Welcome back!

• You may need to log in to Gatech’s Github for us to be able to add you to the classroom Github group. You can debug this with project 0.
• Project 1 will be out soon.
• Read Szeliski 2.1, especially 2.1.4
• First quiz will be Thursday, October 6th.
• Today
  – Image projection
  – Filtering
From the 3D to 2D

\[ P = [x, y, z] \]

\[ p = [x, y] \]

Image

3D world

Slide credit Fei Fei Li
Chapter 2

Image formation

2.1 Geometric primitives and transformations ........................................ 36
  2.1.1 2D transformations .................................................. 39
  2.1.2 3D transformations .................................................. 43
  2.1.3 3D rotations ......................................................... 45
  2.1.4 3D to 2D projections ............................................... 50
  2.1.5 Lens distortions ..................................................... 62
2.2 Photometric image formation ....................................................... 64
  2.2.1 Lighting ............................................................... 65
  2.2.2 Reflectance and shading ........................................... 66
  2.2.3 Optics ................................................................. 73
2.3 The digital camera ................................................................. 78
  2.3.1 Sampling and aliasing .............................................. 82
  2.3.2 Color ................................................................. 85
  2.3.3 Compression ......................................................... 97
2.4 Additional reading ............................................................... 98
2.5 Exercises ................................................................. 99
Projective Geometry

What is lost?

- Length
Length and area are not preserved
Projective Geometry

What is lost?

• Length
• Angles
Projective Geometry

What is preserved?
• Straight lines are still straight
If $x = 2$, $y = 3$, $z = 5$, and $f = 2$
What are $u'$ and $v'$?

$$
u' = -x \frac{f}{z}$$  
$$v' = -y \frac{f}{z}$$

$$u' = -2 \times \frac{2}{5}$$  
$$v' = -3 \times \frac{2}{5}$$
Projection: world coordinates $\rightarrow$ image coordinates

How do we handle the general case?
Interlude: why does this matter?
Relating multiple views
Photo Tourism
Exploring photo collections in 3D

Noah Snavely  Steven M. Seitz  Richard Szeliski
University of Washington  Microsoft Research

SIGGRAPH 2006
Projection: world coordinates $\rightarrow$ image coordinates

How do we handle the general case?
Homogeneous coordinates

Conversion

Converting to *homogeneous* coordinates

\[
(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

homogeneous image coordinates

\[
(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

homogeneous scene coordinates

Converting *from* homogeneous coordinates

\[
\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)
\]

\[
\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)
\]
Homogeneous coordinates

Invariant to scaling

\[ k \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} kx \\ ky \\ kw \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{kx}{kw} \\ \frac{ky}{kw} \\ \frac{kw}{kw} \end{bmatrix} = \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \end{bmatrix} \]

Homogeneous Coordinates  Cartesian Coordinates

Point in Cartesian is ray in Homogeneous
Projection matrix

\[ \mathbf{x} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{X} \]

- \( \mathbf{x} \): Image Coordinates: \((u,v,1)\)
- \( \mathbf{K} \): Intrinsic Matrix (3x3)
- \( \mathbf{R} \): Rotation (3x3)
- \( \mathbf{t} \): Translation (3x1)
- \( \mathbf{X} \): World Coordinates: \((X,Y,Z,1)\)
Projection matrix

Intrinsic Assumptions
- Unit aspect ratio
- Optical center at (0,0)
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

\[
x = K[I \ 0]X \\
\begin{bmatrix}
u \\
v \\
w \\
1
\end{bmatrix} =
\begin{bmatrix}
f & 0 & 0 & 0 \\
0 & f & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]
Projection matrix

Intrinsic Assumptions
• Unit aspect ratio
• Optical center at (0,0)
• No skew

Extrinsic Assumptions
• No rotation
• Camera at (0,0,0)

\[ \mathbf{x} = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{X} \]

\[
\begin{bmatrix}
  u \\
  v \\
  1
\end{bmatrix} =
\begin{bmatrix}
  f & 0 & 0 & 0 \\
  0 & f & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

Slide Credit: Savarese
Remove assumption: known optical center

Intrinsic Assumptions
• Unit aspect ratio
• No skew

Extrinsic Assumptions
• No rotation
• Camera at (0,0,0)

\[ x = K [I \ 0] X \]

\[
\begin{bmatrix}
  u \\
v \\
1
\end{bmatrix}
= \begin{bmatrix}
f & 0 & u_0 & 0 \\
0 & f & v_0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]
Intrinsic Assumptions
• No skew

