中国科学技术大学
University of Science and Technology of China

Introduction

Juyong Zhang
School of Mathematics, USTC

The Course

- Learn how to solve math problems with tools
- Matlab, Mathematica, Python, Eigen, Ceres, etc…
- Grading policy:
  - Homework&Programming: 80%
  - Final Project: 20%

Covered Topics

- Image Processing, Image Filtering
- Face Detection, Face Recognition
- Image Stitching, Image Warping
- Tracking, Optical Flow,
- Stereo Matching, Epipolar Geometry
- Structure From Motion, 3D Surface Reconstruction
- Neural Network, etc…

TA & QQ Group

- 助教
  - 杨乐园（ ly_1207@mail.ustc.edu.cn ）
  - 许子航（ ad123456@mail.ustc.edu.cn ）
- 课程QQ群

USTC数学实验20...

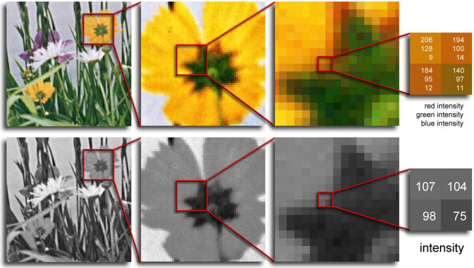
群号：827829466

Digital Image

Color images have 3 values per pixel; monochrome images have 1 value per pixel.

a grid of squares, each of which contains a single color

each square is called a pixel (for *picture element*)



Pixels

- A digital image, *I*, is a mapping from a 2D grid of uniformly spaced discrete points, $\{p = (r,c)\}$, into a set of positive integer values, $\{I(p)\}$, or a set of vector values, *e.g.*, $\{[R\ G\ B]^T(p)\}$.

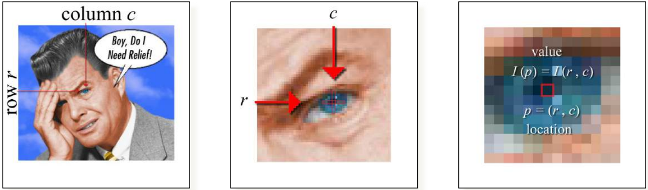
- At each column location in each row of *I* there is a value.

- The pair (p, $I(p)$) is called a “pixel” (for *picture element*).

Pixels

- p = (r,c) is the pixel location indexed by row, r, and column, c.
- I(p) = I(r,c) is the value of the pixel at location p.
- If I(p) is a single number then I is monochrome.
- If I(p) is a vector (ordered list of numbers) then I has multiple bands (e.g., a color image).

Pixels



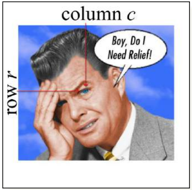
Pixel Location: $p = (r , c)$

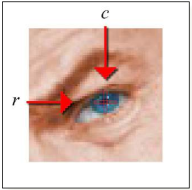
Pixel Value: $I(p) = I(r , c)$

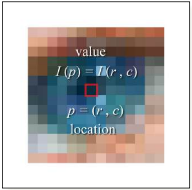
Pixel : $[\, p, I(p)]$

Pixels

Pixel : $[\,p,\ I(p)]$



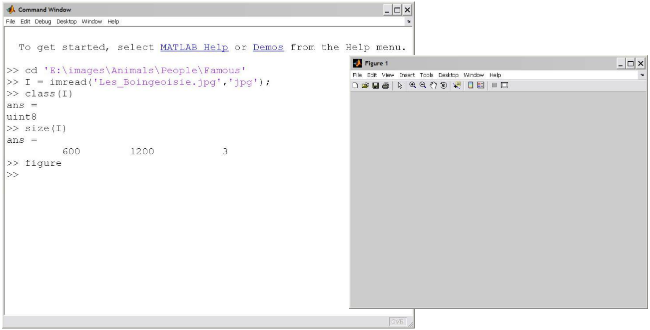




$$\begin{aligned} p &= (r, c) \\ &= (\text{row \#, col \#}) \\ &= (272,\ 277) \end{aligned}$$

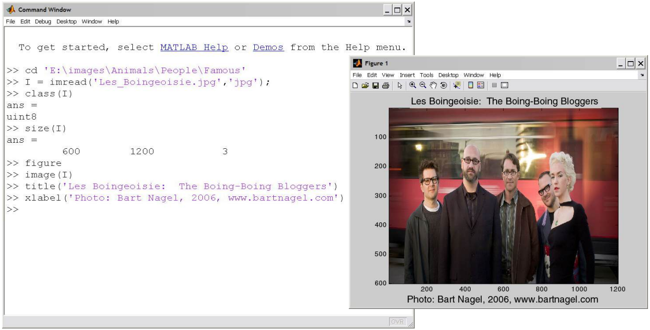
$$I(p) = \begin{bmatrix} \text{red} \\ \text{green} \\ \text{blue} \end{bmatrix} = \begin{bmatrix} 12 \\ 43 \\ 61 \end{bmatrix}$$

Read an Image into Matlab



```
  To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'E:\images\Animals\People\Famous'
>> I = imread('Les_Boingeoisie.jpg','jpg');
>> class(I)
ans =
uint8
>> size(I)
ans =
         600        1200           3
>> figure
>>
```

Read an Image into Matlab



```
To get started, select MATLAB Help or Demos from the Help menu.

>> cd 'E:\images\Animals\People\Famous'
>> I = imread('Les_Boingeoisie.jpg','jpg');
>> class(I)
ans =
uint8
>> size(I)
ans =
         600        1200           3
>> figure
>> image(I)
>> title('Les Boingeoisie:  The Boing-Boing Bloggers')
>> xlabel('Photo: Bart Nagel, 2006, www.bartnagel.com')
>>
```

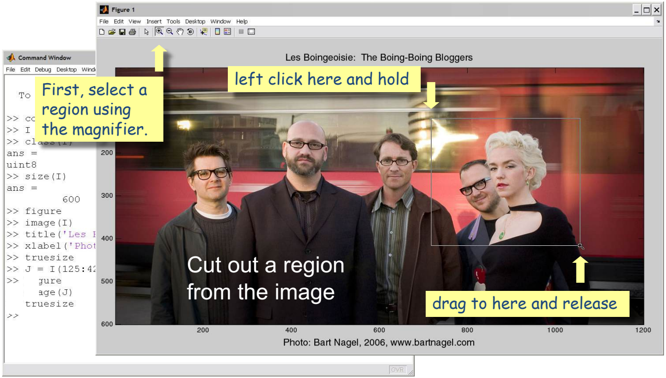
Les Boingeoisie: The Boing-Boing Bloggers

Photo: Bart Nagel, 2006, www.bartnagel.com

Read an Image into Matlab

```
  To get starte

>> cd 'E:\image
>> I = imread('
>> class(I)
ans =
uint8
>> size(I)
ans =
         600
>> figure
>> image(I)
>> title('Les F
>> xlabel('Phot
>> truesize
>> J = I(125:42
>>   gure
     age(J)
   truesize
>>
```

Crop the Image



```
  To

>> co
>> I
>> class(I)
ans =
uint8
>> size(I)
ans =
         600
>> figure
>> image(I)
>> title('Les I
>> xlabel('Phot
>> truesize
>> J = I(125:42
>>    gure
      age(J)
   truesize
>>
```

Les Boingeoisie: The Boing-Boing Bloggers

Photo: Bart Nagel, 2006, www.bartnagel.com

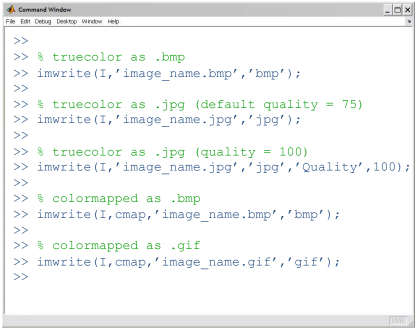
First, select a region using the magnifier.

left click here and hold

drag to here and release

Cut out a region from the image

Saving Images as Files



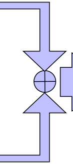
```
>>
>> % truecolor as .bmp
>> imwrite(I,'image_name.bmp','bmp');
>>
>> % truecolor as .jpg (default quality = 75)
>> imwrite(I,'image_name.jpg','jpg');
>>
>> % truecolor as .jpg (quality = 100)
>> imwrite(I,'image_name.jpg','jpg','Quality',100);
>>
>> % colormapped as .bmp
>> imwrite(I,cmap,'image_name.bmp','bmp');
>>
>> % colormapped as .gif
>> imwrite(I,cmap,'image_name.gif','gif');
>>
```

Assuming that
'I' contains the image of the correct class,
that
'cmap' is a colormap,
and that
'image_name' is the file-name that you want.

Double Exposure: Adding Two Images

Jim Woodring - Bumperillo

Mark Rayden – The Ecstasy of Cecelia



Rayden Woodring – The Ecstasy of Bumperillo (?)

Double Exposure: Adding Two Images

Example
Matlab Code

```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Double Exposure: Adding Two Images

Example Matlab Code

```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Cut a section out of the middle of the larger image the same size as the smaller image.

