aperture** problem)

— Place a small window over a corner. If you slide the window in any direction, the image in the window changes.

*Image Credit: Ioannis (Yannis) Gkioulekas (CMU)*
Autocorrelation

Szeliski, Figure 4.5
Corner Detection

Edge detectors perform poorly at corners

**Observations:**
- The gradient is ill defined exactly at a corner
- Near a corner, the gradient has two (or more) distinct values
Harris Corner Detection

1. Compute image gradients over small region
2. Compute the covariance matrix
3. Compute eigenvectors and eigenvalues
4. Use threshold on eigenvalues to detect corners
2. Compute the **covariance matrix** (a.k.a. 2nd moment matrix)

\[ C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix} \]

- **Sum** over small region around the corner
- **Gradient** with respect to x, times gradient with respect to y

Matrix is **symmetric**
Harris Corner Detection

- Filter image with **Gaussian**
- Compute magnitude of the x and y **gradients** at each pixel
- Construct C in a window around each pixel
  - Harris uses a **Gaussian window**
- Solve for product of the λ’s
- If λ’s both are big (product reaches local maximum above threshold) then we have a corner
  - Harris also checks that ratio of λs is not too high
Compute the **covariance matrix** (a.k.a. 2nd moment matrix)

**Sum** over small region around the corner

\[ C = \begin{bmatrix}
\sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\
\sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y
\end{bmatrix} \]

**Gradient** with respect to x, times gradient with respect to y

\[ I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y} \]

\[ \sum_{p \in P} I_x I_y = \text{sum}(\text{array of x gradients} \times \text{array of y gradients}) \]
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:

```
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 1 1 1 1 0 0
0 1 1 1 1 0 0
0 0 1 1 1 0 0
0 0 1 1 1 0 0
0 0 1 1 1 0 0
0 0 1 1 1 0 0
```

This grid represents the surrounding pixels of the green pixel, and we can calculate the corner-ness measure using these values.
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:

\[ I_x = \frac{\partial I}{\partial x} \]
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:

\[ I_x = \frac{\partial I}{\partial x} \]

\[ I_y = \frac{\partial I}{\partial y} \]
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:

\[ \sum \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix} = 3 \]

\[ I_x = \frac{\partial I}{\partial x} \]

\[ I_y = \frac{\partial I}{\partial y} \]
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:

\[ C = \begin{bmatrix} 3 & 2 \\ 2 & 4 \end{bmatrix} \]

\[ I_x = \frac{\partial I}{\partial x} \]

\[ I_y = \frac{\partial I}{\partial y} \]
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:

\[
C = \begin{bmatrix}
3 & 2 \\
2 & 4 \\
\end{bmatrix} \Rightarrow \lambda_1 = 1.4384; \lambda_2 = 5.5616
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & -1 & 1 \\
-1 & 0 & 0 & 0 & 1 & 0 \\
-1 & 0 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & -1 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & -1 & -1 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}
\]
Example: Harris Corner Detection

Let's compute a measure of "corner-ness" for the green pixel:

\[
C = \begin{bmatrix}
  3 & 2 \\
  2 & 4 \\
\end{bmatrix} \Rightarrow \lambda_1 = 1.4384; \lambda_2 = 5.5616
\]

\[
\text{det}(C) - 0.04\text{trace}^2(C) = 6.04
\]
Example: Harris Corner Detection

Let's compute a measure of “corner-ness” for the green pixel:

\[ C = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix} \]

=> \( \lambda_1 = 3; \lambda_2 = 0 \)

\[ \text{det}(C) - 0.04 \text{trace}^2(C) = -0.36 \]
Example: Harris Corner Detection

Let's compute a measure of "corner-ness" for the green pixel:

$$C = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \implies \lambda_1 = 3; \lambda_2 = 2$$

$$\det(C) - 0.04\text{trace}^2(C) = 5$$
Properties: NOT Invariant to Scale Changes

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Characteristic Scale

characteristic scale - the scale that produces peak filter response

we need to search over characteristic scales
Sample Questions: Corners

The Harris corner detector is stable under some image transformations (features are considered stable if the same locations on an object are typically selected in the transformed image).

