Lecture 13: Laplacian Pyramids, Corner Detection
Menu for Today (October 7, 2020)

Topics:
- Laplacian Pyramids (revisited)
- Corner Detection
- Autocorrelation
- Harris Corner Detector

Readings:
- Today's Lecture: Forsyth & Ponce (2nd ed.) 5.3.0 - 5.3.1
- Next Lecture: Forsyth & Ponce (2nd ed.) 6.1, 6.3

Reminders:
- Quiz 2: due at the end of day today
- Assignment 2: Face Detection in a Scaled Representation is October 14th
Today’s “fun” Example:

Wait for it! :)

Lecture 12: Re-cap

Physical properties of a 3D scene cause “edges” in an image:
- depth discontinuity
- surface orientation discontinuity
- reflectance discontinuity
- illumination boundaries

Two generic approaches to edge detection:
- local extrema of a first derivative operator → Canny
- zero crossings of a second derivative operator → Marr/Hildreth

Many algorithms consider “boundary detection” as a high-level recognition task and output a probability or confidence that a pixel is on a human-perceived boundary
What happens to the details?
— They get smoothed out as we move to higher levels

What is preserved at the higher levels?
— Mostly large uniform regions in the original image

How would you reconstruct the original image from the image at the upper level?
— That's not possible
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
Laplacian Pyramid

At each level, retain the residuals instead of the blurred images themselves.

Why is it called Laplacian Pyramid?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Why Laplacian Pyramid?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Laplacian is a Bandpass Filter

![Diagram showing the process of applying a Laplacian filter to an image through Fourier Transform (FFT), low-pass filtering, and complex element-wise multiplication.]
Laplacian is a Bandpass Filter

- **Image**
- **FFT (Mag)**
- **Low pass**
  - **complex element-wise multiplication**
  - **lower sigma**
  - **filtered image**
- **FFT (Mag)**
- **Low pass**
  - **complex element-wise multiplication**
  - **larger sigma**
  - **filtered image**
Laplacian is a Bandpass Filter
**Laplacian Pyramid**

At each level, retain the residuals instead of the blurred images themselves.

**Why is it called Laplacian Pyramid?**

Can we reconstruct the original image using the pyramid?  
— Yes we can!
Laplacian Pyramid

At each level, retain the residuals instead of the blurred images themselves.

Why is it called Laplacian Pyramid?

Can we reconstruct the original image using the pyramid?
— Yes we can!

What do we need to store to be able to reconstruct the original image?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Let’s start by just looking at **one level**

\[ \text{level 0} = \text{level 1 (upsampled)} + \text{residual} \]

Does this mean we need to store both residuals and the blurred copies of the original?

*Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)*
Constructing a **Laplacian** Pyramid

Algorithm

```
repeat:
    filter
    compute residual
    subsample
until min resolution reached
```
Constructing a Laplacian Pyramid

Algorithm

repeat:
  filter
  compute residual
  subsample
until min resolution reached
Constructing a **Laplacian** Pyramid

**Algorithm**

repeat:
  filter
  compute residual
  subsample
until min resolution reached
Constructing a **Laplacian Pyramid**

**Algorithm**

repeat:
  filter
  compute residual
  subsample
until min resolution reached
Constructing a **Laplacian Pyramid**

It's a Gaussian Pyramid

**Algorithm**

repeat:
  filter
  compute residual
  subsample
until min resolution reached
Constructing a **Laplacian** Pyramid

It’s a Gaussian Pyramid

**Algorithm**

repeat:
  filter
  compute residual
  subsample
until min resolution reached

This is a Laplacian Pyramid

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

Algorithm:

repeat:
  upsample
  sum with residual
until orig resolution reached
Gaussian vs Laplacian Pyramid

Which one takes more space to store?

Shown in opposite order for space

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Aside: Image Blending

**Aside: Image Blending**

Aside: Image Blending

https://becominghuman.ai/image-blending-using-laplacian-pyramids-2f8e9982077f
Aside: Image Blending
Aside: Image Blending

https://becominghuman.ai/image-blending-using-laplacian-pyramids-2f8e9982077f
High-level Intuition: Smoother blending of flatter regions, sharper blending of more detailed regions
Aside: Image Blending

**Algorithm:**

1. Build Laplacian pyramid LA and LB from images A and B

2. Build a Gaussian pyramid GR from mask image R (the mask defines which image pixels should be coming from A or B)

3. From a combined (blended) Laplacian pyramid LS, using nodes of GR as weights: $LS(i,j) = GR(i,j) \times LA(i,j) + (1-GR(i,j)) \times LB(i,j)$

4. Reconstruct the final blended image from LS
Aside: Image Blending

Left

Right

Mask

Blended
Aside: Image Blending

© david dmartin (Boston College)
Aside: Image Blending

© Chris Cameron
Today’s “fun” Example: Eulerian Video Magnification

Video From: Wu at al., Siggraph 2012
Today’s “fun” Example: Eulerian Video Magnification

Source

Motion-amplified (x10)

Video From: Wu at al., Siggraph 2012
Today’s “fun” Example: Eulerian Video Magnification

Figure From: Wu at al., Siggraph 2012
Lecture 13: Corner Detection

( unless otherwise stated slides are taken or adopted from Bob Woodham, Jim Little and Fred Tung )
When might **template matching fail**?