Double Exposure: Adding Two Images

Example Matlab Code

```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Note that the images are averaged, pixelwise.

Double Exposure: Adding Two Images

Example Matlab Code

```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> figure
>> image(JW)
>> truesize
>> title('Bumperillo')
>> xlabel('Jim Woodring')
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> figure
>> image(MR)
>> truesize
>> title('The Ecstasy of Cecelia')
>> xlabel('Mark Ryden')
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusM)
>> truesize
>> title('The Ecstasy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
```

Note the data class conversions.

Intensity Masking: Multiplying Two Images

Jim Woodring - Bumperillo

Mark Rayden – The Ecstasy of Cecelia

Rayden Woodring – Bumperillo Ecstasy (?)

Intensity Masking: Multiplying Two Images

Example
Matlab Code

```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Extacsy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
>> JWtimesMR = double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:)).*double(MR);
>> M = max(JWtimesMR(:));
>> m = min(JWtimesMR(:));
>> JWtimesMR = uint8(255*(double(JWtimesMR)-m)/(M-m));
>> figure
>> image(JWtimesMR)
>> truesize
>> title('EcstasyBumperillo')
```

Intensity Masking: Multiplying Two Images

Example Matlab Code

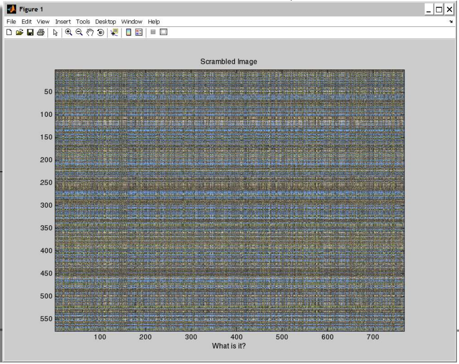
```
>> JW = imread('Jim Woodring - Bumperillo.jpg','jpg');
>> MR = imread('Mark Ryden - The Ecstasy of Cecelia.jpg','jpg');
>> [RMR,CMR,DMR] = size(MR);
>> [RJW,CJW,DJW] = size(JW);
>> rb = round((RJW-RMR)/2);
>> cb = round((CJW-CMR)/2);
>> JWplusMR = uint8((double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:))+double(MR))/2);
>> figure
>> image(JWplusMR)
>> truesize
>> title('The Extacsy of Bumperillo')
>> xlabel('Jim Woodring + Mark Ryden')
>> JWtimesMR = double(JW(rb:(rb+RMR-1),cb:(cb+CMR-1),:)).*double(MR);
>> M = max(JWtimesMR(:));
>> m = min(JWtimesMR(:));
>> JWtimesMR = uint8(255*(double(JWtimesMR)-m)/(M-m));
>> figure
>> image(JWtimesMR)
>> truesize
>> title('EcstasyBumperillo')
```

Note that the images are multiplied, pixelwise.

Note how the image intensities are scaled back into the range 0-255.

Pixel Indexing in Matlab

```
>> I = imread('Lawraa - Flickr - 278635073_883bd891ec_o.jpg','jpg');
>> size(I)
ans =
   576   768     3
>> r = randperm(576);
>> c = randperm(768);
>> J = I(r,c,:);
>> figure
>> image(J)
>> truesize
>> title('Scrambled Image')
>> xlabel('What is it?')
```



Point Processing of Images

- In a digital image, point = pixel.
- Point processing transforms a pixel’s value as function of its value alone;
- it does not depend on the values of the pixel’s neighbors.

Point Processing of Images

- Brightness and contrast adjustment
- Gamma correction
- Histogram equalization
- Histogram matching
- Color correction.

Point Processing

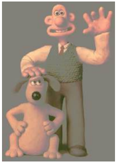

- gamma


- brightness


original


+ brightness


+ gamma


histogram mod


- contrast


original


+ contrast


histogram EQ

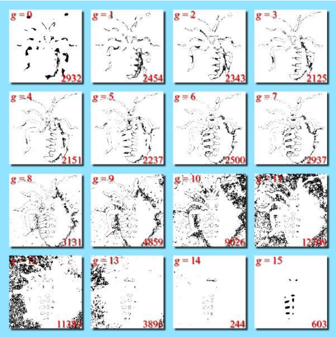
The Histogram of a Grayscale Image

- Let **I** be a 1-band (grayscale) image.
- **I**(r,c) is an 8-bit integer between 0 and 255.
- Histogram, $h_{\mathbf{I}}$, of **I**:
  - a 256-element array, $h_{\mathbf{I}}$
  - $h_{\mathbf{I}}(g)$, for $g = 1, 2, 3, \ldots, 256$, is an integer
  - $h_{\mathbf{I}}(g)$ = number of pixels in **I** that have value g-1.

The Histogram of a Grayscale Image

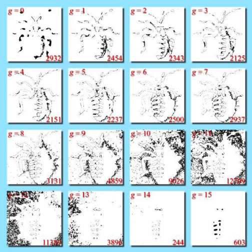
16-level (4-bit) image

lower RHC: number of pixels with intensity g



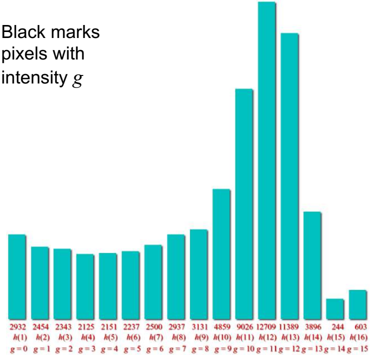
black marks pixels with intensity g

The Histogram of a Grayscale Image

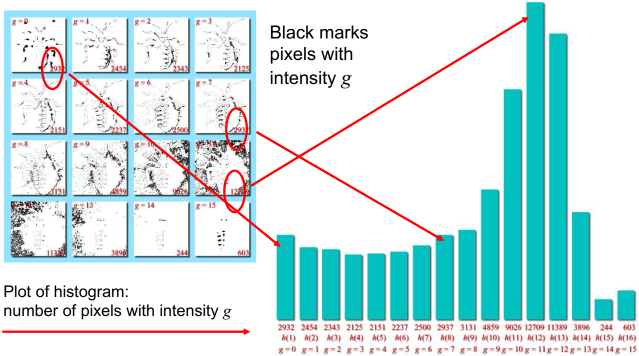


Black marks pixels with intensity g

Plot of histogram:
number of pixels with intensity g



The Histogram of a Grayscale Image

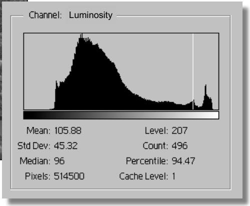


Black marks pixels with intensity g

Plot of histogram: number of pixels with intensity g

The Histogram of a Grayscale Image

$h_{\mathbf{I}}(g+1) =$ the number of pixels in **I** with graylevel g.



Channel: Luminosity

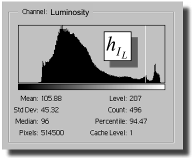
Mean: 105.88
Level: 207
Std Dev: 45.32
Count: 496
Median: 96
Percentile: 94.47
Pixels: 514500
Cache Level: 1

The Histogram of a Color Image

- If $\mathbf{I}$ is a 3-band image (truecolor, 24-bit)
- then $\mathbf{I}(r,c,b)$ is an integer between 0 and 255.
- Either $\mathbf{I}$ has 3 histograms:
  - $h_{\mathbf{R}}(g+1)$ = # of pixels in $\mathbf{I}(:,:,1)$ with intensity value g
  - $h_{\mathbf{G}}(g+1)$ = # of pixels in $\mathbf{I}(:,:,2)$ with intensity value g
  - $h_{\mathbf{B}}(g+1)$ = # of pixels in $\mathbf{I}(:,:,3)$ with intensity value g
- or 1 vector-valued histogram, $h(g,1,b)$ where
  - $h(g+1,1,1)$ = # of pixels in $\mathbf{I}$ with red intensity value g
  - $h(g+1,1,2)$ = # of pixels in $\mathbf{I}$ with green intensity value g
  - $h(g+1,1,3)$ = # of pixels in $\mathbf{I}$ with blue intensity value g

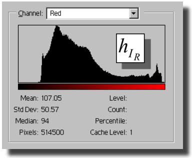
The Histogram of a Color Image

There is one histogram per color band R, G, & B. Value histogram is from 1 band = (R+G+B)/3



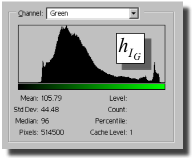
Channel: Luminosity

Mean: 105.88 Level: 207
Std Dev: 45.32 Count: 496
Median: 96 Percentile: 94.47
Pixels: 514500 Cache Level: 1



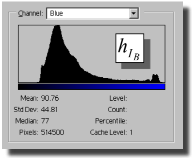
Channel: Red

Mean: 107.05 Level:
Std Dev: 50.57 Count:
Median: 94 Percentile:
Pixels: 514500 Cache Level: 1



Channel: Green

Mean: 105.79 Level:
Std Dev: 44.48 Count:
Median: 96 Percentile:
Pixels: 514500 Cache Level: 1



Channel: Blue

Mean: 90.76 Level:
Std Dev: 44.81 Count:
Median: 77 Percentile:
Pixels: 514500 Cache Level: 1

Value or Luminance Histograms

The Value histogram of a 3-band (truecolor) image, **I**, is the histogram of the value image,

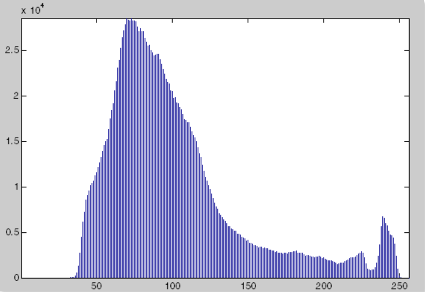
$$\mathbf{V}(r,c)=\frac{1}{3}[\mathbf{R}(r,c)+\mathbf{G}(r,c)+\mathbf{B}(r,c)]$$

Where **R**, **G**, and **B** are the red, green, and blue bands of **I**.
The luminance histogram of **I** is the histogram of the luminance image,

$$\mathbf{L}(r,c)=0.299\cdot\mathbf{R}(r,c)+0.587\cdot\mathbf{G}(r,c)+0.114\cdot\mathbf{B}(r,c)$$

Value Histogram

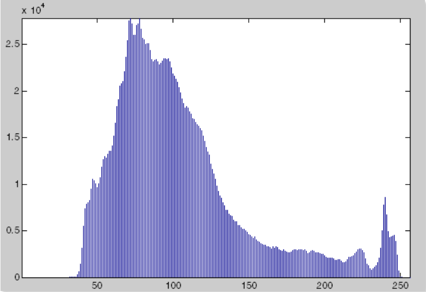
Value image, **V**.



Histogram of the value image.

Luminance Histogram

Luminance image, **L**.



Histogram of the luminance image.

Multi-Band Histogram Calculator in Matlab

```
% Multi-band histogram calculator
function h=histogram(I)

[R C B]=size(I);

% allocate the histogram
h=zeros(256,1,B);

% range through the intensity values
for g=0:255
   h(g+1,1,:) = sum(sum(I==g)); % accumulate
end

return;
```

Multi-Band Histogram Calculator in Matlab

```
% Multi-band histogram calc
function h=histogram(I)

[R C B]=size(I);

% allocate the histogram
h=zeros(256,1,B);