Extrinsic Assumptions
• No rotation
• Camera at (0,0,0)

\[
x = K [I \ 0] X
\]
Remove assumption: non-skewed pixels

Intrinsic Assumptions
Extrinsic Assumptions
• No rotation
• Camera at (0,0,0)

\[
x = K[I \ 0] X
\]

\[
\begin{bmatrix}
u \\
v \\
1
\end{bmatrix} =
\begin{bmatrix}
\alpha & s & u_0 & 0 \\
0 & \beta & v_0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

Note: different books use different notation for parameters
Oriented and Translated Camera
Intrinsic Assumptions
Extrinsic Assumptions
• No rotation

\[ x = K[I \ t]X \]

\[
\begin{bmatrix}
  u \\
v \\
w
\end{bmatrix} =
\begin{bmatrix}
\alpha & 0 & u_0 \\
0 & \beta & v_0 \\
1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & t_z \\
1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]
3D Rotation of Points

Rotation around the coordinate axes, \textit{counter-clockwise}:

\[
R_x (\alpha) = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \alpha & -\sin \alpha \\
0 & \sin \alpha & \cos \alpha
\end{bmatrix}
\]

\[
R_y (\beta) = \begin{bmatrix}
\cos \beta & 0 & \sin \beta \\
0 & 1 & 0 \\
-\sin \beta & 0 & \cos \beta
\end{bmatrix}
\]

\[
R_z (\gamma) = \begin{bmatrix}
\cos \gamma & -\sin \gamma & 0 \\
\sin \gamma & \cos \gamma & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
Allow camera rotation

\[ \mathbf{x} = \mathbf{K} [\mathbf{R} \quad \mathbf{t}] \mathbf{X} \]
Degrees of freedom

\[
\begin{align*}
x &= \mathbf{K}[\mathbf{R} \quad \mathbf{t}] \mathbf{X} \\
\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\end{align*}
\]
Field of View (Zoom, focal length)

From London and Upton
Beyond Pinholes: Radial Distortion

Image from Martin Habbecke

Corrected Barrel Distortion
Things to remember

- Vanishing points and vanishing lines
- Pinhole camera model and camera projection matrix
- Homogeneous coordinates

\[ x = K[R \quad t]X \]

\[(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \]
Reminder: read your book

- Lectures have assigned readings
- Szeliski 2.1 and especially 2.1.4 cover the geometry of image formation
Image Filtering

Computer Vision

James Hays

Many slides by Derek Hoiem
From the 3D to 2D

- Let’s now focus on 2D
- Extract building blocks
Extract useful building blocks
Hybrid Images

Upcoming classes: two views of filtering

• Image filters in spatial domain
  – Filter is a mathematical operation of a grid of numbers
  – Smoothing, sharpening, measuring texture

• Image filters in the frequency domain
  – Filtering is a way to modify the frequencies of images
  – Denoising, sampling, image compression
Image filtering (or convolution)

• Image filtering: compute function of local neighborhood at each position

• Really important!
  – Enhance images
    • Denoise, resize, increase contrast, etc.
  – Extract information from images
    • Texture, edges, distinctive points, etc.
  – Detect patterns
    • Template matching
  – Deep Convolutional Networks
Example: box filter

\[ g[\cdot,\cdot] \]

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

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

\[ g[\cdot, \cdot] = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \]

\[ f[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]
Image filtering

\[ f[\cdot,\cdot] \]

\[ h[\cdot,\cdot] \]

\[
h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l]
\]

Credit: S. Seitz
Image filtering

\[ f[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

\[ h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l] \]

Credit: S. Seitz
Image filtering

\[ f[\ldots] \]

\[ h[\ldots] \]

\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \]

Credit: S. Seitz
Image filtering

\[ f[\cdot, \cdot] \quad \text{and} \quad h[\cdot, \cdot] \]

\[
h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]
\]

Credit: S. Seitz
Image filtering

\[ f[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

\[
\begin{align*}
  h[m, n] &= \sum_{k, l} g[k, l] f[m + k, n + l]
\end{align*}
\]

Credit: S. Seitz
Image filtering

$$f[\cdot,\cdot]$$

$$h[\cdot,\cdot]$$

$$h[m,n] = \sum_{k,l} g[k,l] \, f[m+k,n+l]$$

Credit: S. Seitz
Image filtering

\[ f[\cdot,\cdot] \]

\[ g[\cdot,\cdot] \]

\[ h[\cdot,\cdot] \]

\[ h[m,n] = \sum_{k,l} g[k,l] \cdot f[m+k,n+l] \]
What does it do?
• Replaces each pixel with an average of its neighborhood
• Achieve smoothing effect (remove sharp features)
Smoothing with box filter
Practice with linear filters