- Different scales
- Different orientation
- Lighting conditions
- Left vs. Right hand
- Partial Occlusions
- Different Perspective
- Motion / blur
Motivation: Template Matching in Scaled Representation

When might template matching in scaled representation fail?

- Different scales
- Different orientation
- Lighting conditions
- Left vs. Right hand
- Partial Occlusions
- Different Perspective
- Motion / blur
Motivation: Edge Matching in Scaled Representation

When might edge matching in scaled representation fail?

- Different scales
- Different orientation
- Lighting conditions
- Left vs. Right hand
- Partial Occlusions
- Different Perspective
- Motion / blur
Motivation: Edge Matching in Scaled Representation

- Different scales
- Different orientation
- Lighting conditions
- Left vs. Right hand

- Partial Occlusions
- Different Perspective
- Motion / blur
Planar Object **Instance Recognition**

Database of planar objects

Instance recognition

**Slide Credit:** Ioannis (Yannis) Gkioulekas (CMU)
Recognition under **Occlusion**

*Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)*
Image Matching

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
Image Matching
Finding **Correspondences**

NASA Mars Rover images

*Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)*
Finding **Correspondences**
What is a **Good Feature**?

Pick a point in the image.
Find it again in the next image.
What is a **Good Feature**?

Pick a point in the image.
Find it again in the next image.

*Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)*
What is a **Good Feature**?

Pick a point in the image. Find it again in the next image.

*Slide Credit:* Ioannis (Yannis) Gkioulekas (CMU)
What is a **Good Feature**?

**Local**: features are local, robust to occlusion and clutter

**Accurate**: precise localization

**Robust**: noise, blur, compression, etc. do not have a big impact on the feature.

**Distinctive**: individual features can be easily matched

**Efficient**: close to real-time performance
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
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:
Why are corners **distinct**?

A corner can be **localized reliably**.

Thought experiment:

— Place a small window over a patch of constant image value.
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.

“flat” region: no change in all directions
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.

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

“edge”: no change along the edge direction

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
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.

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
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)*
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
How do you find a corner?

Easily recognized by looking through a small window

Shifting the window should give large change in intensity

[Moravec 1980]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
**Autocorrelation** is the correlation of the image with itself.

- Windows centered on an edge point will have autocorrelation that falls off slowly in the direction along the edge and rapidly in the direction across (perpendicular to) the edge.

- Windows centered on a corner point will have autocorrelation that falls off rapidly in all directions.
Autocorrelation

Szeliski, Figure 4.5
Autocorrelation

Szeliski, Figure 4.5
Autocorrelation

Szeliski, Figure 4.5
Autocorrelation

Szeliski, Figure 4.5
Szeliski, Figure 4.5
Autocorrelation

Szeliski, Figure 4.5
Autocorrelation is the correlation of the image with itself.

— Windows centered on an edge point will have autocorrelation that falls off slowly in the direction along the edge and rapidly in the direction across (perpendicular to) the edge.

— Windows centered on a corner point will have autocorrelation that falls of rapidly in all directions.
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

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

\[
\begin{bmatrix}
\sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\
\sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y
\end{bmatrix}
\]
1. Compute **image gradients** over a small region (not just a single pixel)

array of x gradients

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

array of y gradients

\[ I_y = \frac{\partial I}{\partial y} \]

*Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)*
Visualization of Gradients

image

X derivative

Y derivative

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)
What Does a **Distribution** Tells You About the **Region**?

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

**Slide Credit:** Ioannis (Yannis) Gkioulekas (CMU)
What Does a **Distribution** Tells You About the **Region**?

\[ I_y = \frac{\partial I}{\partial y} \]

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

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)
What Does a **Distribution** Tells You About the **Region**?

Distribution reveals the **orientation** and **magnitude**
What Does a **Distribution** Tells You About the **Region**?

Distribution reveals the **orientation** and **magnitude**

How do we quantify the **orientation** and **magnitude**?

*Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)*
2. Compute the **covariance matrix** (a.k.a. 2nd moment matrix)

\[
C = \begin{bmatrix}
\sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\
\sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y
\end{bmatrix}
\]
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
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
2. 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

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

\[
I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}
\]
2. 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

Matrix is **symmetric**
2. Compute the **covariance matrix** (a.k.a. 2nd moment matrix)

By computing the **gradient covariance matrix** ...