% range through the intensi
for g=0:255
   h(g+1,1,:) = sum(sum(I==g)); % accumulate
end
```

Loop through all intensity levels (0-255)
Tag the elements that have value g.
The result is an RxCxB *logical* array that has a 1 wherever $\mathbf{I}(r,c,b) = g$ and 0's everywhere else.
Compute the number of ones in each band of the image for intensity g.
Store that value in the 256x1xB histogram at $h(g+1,1,b)$.

If `B==3`, then `h(g+1,1,:)` contains 3 numbers: the number of pixels in bands 1, 2, & 3 that have intensity `g`.

`sum(sum(I==g))` computes one number for each band in the image.

Point Ops via Functional Mappings

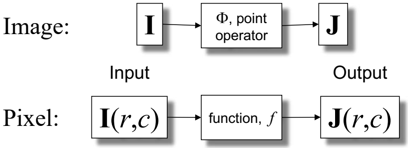


Image: $\mathbf{I} \rightarrow$ Φ, point operator $\rightarrow \mathbf{J}$

Input Output

Pixel: $\mathbf{I}(r,c) \rightarrow$ function, f $\rightarrow \mathbf{J}(r,c)$

$$\mathbf{J} = \mathsf{F}\left[\mathbf{I}\right]$$

If $\mathbf{I}(r,c) = g$

and $f(g) = k$

then $\mathbf{J}(r,c) = k$.

The transformation of image **I** into image **J** is accomplished by replacing each input intensity, *g*, with a specific output intensity, *k*, at every location (*r*,*c*) where **I**(*r*,*c*) = *g*.

The rule that associates *k* with *g* is usually specified with a function, *f*, so that *f* (*g*) = *k*.

Point Ops via Functional Mappings

One-band Image

$$\mathbf{J}(r,c) = f\left(\,\mathbf{I}(r,c)\,\right),$$

for all pixels locations (r,c).

Three-band Image

$$\mathbf{J}(r,c,b) = f\left(\mathbf{I}(r,c,b)\right), \text{ or}$$

$$\mathbf{J}(r,c,b) = f_b\left(\,\mathbf{I}(r,c,b)\,\right),$$

for $b = 1,2,3$ and all (r,c).

Point Ops via Functional Mappings

One-band Image

$$\mathbf{J}(r,c)=f\left(\,\mathbf{I}(r,c)\,\right),$$

for all pixels locations (r,c).

Either all 3 bands are mapped through the same function, f, or …

Three-band Image

$$\mathbf{J}(r,c,b)=f\left(\mathbf{I}(r,c,b)\right),\ \text{or}$$

$$\mathbf{J}(r,c,b)=f_b\left(\,\mathbf{I}(r,c,b)\,\right),$$

for $b=1,2,3$ and all $(r,c$

… each band is mapped through a separate function, f_b.

Point Operations using Look-up Tables

A look-up table (LUT) implements a functional mapping.

If $k = f(g)$,

for $g = 0, \ldots, 255$,

and if k takes on

values in $\{0, \ldots, 255\}$, …

… then the LUT that implements f is a 256x1 array whose $(g+1)^{\text{th}}$ value is $k = f(g)$.

To remap a 1-band image, $\mathbf{I}$, to $\mathbf{J}$:

$$\mathbf{J} = \mathrm{LUT}(\mathbf{I} + 1)$$

Point Operations using Look-up Tables

If **I** is 3-band, then

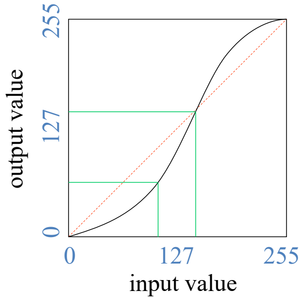
a) each band is mapped separately using the same LUT for each band *or*

b) each band is mapped using different LUTs – one for each band.

a) $\mathbf{J} = \mathrm{LUT}(\mathbf{I}+1)$, *or*

b) $\mathbf{J}(:,:,b) = \mathrm{LUT}_b(\mathbf{I}(:,:,b)+1)$ for $b = 1,2,3.$

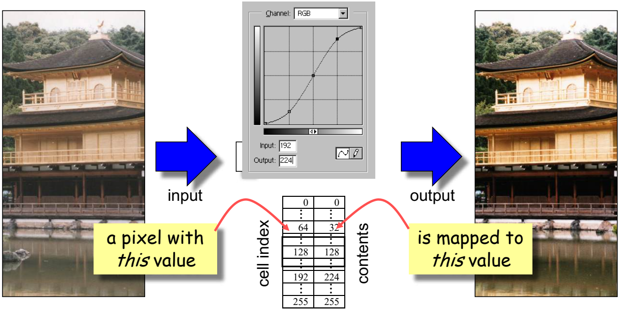
Point Operations = Look-up Table Ops



E.g.:	index	value
	...	...
	101	64
	102	68
	103	69
	104	70
	105	70
	106	71
	...	...

input output

Look-Up Tables



Channel: RGB

Input: 192

Output: 224

a pixel with *this* value

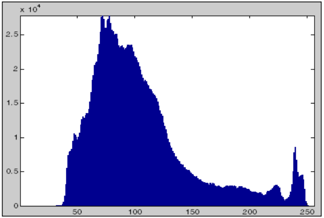
is mapped to *this* value

0	0
⋮	⋮
64	32
⋮	⋮
128	128
⋮	⋮
192	224
⋮	⋮
255	255

Point Processes: Original Image

Kinkaku-ji (金閣寺 Temple of the Golden Pavilion), also known as Rokuon-ji (鹿苑寺 Deer Garden Temple), is a Zen Buddhist temple in Kyoto, Japan.

Photo by Richard Alan Peters II, August 1993.

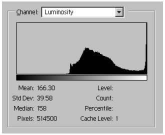


Luminance Histogram

Point Processes: Increase Brightness

$$\mathbf{J}(r,c,b)=\begin{cases}\mathbf{I}(r,c,b)+g, & \text{if } \mathbf{I}(r,c,b)+g<256\\ 255, & \text{if } \mathbf{I}(r,c,b)+g>255\end{cases}$$

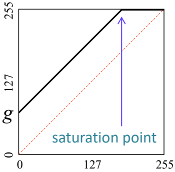
$g \geq 0$ and $b \in \{1,2,3\}$ is the band index.



Channel: Luminosity

Mean: 166.30
Std Dev: 39.58
Median: 158
Pixels: 514500

Level:
Count:
Percentile:
Cache Level: 1

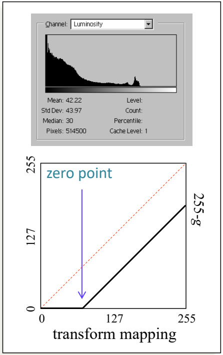


transform mapping

Point Processes: Decrease Brightness

$$\mathbf{J}(r,c,b) = \begin{cases} 0, & \text{if } \mathbf{I}(r,c,b) - g < 0 \\ \mathbf{I}(r,c,b) - g, & \text{if } \mathbf{I}(r,c,b) - g > 0 \end{cases}$$

$g \geq 0$ and $b \in \{1,2,3\}$ is the band index.



Channel: Luminosity

Mean: 42.22

Std Dev: 43.97

Median: 30

Pixels: 514500

Level:

Count:

Percentile:

Cache Level: 1

Point Processes: Decrease Contrast

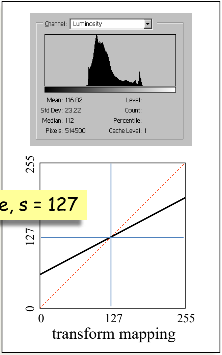
$$\mathbf{T}(r,c,b) = a\left[\mathbf{I}(r,c,b) - s\right] + s,$$

where $0 \le a < 1.0$,

$s \in \{0,1,2,\ldots,255\}$, and

$b \in \{1,2,3\}$.

s is the center of the contrast function.



Channel: Luminosity

Mean: 116.82
Std Dev: 23.22
Median: 112
Pixels: 514500
Level:
Count:
Percentile:
Cache Level: 1

transform mapping

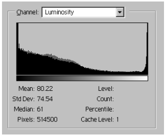
Here, *s* = 127

Point Processes: Increase Contrast

$$\mathbf{T}(r,c,b) = a[\mathbf{I}(r,c,b) - s] + s$$

$$\mathbf{J}(r,c,b) = \begin{cases} 0, & \text{if } \mathbf{T}(r,c,b) < 0, \\ \mathbf{T}(r,c,b), & \text{if } 0 \le \mathbf{T}(r,c,b) \le 255, \\ 255, & \text{if } \mathbf{T}(r,c,b) > 255. \end{cases}$$

$$a > 1,\quad s \in \{0,\ldots,255\},\quad b \in \{1,2,3\}$$



Channel: Luminosity

Mean: 80.22

Std Dev: 74.54

Median: 61

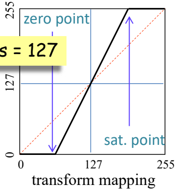
Pixels: 514500

Level:

Count:

Percentile:

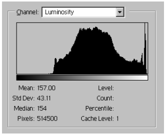
Cache Level: 1



Here, s = 127

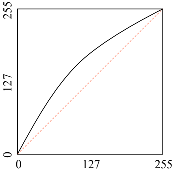
Point Processes: Increased Gamma

$$\mathbf{J}(r,c) = 255 \cdot \left[\frac{\mathbf{I}(r,c)}{255}\right]^{\frac{1}{\gamma}} \quad \text{for} \quad \gamma > 1.0$$



Mean: 157.00
Std Dev: 43.11
Median: 154
Pixels: 514500

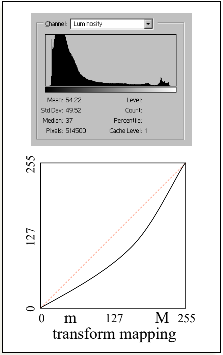
Level:
Count:
Percentile:
Cache Level: 1



transform mapping

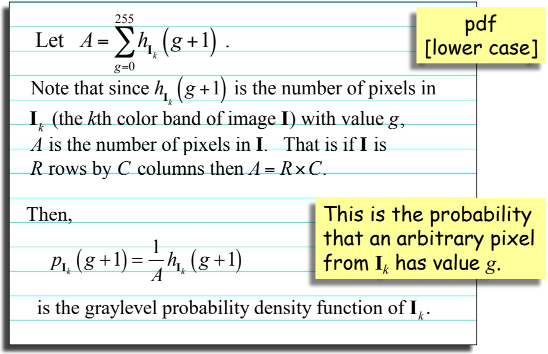
Point Processes: Decreased Gamma