Original

Source: D. Lowe
Practice with linear filters

Original

Filtered
(no change)

Source: D. Lowe
Practice with linear filters

Original

Source: D. Lowe
Practice with linear filters

Original

Shifted left
By 1 pixel

Source: D. Lowe
Practice with linear filters

Original

(Note that filter sums to 1)
Practice with linear filters

Original

Sharpening filter
- Accentuates differences with local average

Source: D. Lowe
Sharpening

Source: D. Lowe
Other filters

Sobel

Vertical Edge (absolute value)
Other filters

Sobel

Horizontal Edge (absolute value)
Filtering vs. Convolution

- **2d filtering**

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k, n+l] \]

- **2d convolution**

\[ h[m,n] = \sum_{k,l} f[k,l] I[m-k, n-l] \]

In Python you can use https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve2d.html
Key properties of linear filters

**Linearity:**
\[
imfilter(I, f_1 + f_2) = imfilter(I,f_1) + imfilter(I,f_2)
\]

**Shift invariance:** same behavior regardless of pixel location
\[
imfilter(I,\text{shift}(f)) = \text{shift}(imfilter(I,f))
\]

Any linear, shift-invariant operator can be represented as a convolution

Source: S. Lazebnik
More properties

• Commutative: \( a * b = b * a \)
  – Conceptually no difference between filter and signal
  – But particular filtering implementations might break this equality

• Associative: \( a * (b * c) = (a * b) * c \)
  – Often apply several filters one after another: \(((a * b_1) * b_2) * b_3)\)
  – This is equivalent to applying one filter: \(a * (b_1 * b_2 * b_3)\)

• Distributes over addition: \( a * (b + c) = (a * b) + (a * c) \)

• Scalars factor out: \( ka * b = a * kb = k \left( a * b \right) \)

• Identity: unit impulse \( e = [0, 0, 1, 0, 0] \),
  \( a * e = a \)

Source: S. Lazebnik
Important filter: Gaussian

- Weight contributions of neighboring pixels by nearness

\[
G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2 + y^2)}{2\sigma^2}}
\]

<table>
<thead>
<tr>
<th>0.003</th>
<th>0.013</th>
<th>0.022</th>
<th>0.013</th>
<th>0.003</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.013</td>
<td>0.059</td>
<td>0.097</td>
<td>0.059</td>
<td>0.013</td>
</tr>
<tr>
<td>0.022</td>
<td>0.097</td>
<td>0.159</td>
<td>0.097</td>
<td>0.022</td>
</tr>
<tr>
<td>0.013</td>
<td>0.059</td>
<td>0.097</td>
<td>0.059</td>
<td>0.013</td>
</tr>
<tr>
<td>0.003</td>
<td>0.013</td>
<td>0.022</td>
<td>0.013</td>
<td>0.003</td>
</tr>
</tbody>
</table>

5 x 5, \( \sigma = 1 \)

Slide credit: Christopher Rasmussen
Smoothing with Gaussian filter
Smoothing with box filter
Gaussian filters

• Remove “high-frequency” components from the image (low-pass filter)
  – Images become more smooth

• Convolution with self is another Gaussian
  – So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  – Convolving two times with Gaussian kernel of width \( \sigma \) is same as convolving once with kernel of width \( \sigma \sqrt{2} \)

• *Separable* kernel
  – Factors into product of two 1D Gaussians

Source: K. Grauman
Separability of the Gaussian filter

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

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

The 2D Gaussian can be expressed as the product of two functions, one a function of \( x \) and the other a function of \( y \).

In this case, the two functions are the (identical) 1D Gaussian filter.

Source: D. Lowe
Separability example

2D convolution (center location only)

The filter factors into a product of 1D filters:

Perform convolution along rows:

Followed by convolution along the remaining column:

Source: K. Grauman
Separability

- Why is separability useful in practice?
Some practical matters
Practical matters

How big should the filter be?

• Values at edges should be near zero
• Rule of thumb for Gaussian: set filter half-width to about 3 $\sigma$
Practical matters

• What about near the edge?
  – the filter window falls off the edge of the image
  – need to extrapolate
  – methods:
    • clip filter (black)
    • wrap around
    • copy edge
    • reflect across edge
To be continued...
Next class: Light and Color and
Thinking in Frequency