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

we are fitting a **quadratic** to the gradients over a small image region
Simple Case

Local Image Patch

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

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

Local Image Patch

(high value along vertical strip of pixels and 0 elsewhere)

$$\begin{align*}
C &= \begin{bmatrix}
\sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\
\sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y
\end{bmatrix} = ?
\end{align*}$$
Simple Case

Local Image Patch

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

high value along vertical strip of pixels and 0 elsewhere

high value along horizontal strip of pixels and 0 elsewhere
Simple Case

Local Image Patch

\[ C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \]

\( I_x \) high value along vertical strip of pixels and 0 elsewhere

\( I_y \) high value along horizontal strip of pixels and 0 elsewhere
General Case

It can be shown that since every $C$ is symmetric:

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

... so general case is like a rotated version of the simple one.
3. Computing **Eigenvalues** and **Eigenvectors**
Given a square matrix $A$, a scalar $\lambda$ is called an eigenvalue of $A$ if there exists a nonzero vector $v$ that satisfies

$$A v = \lambda v$$

The vector $v$ is called an eigenvector for $A$ corresponding to the eigenvalue $\lambda$.

The eigenvalues of $A$ are obtained by solving

$$\det(A - \lambda I) = 0$$
3. Computing **Eigenvalues** and **Eigenvectors**

- **Eigenvalue**
  - \( Ce = \lambda e \)
  - \((C - \lambda I)e = 0\)

**Slide Credit:** Ioannis (Yannis) Gkioulekas (CMU)
3. Computing **Eigenvalues** and **Eigenvectors**

- **Eigenvalue**
  - $Ce = \lambda e$

- **Eigenvector**
  - $(C - \lambda I)e = 0$

1. Compute the determinant of 
  (returns a polynomial)

$$C - \lambda I$$
3. Computing **Eigenvalues** and **Eigenvectors**

**eigenvector**

\[ Ce = \lambda e \]

**eigenvalue**

\[ (C - \lambda I)e = 0 \]

1. Compute the determinant of
   (returns a polynomial)

\[ \det(C - \lambda I) = 0 \]

2. Find the roots of polynomial
   (returns eigenvalues)

\[ C - \lambda I \]
### 3. Computing Eigenvalues and Eigenvectors

Eigenvector \( C e = \lambda e \)

Eigenvector \((C - \lambda I)e = 0\)

#### 1. Compute the determinant of

\( C - \lambda I \)

*returns a polynomial*

#### 2. Find the roots of polynomial

\( \det(C - \lambda I) = 0 \)

*returns eigenvalues*

#### 3. For each eigenvalue, solve

\( (C - \lambda I)e = 0 \)

*returns eigenvectors*

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

$$C = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

1. Compute the determinant of (returns a polynomial) $C - \lambda I$

2. Find the roots of polynomial (returns eigenvalues) $\det(C - \lambda I) = 0$

3. For each eigenvalue, solve (returns eigenvectors) $(C - \lambda I)e = 0$
Example

\[ C = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \]

\[ \det \left( \begin{bmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{bmatrix} \right) \]

1. Compute the determinant of \( C - \lambda I \) (returns a polynomial)

2. Find the roots of polynomial \( \det(C - \lambda I) = 0 \) (returns eigenvalues)

3. For each eigenvalue, solve \( (C - \lambda I)e = 0 \) (returns eigenvectors)

\[ \text{Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)} \]
Example

\[ C = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \]

\[ \text{det} \left( \begin{bmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{bmatrix} \right) \]

\[ (2 - \lambda)(2 - \lambda) - (1)(1) \]

1. Compute the determinant of \( C - \lambda I \) (returns a polynomial)

\[ C - \lambda I \]

2. Find the roots of polynomial \( \det(C - \lambda I) = 0 \) (returns eigenvalues)

\[ \det(C - \lambda I) = 0 \]

3. For each eigenvalue, solve \( (C - \lambda I)e = 0 \) (returns eigenvectors)

\[ (C - \lambda I)e = 0 \]
Example

\[ C = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \]

\[
\det \left( \begin{bmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{bmatrix} \right) = (2 - \lambda)(2 - \lambda) - (1)(1) = 0
\]

1. Compute the determinant of (returns a polynomial)
   \[ C - \lambda I \]

2. Find the roots of polynomial (returns eigenvalues)
   \[ \det(C - \lambda I) = 0 \]

3. For each eigenvalue, solve (returns eigenvectors)
   \[ (C - \lambda I) e = 0 \]
Example

\[ C = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \]

\[
\text{det} \left( \begin{bmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{bmatrix} \right) = (2 - \lambda)(2 - \lambda) - (1)(1) = 0
\]

1. Compute the determinant of \( C \) (returns a polynomial)

2. Find the roots of polynomial (returns eigenvalues)

3. For each eigenvalue, solve \( (C - \lambda I)e = 0 \) (returns eigenvectors)

\[
\lambda^2 - 4\lambda + 3 = 0
\]

\[
(\lambda - 3)(\lambda - 1) = 0
\]

\[
\lambda_1 = 1, \lambda_2 = 3
\]

\[ C - \lambda I \]

\[ \text{det}(C - \lambda I) = 0 \]