$$\mathbf{J}(r,c) = 255\cdot\left[\frac{\mathbf{I}(r,c)}{255}\right]^{\frac{1}{\gamma}} \quad \text{for} \quad \gamma < 1.0$$



Mean: 54.22
Std Dev: 49.52
Median: 37
Pixels: 514500
Level:
Count:
Percentile:
Cache Level: 1

The Probability Density Function of an Image



Let $A = \sum_{g=0}^{255} h_{\mathbf{I}_k}(g+1)$.

pdf
[lower case]

Note that since $h_{\mathbf{I}_k}(g+1)$ is the number of pixels in $\mathbf{I}_k$ (the kth color band of image $\mathbf{I}$) with value g, A is the number of pixels in $\mathbf{I}$. That is if $\mathbf{I}$ is R rows by C columns then $A = R \times C$.

Then,

$$p_{\mathbf{I}_k}(g+1) = \frac{1}{A} h_{\mathbf{I}_k}(g+1)$$

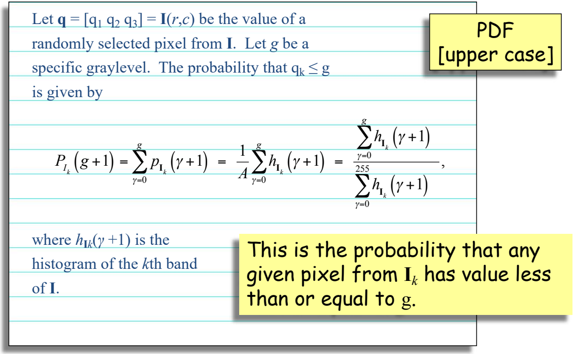
This is the probability that an arbitrary pixel from $\mathbf{I}_k$ has value g.

is the graylevel probability density function of $\mathbf{I}_k$.

The Probability Density Function of an Image

- $p_{\text{band}}(g+1)$ is the fraction of pixels in (a specific band of) an image that have intensity value g.
- $p_{\text{band}}(g+1)$ is the probability that a pixel randomly selected from the given band has intensity value g.
- Whereas the sum of the histogram $h_{\text{band}}(g+1)$ over all g from 1 to 256 is equal to the number of pixels in the image, the sum of $p_{\text{band}}(g+1)$ over all g is 1.
- p_{band} is the normalized histogram of the band.

The Probability Distribution Function of an Image



Let $\mathbf{q} = [q_1\ q_2\ q_3] = \mathbf{I}(r,c)$ be the value of a randomly selected pixel from **I**. Let g be a specific graylevel. The probability that $q_k \le g$ is given by

$$P_{I_k}\left(g+1\right) = \sum_{\gamma=0}^{g} p_{\mathbf{I}_k}\left(\gamma+1\right) = \frac{1}{A}\sum_{\gamma=0}^{g} h_{\mathbf{I}_k}\left(\gamma+1\right) = \frac{\sum_{\gamma=0}^{g} h_{\mathbf{I}_k}\left(\gamma+1\right)}{\sum_{\gamma=0}^{255} h_{\mathbf{I}_k}\left(\gamma+1\right)},$$

where $h_{\mathbf{I}k}(\gamma+1)$ is the histogram of the kth band of **I**.

PDF
[upper case]

This is the probability that any given pixel from $\mathbf{I}_k$ has value less than or equal to g.

Point Processes: Histogram Equalization

Task: remap image **I** so that its histogram is as close to constant as possible

Let $P_{\mathbf{I}}(\mathrm{g}+1)$ be the probability distribution function of **I**.

Then **J** has, as closely as possible, the correct histogram if

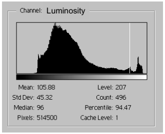
$$\mathbf{J}(r,c,b) = 255 \cdot P_{\mathbf{I}}\left[\mathbf{I}(r,c,b)+1\right].$$

The PDF itself is used as the LUT.

all bands processed similarly

Point Processes: Histogram Equalization

$$\mathbf{J}(r,c,b) = 255 \cdot P_{\mathbf{I}}(g+1),$$
$$g = \mathbf{I}(r,c,b),\ \ \mathrm{b} \in \{1,2,3\}.$$

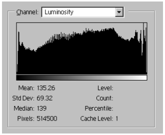


Channel: Luminosity

Mean: 105.88
Std Dev: 45.32
Median: 96
Pixels: 514500

Level: 207
Count: 496
Percentile: 94.47
Cache Level: 1

before



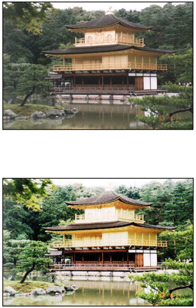
Channel: Luminosity

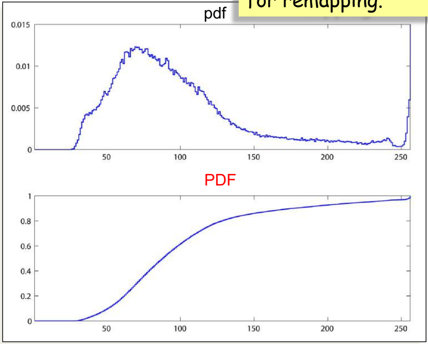
Mean: 135.26
Std Dev: 69.32
Median: 139
Pixels: 514500

Level:
Count:
Percentile:
Cache Level: 1

after

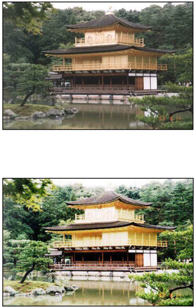
Histogram EQ

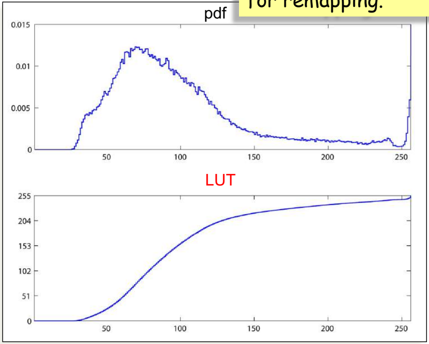




The PDF is the LUT for remapping.

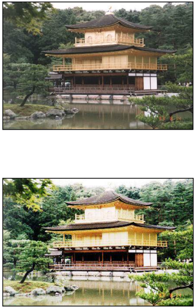
Histogram EQ

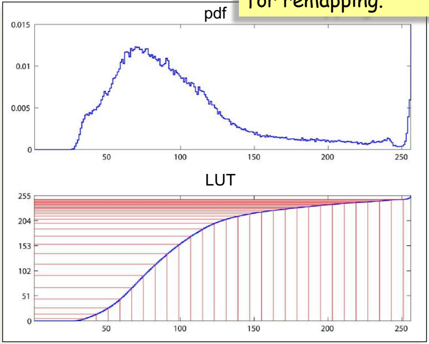




The PDF is the LUT for remapping.

Histogram EQ





The PDF is the LUT for remapping.

Point Processes: Histogram Equalization

Task: remap image **I** with min = $m_{\mathbf{I}}$ and max = $M_{\mathbf{I}}$ so that its histogram is as close to constant as possible and has min = $m_{\mathbf{J}}$ and max = $M_{\mathbf{J}}$.

Let $P_{\mathbf{I}}(\mathrm{g}+1)$ be the probability distribution function of **I**.

Then **J** has, as closely as possible, the correct histogram if

Using intensity extrema

$$\mathbf{J}(r,c)=(M_{\mathbf{J}}-m_{\mathbf{J}})\frac{P_{\mathbf{I}}[\mathbf{I}(r,c)+1]-P_{\mathbf{I}}(m_{\mathbf{I}}+1)}{P_{\mathbf{I}}(M_{\mathbf{I}}+1)-P_{\mathbf{I}}(m_{\mathbf{I}}+1)}+m_{\mathbf{J}}.$$

Point Processes: Histogram Matching

Task: remap image **I** so that it has, as closely as possible, the same histogram as image **J**.

Because the images are digital it is not, in general, possible to make $h_{\mathbf{I}} \equiv h_{\mathbf{J}}$. Therefore, $p_{\mathbf{I}} \not\equiv p_{\mathbf{J}}$.

Q: How, then, can the matching be done?
A: By matching percentiles.

Matching Percentiles

… assuming a 1-band image or a single band of a color image.

Recall:

- The PDF of image **I** is such that $0 \le P_{\mathbf{I}}(g_{\mathbf{I}}) \le 1$.
- $P_{\mathbf{I}}(g_{\mathbf{I}}+1) = c$ means that c is the fraction of pixels in **I** that have a value less than or equal to $g_{\mathbf{I}}$.
- $100c$ is the *percentile* of pixels in **I** that are less than or equal to $g_{\mathbf{I}}$.

To match percentiles, replace all occurrences of value $g_{\mathbf{I}}$ in image **I** with the value, $g_{\mathbf{J}}$, from image **J** whose percentile in **J** most closely matches the percentile of $g_{\mathbf{I}}$ in image **I**.

Matching Percentiles

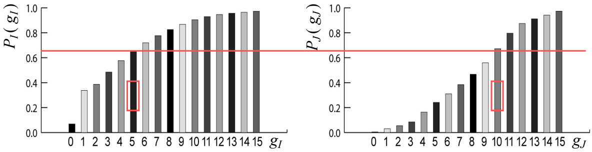
… assuming a 1-band image or a single band of a color image.

So, to create an image, **K**, from image **I** such that **K** has nearly the same PDF as image **J** do the following:

If $\mathbf{I}(r,c) = g_\mathbf{I}$ then let $K(r,c) = g_\mathbf{J}$ where $g_\mathbf{J}$ is such that

$P_\mathbf{I}(g_\mathbf{I}) > P_\mathbf{J}(g_\mathbf{J} - 1)$ AND $P_\mathbf{I}(g_\mathbf{I}) \leq P_\mathbf{J}(g_\mathbf{J})$.

Example:
$\mathbf{I}(r,c) = 5$
$P_\mathbf{I}(5) = 0.65$
$P_\mathbf{J}(9) = 0.56$
$P_\mathbf{J}(10) = 0.67$
$\mathbf{K}(r,c) = 10$



Histogram Matching Algorithm

Assuming a 1-band image or a single band of a color image.