**True** or **false**: The Harris corner detector is stable under image blur.
Texture

We will look at two main questions:

1. How do we represent texture?
   → Texture **analysis**

2. How do we generate new examples of a texture?
   → Texture **synthesis**
Why might we want to synthesize texture?

1. To fill holes in images (inpainting)
   — Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.
   — We need to find something to put in place of the pixels that were removed
   — We synthesize regions of texture that fit in and look convincing

2. To produce large quantities of texture for computer graphics
   — Good textures make object models look more realistic
Texture **Synthesis**

Szeliski, Fig. 10.49
Bush campaign digitally altered TV ad

President Bush’s campaign acknowledged Thursday that it had digitally altered a photo that appeared in a national cable television commercial. In the photo, a handful of soldiers were multiplied many times.

This section shows a sampling of the duplication of soldiers.

Original photograph

Photo Credit: Associated Press
— What is **conditional** probability distribution of $p$, given the neighbourhood window?

— Directly search the input image for all such neighbourhoods to produce a **histogram** for $p$

— To **synthesize** $p$, pick one match at random
Efros and Leung: Synthesizing One Pixel

— Since the sample image is finite, an exact neighbourhood match might not be present

— Find the **best match** using SSD error, weighted by Gaussian to emphasize local structure, and take all samples within some distance from that match
For multiple pixels, "grow" the texture in layers
— In the case of hole-filling, start from the edges of the hole

For an interactive demo, see
(written by Julieta Martinez, a previous CPSC 425 TA)
Randomness Parameter

Slide Credit: http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt
“Big Data” Meets Inpainting

Original Image

Input

Figure Credit: Hays and Efros 2007
“Big Data” Meets Inpainting

Scene Matches

Input              Scene Matches              Output

Figure Credit: Hays and Efros 2007
Algorithm sketch (Hays and Efros 2007):

1. Create a short list of a few hundred “best matching" images based on global image statistics

2. Find patches in the short list that match the context surrounding the image region we want to fill

3. Blend the match into the original image

Purely data-driven, requires no manual labeling of images
Goal of Texture **Analysis**

Compare textures and decide if they’re made of the same “stuff”

Credit: Bill Freeman
**Texture Representation**

**Observation:** Textures are made up of generic sub-elements, repeated over a region with similar statistical properties.

**Idea:** Find the sub-elements with filters, then represent each point in the image with a summary of the pattern of sub-elements in the local region.
Texture Representation

**Observation:** Textures are made up of generic sub-elements, repeated over a region with similar statistical properties.

**Idea:** Find the sub-elements with filters, then represent each point in the image with a summary of the pattern of sub-elements in the local region.

**Question:** What filters should we use?

**Answer:** Human vision suggests spots and oriented edge filters at a variety of different orientations and scales.
Texture Representation

**Observation**: Textures are made up of generic sub-elements, repeated over a region with similar statistical properties

**Idea**: Find the sub-elements with filters, then represent each point in the image with a summary of the pattern of sub-elements in the local region

**Question**: What filters should we use?

**Answer**: Human vision suggests spots and oriented edge filters at a variety of different orientations and scales

**Question**: How do we “summarize”?

**Answer**: Compute the mean or maximum of each filter response over the region — Other statistics can also be useful
Texture Representation

Figure Credit: Leung and Malik, 2001
Spots and Bars (Fine Scale)

Forsythe & Ponce (1st ed.) Figures 9.3–9.4
Spots and Bars (Coarse Scale)

Forsyth & Ponce (1st ed.) Figures 9.3 and 9.5
Laplacian Pyramid
Laplacian Pyramid

Building a **Laplacian** pyramid:
- Create a Gaussian pyramid
- Take the difference between one Gaussian pyramid level and the next (before subsampling)

**Properties**
- Also known as the difference-of-Gaussian (DOG) function, a close approximation to the Laplacian
- It is a band pass filter – each level represents a different band of spatial frequencies

**Reconstructing** the original image:
- Reconstruct the Gaussian pyramid starting at top
Constructing a *Laplacian* Pyramid

**Algorithm**

repeat:
  filter
  compute residual
  subsample
until min resolution reached

*Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)*
Reconstructing the Original Image

Algorithm
repeat:
  upsample
  sum with residual
until orig resolution reached

Slide Credit: Ioannis (Yannis) Gkioulkas (CMU)
Gaussian vs Laplacian Pyramid

Shown in opposite order for space

Which one takes more space to store?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Oriented Pyramids

Forsyth & Ponce (1st ed.) Figure 9.13
Steps:

1. Form a Laplacian and oriented pyramid (or equivalent set of responses to filters at different scales and orientations)

2. Square the output (makes values positive)

3. Average responses over a neighborhood by blurring with a Gaussian

4. Take statistics of responses
   - Mean of each filter output
   - Possibly standard deviation of each filter
Sample Question: Texture

How does the top-most image in a Laplacian pyramid differ from the others?