```
$[R,C]$ = size($\mathbf{I}$);
$\mathbf{K}$ = zeros($R$,$C$);
$g_\mathbf{J} = m_\mathbf{J}$;
for $g_\mathbf{I} = m_\mathbf{I}$ to $M_\mathbf{I}$
    while $g_\mathbf{J} < 255$ AND $P_\mathbf{I}(g_\mathbf{I}+1) < 1$ AND
              $P_\mathbf{J}(g_\mathbf{J}+1) < P_\mathbf{I}(g_\mathbf{I}+1)$
        $g_\mathbf{J} = g_\mathbf{J} + 1$;
    end
    $\mathbf{K} = \mathbf{K} + [g_\mathbf{J} \cdot (\mathbf{I} == g_\mathbf{I})]$
end
```

This directly matches image **I** to image **J**.

$P_\mathbf{I}(g_\mathbf{I}+1)$: PDF of **I**

$P_\mathbf{J}(g_\mathbf{J}+1)$: PDF of **J**.

$m_\mathbf{J} =$ min **J**,

$M_\mathbf{J} =$ max **J**,

$m_\mathbf{I} =$ min **I**,

$M_\mathbf{I} =$ max **I**.

Better to use a LUT.

Example: Histogram Matching

Image pdf

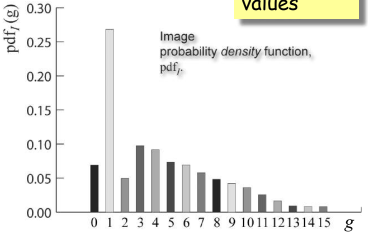


Image with 16 intensity values

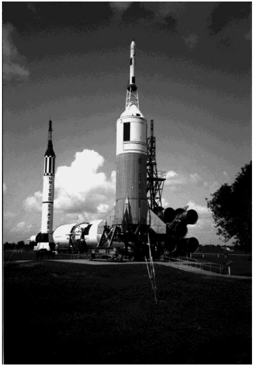
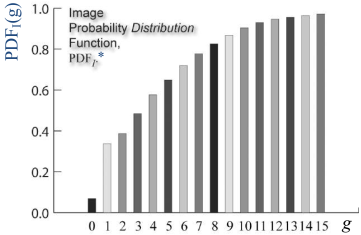
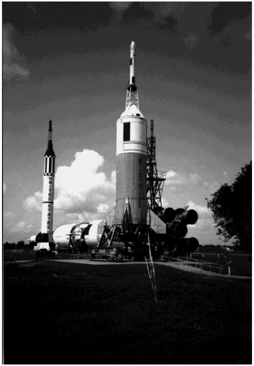


Image PDF

Example: Histogram Matching

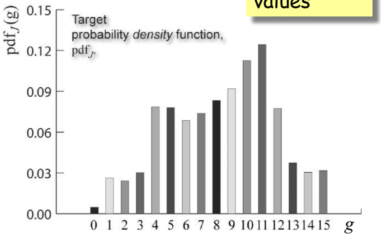


*a.k.a Cumulative Distribution Function, CDF_I.

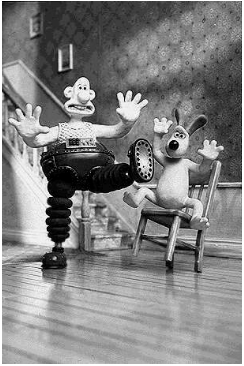


Example: Histogram Matching

Target pdf

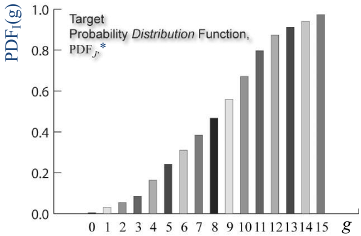


Target with 16 intensity values

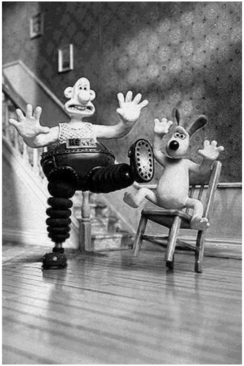


Target PDF

Example: Histogram Matching



*a.k.a Cumulative Distribution Function, CDF$_J$.



Histogram Matching with a Lookup Table

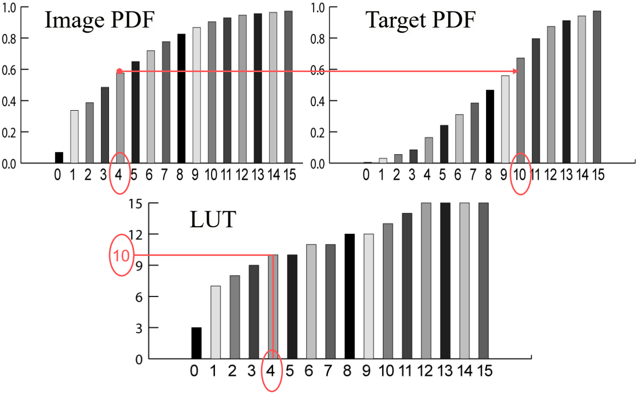
Often it is faster or more versatile to use a lookup table (LUT). Rather than remapping each pixel in the image separately, one can create a table that indicates to which target value each input value should be mapped. Then

$$\mathbf{K} = \mathrm{LUT}[\mathbf{I}+1]$$

In *Matlab* if the LUT is a 256 × 1 matrix with values from 0 to 255 and if image **I** is of type **`uint8`**, it can be remapped with the following code:

```
K = uint8(LUT(I+1));
```

LUT Creation



Look Up Table for Histogram Matching

```
$\mathrm{LUT} =$ zeros(256,1);
$g_{\mathbf{J}}$ = 0;
for $g_{\mathbf{I}}$ = 0 to 255
    while $P_{\mathbf{J}}(g_{\mathbf{J}}+1) < P_{\mathbf{I}}(g_{\mathbf{I}}+1)$ AND $g_{\mathbf{J}} < 255$
        $g_{\mathbf{J}} = g_{\mathbf{J}} + 1;$
    end
    $\mathrm{LUT}(g_{\mathbf{I}}+1) = g_{\mathbf{J}};$
end
```

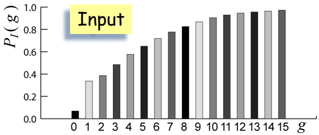
This creates a look-up table which can then be used to remap the image.

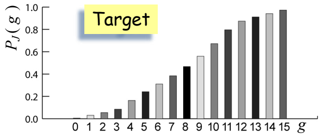
$P_{\mathbf{I}}(g_{\mathbf{I}}+1)$: PDF of **I**,

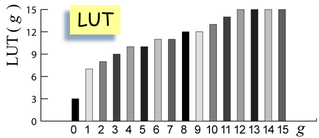
$P_{\mathbf{J}}(g_{\mathbf{J}}+1)$: PDF of **J**,

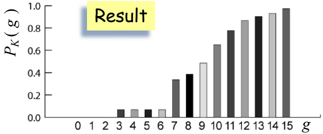
$\mathrm{LUT}(g_{\mathbf{I}}+1)$: Look- Up Table

Input & Target PDFs, LUT and Resultant PDF

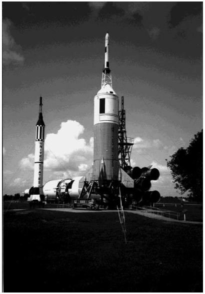








Example: Histogram Matching



original

target

remapped

中国科学技术大学
University of Science and Technology of China

Image Convolution

Juyong Zhang
School of Mathematics, USTC

Spatial Filtering

Let **I** and **J** be images such that **J** = T[**I**].

T[·] represents a transformation, such that,

$$\mathbf{J}(r,c) = \mathrm{T}\left[\mathbf{I}\right](r,c) =$$

$$f\left(\left\{\mathbf{I}(\rho,\chi) \,\middle|\, \rho \in \left\{r-s,\ldots,r,\ldots r+s\right\}, \chi \in \left\{c-d,\ldots,c,\ldots c+d\right\}\right\}\right).$$

That is, the value of the transformed image, **J**, at pixel location (*r,c*) is a function of the values of the original image, **I**, in a 2*s*+1 × 2*d* +1 rectangular neighborhood centered on pixel location (*r,c*).

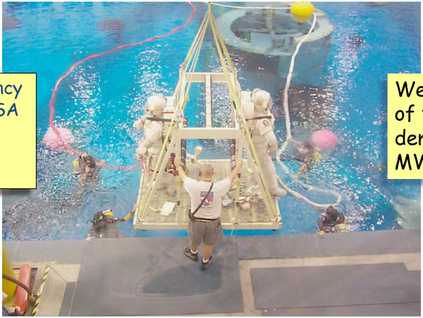
Moving Windows

- The value, J(r,c) = T[I](r,c), is a function of a rectangular neighborhood centered on pixel location (r,c) in I.

- There is a different neighborhood for each pixel location, but if the dimensions of the neighbor-hood are the same for each location, then transform T is sometimes called a moving window transform.

Moving-Window Transformations

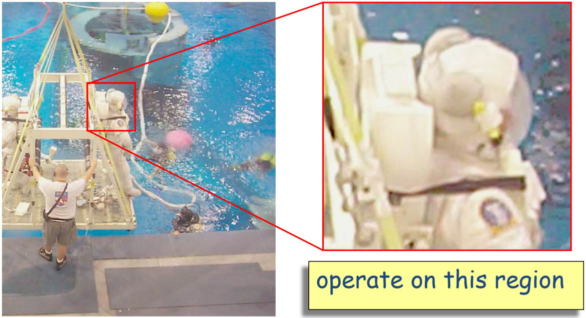
Neutral Buoyancy Facility at NASA Johnson Space Center



We'll take a section of this image to demonstrate the MWT

photo: R.A.Peters II, 1999

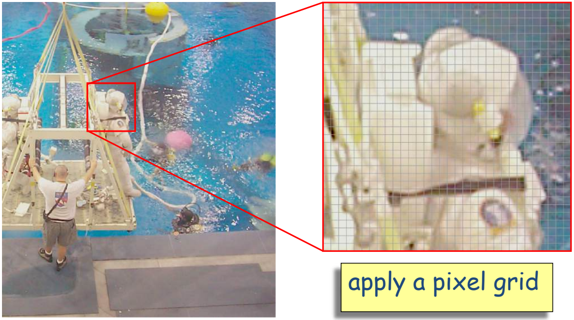
Moving-Window Transformations



operate on this region

Moving-Window Transformations

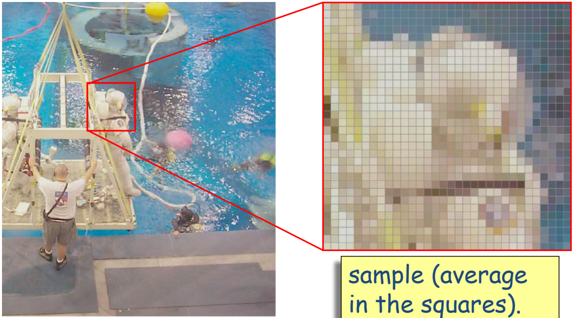
Pixelize the section to better see the effects.



apply a pixel grid

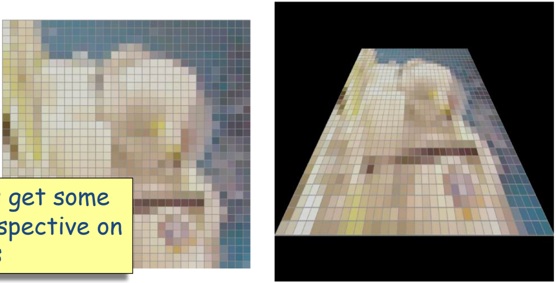
Moving-Window Transformations

Pixelize the section to better see the effects.



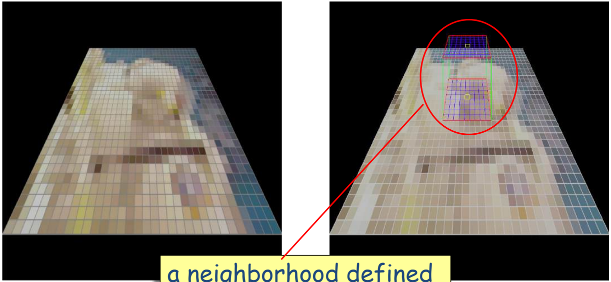
sample (average in the squares).

Moving-Window Transformations



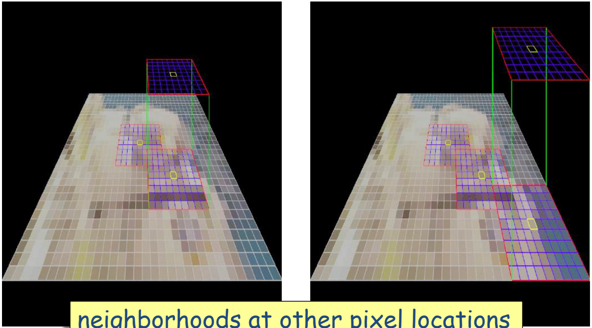
lets get some perspective on this

Moving-Window Transformations



a neighborhood defined by a weight matrix

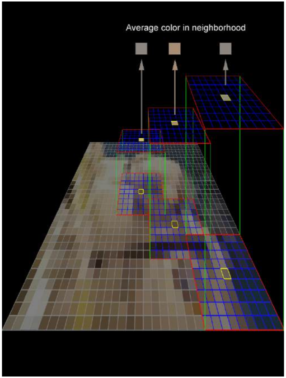
Moving-Window Transformations



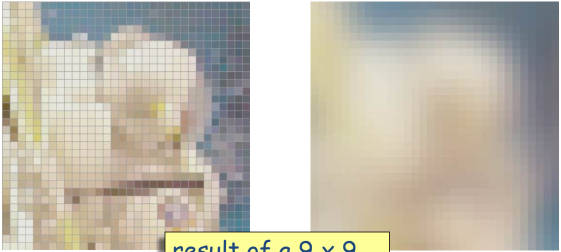
neighborhoods at other pixel locations

Linear Moving-Window Transformations
(*i.e.* convolution)

The output of the transform at each pixel is the (weighted) average of the pixels in the neighborhood.



Moving-Window Transformations



result of a 9 x 9 uniform averaging

Convolution: Mathematical Representation

If a MW transformation is *linear* then it is a *convolution*:

$$\mathbf{J}(r,c) = \left[\mathbf{I} * \mathbf{h}\right](r,c) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathbf{I}(r-\rho, c-\chi)\mathbf{h}(\rho,\chi)\,d\rho\,d\chi\,,$$

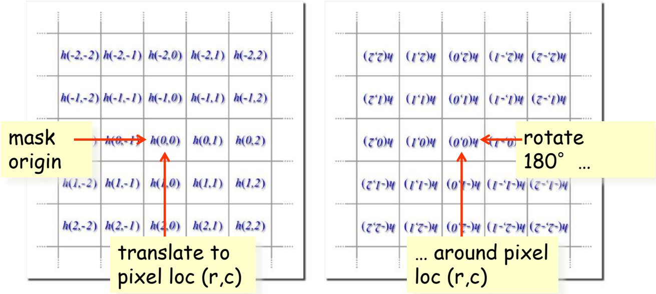
for a real image ($\mathbf{I}: R \times R \rightarrow R$), or for a digital image ($\mathbf{I}: Z \times Z \rightarrow Z$):

$$\mathbf{J}(r,c) = \left[\mathbf{I} * \mathbf{h}\right](r,c) = \sum_{\rho=-s}^{s}\sum_{\chi=-d}^{d} \mathbf{I}(r-\rho, c-\chi)\mathbf{h}(\rho,\chi)$$

Convolution Mask (Weight Matrix)

- The object, $\mathbf{h}(\rho, \chi)$, in the equation is a weighting function, or in the discrete case, a rectangular matrix of numbers.
- The matrix is the moving window.
- Pixel (r,c) in the output image is the weighted sum of pixels from the original image in the neighborhood of (r,c) traced by the matrix.
- Each pixel in the neighborhood of (r,c) is multiplied by the corresponding matrix value — after the matrix is rotated by 180º.
- The sum of those products is the value of pixel (r,c) in the output image

Convolution Masks: Moving Window



$h(-2,-2)$	$h(-2,-1)$	$h(-2,0)$	$h(-2,1)$	$h(-2,2)$
$h(-1,-2)$	$h(-1,-1)$	$h(-1,0)$	$h(-1,1)$	$h(-1,2)$
)	$h(0,-1)$	$h(0,0)$	$h(0,1)$	$h(0,2)$
$h(1,-2)$	$h(1,-1)$	$h(1,0)$	$h(1,1)$	$h(1,2)$
$h(2,-2)$	$h(2,-1)$	$h(2,0)$	$h(2,1)$	$h(2,2)$

mask origin

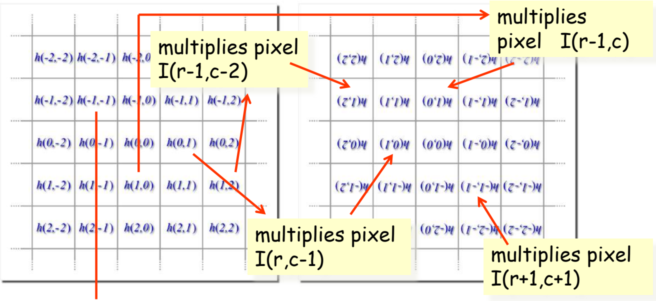
translate to pixel loc (r,c)

$h(2,2)$	$h(2,1)$	$h(2,0)$	$h(2,-1)$	$h(2,-2)$
$h(1,2)$	$h(1,1)$	$h(1,0)$	$h(1,-1)$	$h(1,-2)$
$h(0,2)$	$h(0,1)$	$h(0,0)$	$h(0,-1)$	
$h(-1,2)$	$h(-1,1)$	$h(-1,0)$	$h(-1,-1)$	$h(-1,-2)$
$h(-2,2)$	$h(-2,1)$	$h(-2,0)$	$h(-2,-1)$	$h(-2,-2)$

rotate 180° …

… around pixel loc (r,c)

Convolution Masks: Moving Window



h(-2,-2)	h(-2,-1)	h(-2,0		
h(-1,-2)	h(-1,-1)	h(-1,0)	h(-1,1)	h(-1,2)
h(0,-2)	h(0,-1)	h(0,0)	h(0,1)	h(0,2)
h(1,-2)	h(1,-1)	h(1,0)	h(1,1)	h(1,2)
h(2,-2)	h(2,-1)	h(2,0)	h(2,1)	h(2,2)

h(2,2)	h(2,1)	h(2,0)	h(2,-1)	h(2,-2)
h(1,2)	h(1,1)	h(1,0)	h(1,-1)	h(1,-2)
h(0,2)	h(0,1)	h(0,0)	h(0,-1)	h(0,-2)
h(-1,2)	h(-1,1)	h(-1,0)	h(-1,-1)	h(-1,-2)
		h(-2,0)	h(-2,-1)	h(-2,-2)

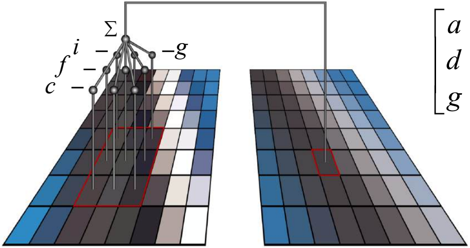
multiplies pixel I(r-1,c)

multiplies pixel I(r-1,c-2)

multiplies pixel I(r,c-1)

multiplies pixel I(r+1,c+1)

Convolution by Moving Window



$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

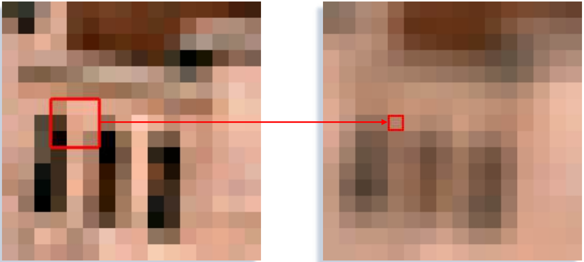
Moving Window Transform: Example

Another example

original

3x3 average

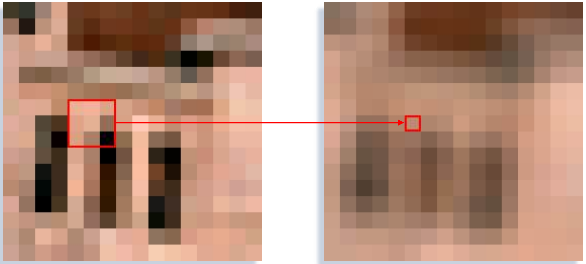
Moving Window Transform: Example



original

3x3 average

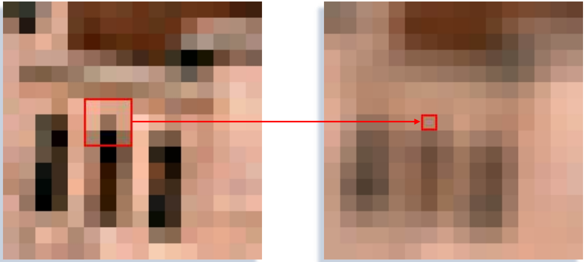
Moving Window Transform: Example



original

3x3 average

Moving Window Transform: Example



original

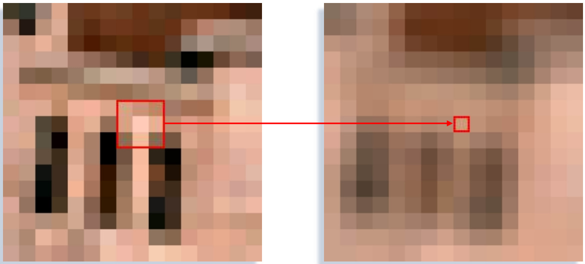
3x3 average

Moving Window Transform: Example

original

3x3 average

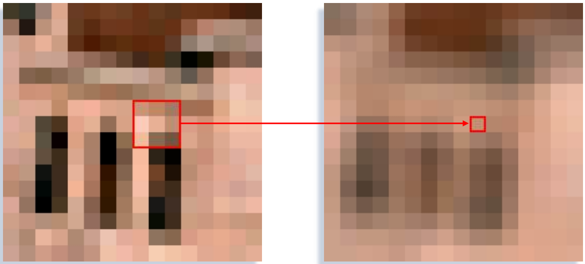
Moving Window Transform: Example



original

3x3 average

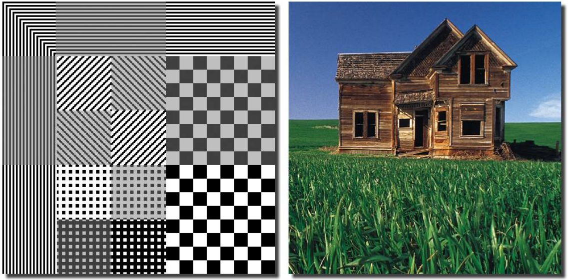
Moving Window Transform: Example



original

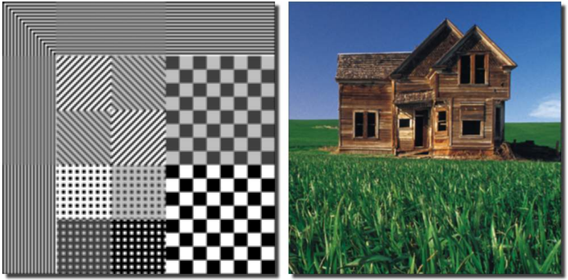
3x3 average

Convolution Examples: Original Images



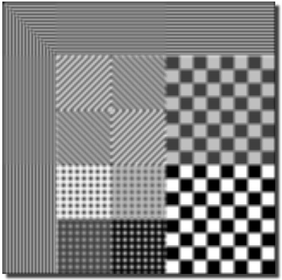
Convolution Examples: 3 × 3 Blur

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$



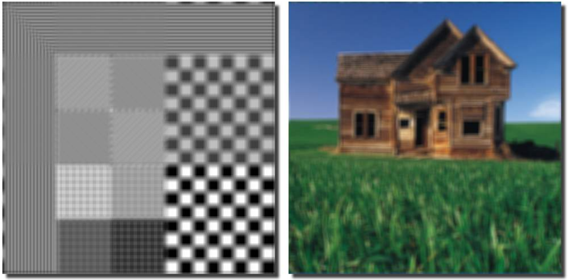
Convolution Examples: 5 × 5 Blur

$$\frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



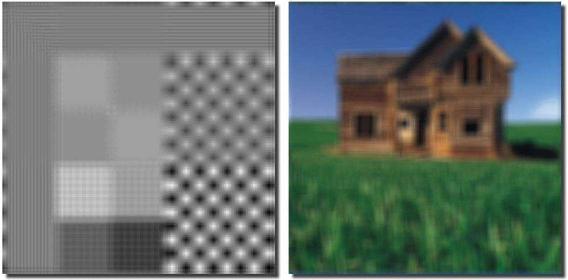
Convolution Examples: 9 × 9 Blur

$$\frac{1}{81}\begin{bmatrix}1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\end{bmatrix}$$

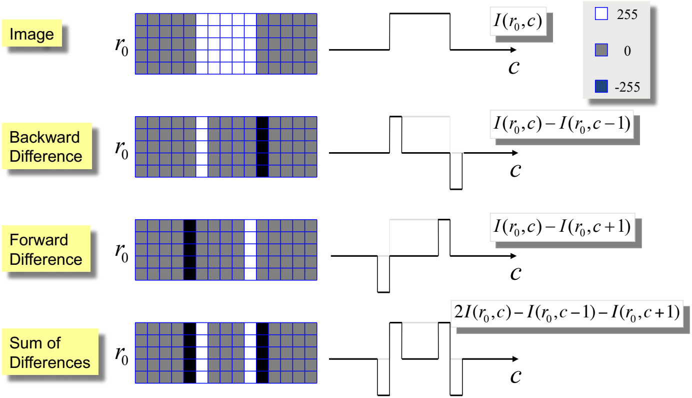


Convolution Examples: 17 × 17 Blur

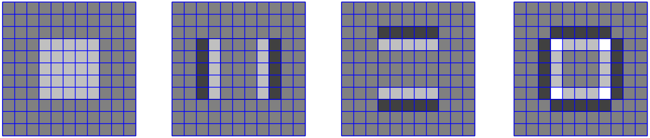
$$\frac{1}{289}\begin{bmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}$$



Vertical Edge Detection



Symmetric Edge Detection



$I(r,c)$

$2I(r,c) - I(r,c-1) - I(r,c+1)$

$2I(r,c) - I(r-1,c) - I(r+1,c)$

$4I(r,c) - I(r-1,c) - I(r+1,c) - I(r,c-1) - I(r,c+1)$

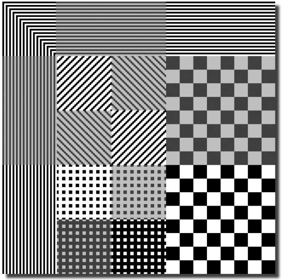
☐ 510
░ 255
▒ 0
■ -255

-1	2	-1

	-1	
	2	
	-1	

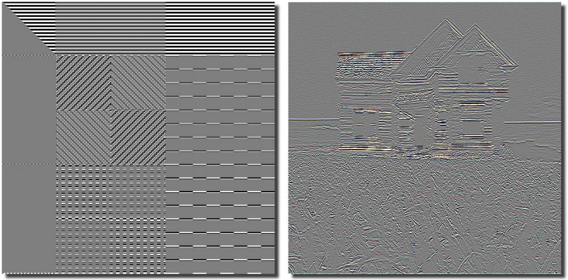
	-1	
-1	4	-1
	-1	

Convolution Examples: Original Images



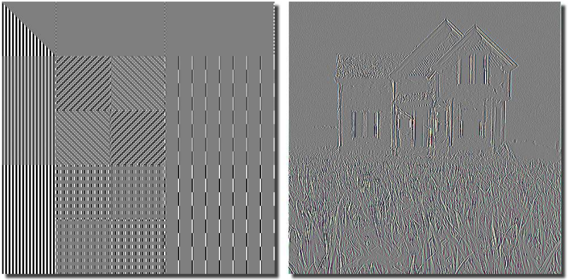
Convolution Examples: Vertical Difference

$$\begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}$$



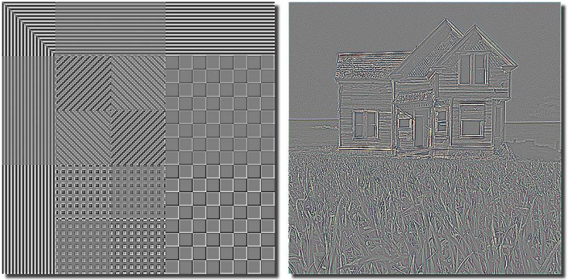
Convolution Examples: Horizontal Difference

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$$



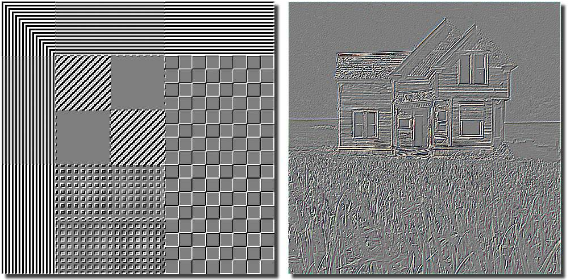
Convolution Examples: H + V Diff.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



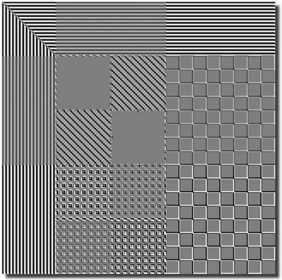
Convolution Examples: Diagonal Difference

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$



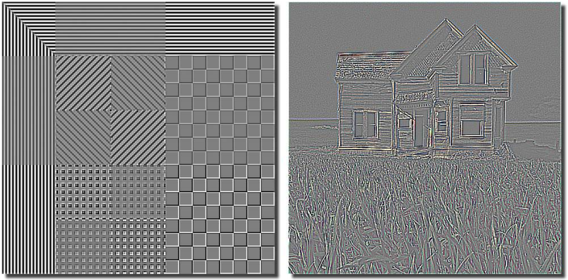
Convolution Examples: Diagonal Difference

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 2 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$



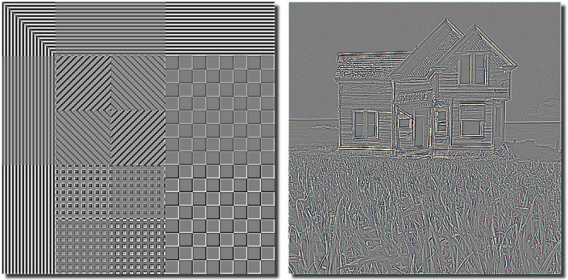
Convolution Examples: D + D Difference

$$\begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix}$$



Convolution Examples: H + V + D Diff.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



The Median Filter

- Returns the median value of the pixels in a neighborhood
- Is non-linear
- Is similar to a uniform blurring filter which returns the mean value of the pixels in a neighborhood of a pixel
- Unlike a mean value filter the median tends to preserve step edges

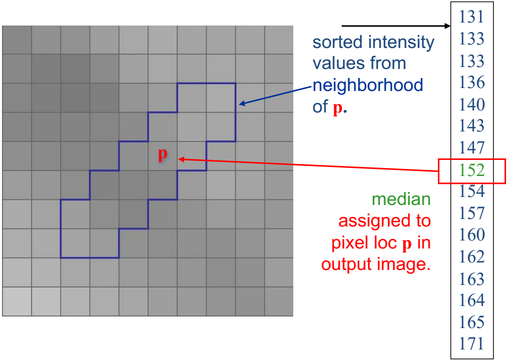
Median Filter: General Definition

$$\text{med}\left\{\mathbf{I},\mathbf{Z}\right\}\left(\mathbf{p}\right)=\underset{\mathbf{q}\in\text{supp}\left(\mathbf{Z}+\mathbf{p}\right)}{\text{median}}\left\{\mathbf{I}\left(\mathbf{q}\right)\right\}$$

This can be computed as follows:

1. Let **I** be a monochrome (1-band) image.
2. Let **Z** define a neighborhood of arbitrary shape.
3. At each pixel location, $\mathbf{p} = (r,c)$, in **I** …
4. … select the n pixels in the **Z**-neighborhood of **p**,
5. … sort the n pixels in the neighborhood of **p,** by value, into a list $L(j)$ for $j = 1,\ldots,n$.
6. The output value at **p** is $L(m)$, where $m = \lfloor\frac{n}{2}\rfloor + 1$

Median Filter: General Definition

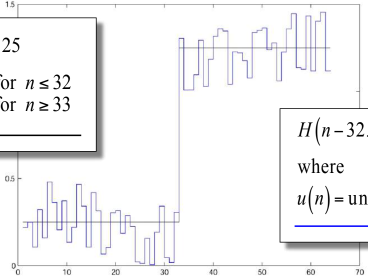


sorted intensity values from neighborhood of **p.**

131
133
133
136
140
143
147
152
154
157
160
162
163
164
165
171

median assigned to pixel loc **p** in output image.

A Noisy Step Edge



$$H(n-32.5)+0.25$$

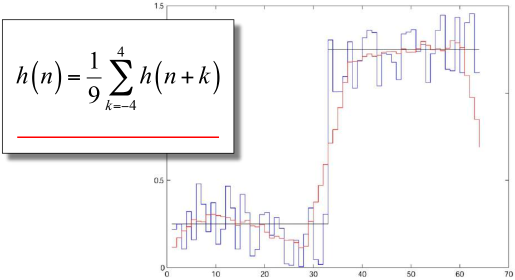
$$=\begin{cases} 0.25 & \text{for } n \le 32 \\ 1.25 & \text{for } n \ge 33 \end{cases}$$

$$H(n-32.5)+u(n)$$

where

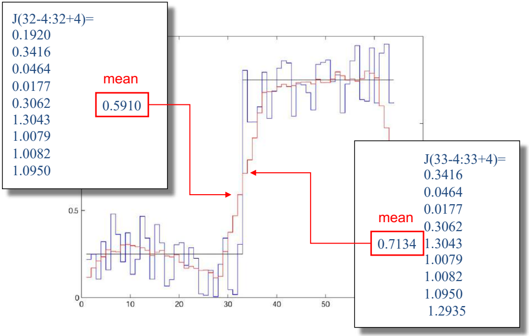
$$u(n)=\text{unif}\,(-0.25\,,0.25)$$

Blurred Noisy 1D Step Edge



$$h(n) = \frac{1}{9}\sum_{k=-4}^{4} h(n+k)$$

Blurred Noisy 1D Step Edge



J(32-4:32+4)=
0.1920
0.3416
0.0464
0.0177
0.3062
1.3043
1.0079
1.0082
1.0950

mean

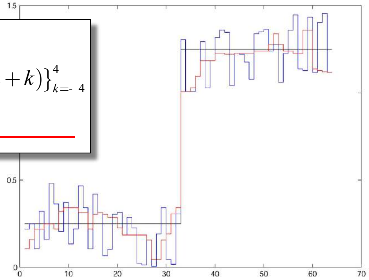
0.5910

J(33-4:33+4)=
0.3416
0.0464
0.0177
0.3062
1.3043
1.0079
1.0082
1.0950
1.2935

mean

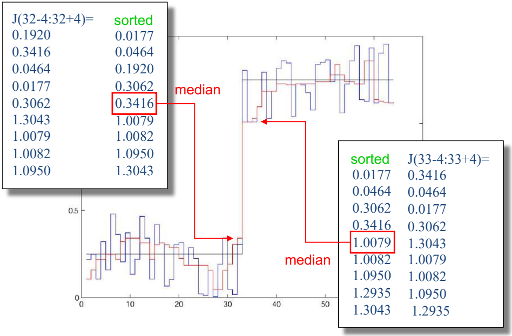
0.7134

Median Filtered Noisy 1D Step Edge



$$h(n) = \text{med}\{h(n+k)\}_{k=-4}^{4}$$

Median Filtered Noisy 1D Step Edge



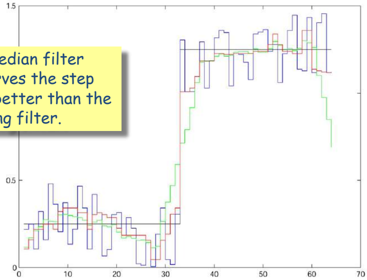
J(32-4:32+4)=	sorted
0.1920	0.0177
0.3416	0.0464
0.0464	0.1920
0.0177	0.3062
0.3062	0.3416
1.3043	1.0079
1.0079	1.0082
1.0082	1.0950
1.0950	1.3043

median

sorted	J(33-4:33+4)=
0.0177	0.3416
0.0464	0.0464
0.3062	0.0177
0.3416	0.3062
1.0079	1.3043
1.0082	1.0079
1.0950	1.0082
1.2935	1.0950
1.3043	1.2935

median

Median vs. Blurred



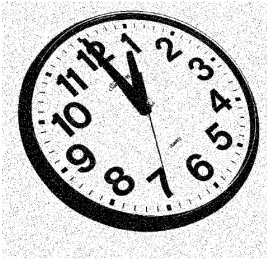
The median filter preserves the step edge better than the blurring filter.

median

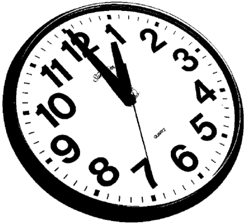
blurred

noisy

Median Filtering of Binary Images

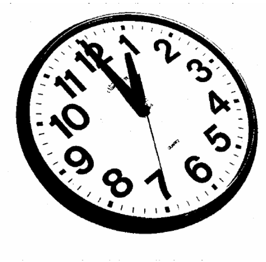


Noisy

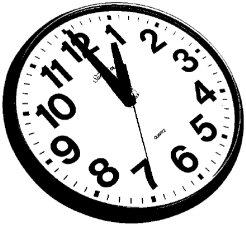


Original

Median Filtering of Binary Images

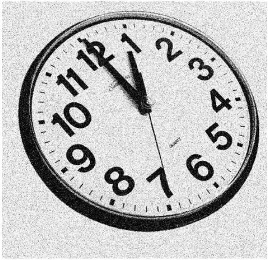


Median Filtered Noisy



Original

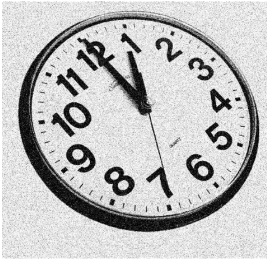
Filtering of Grayscale Images

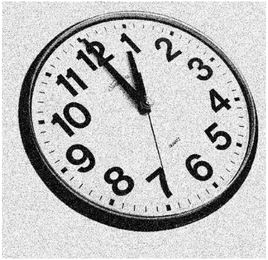


Noisy

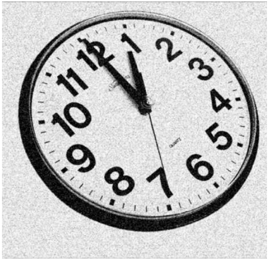
Original

Filtering of Grayscale Images

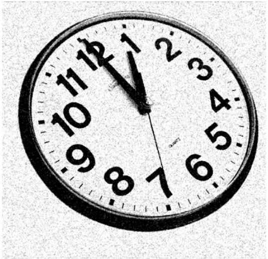

Noisy


Noisy

Filtering of Grayscale Images

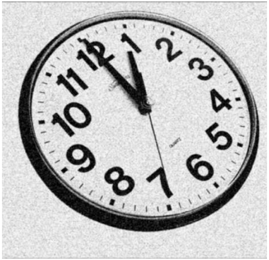


3x3-blur x 1

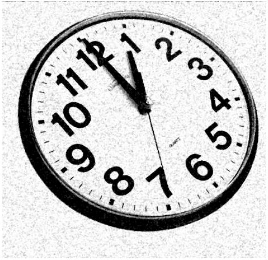


3x3-median x 1

Filtering of Grayscale Images

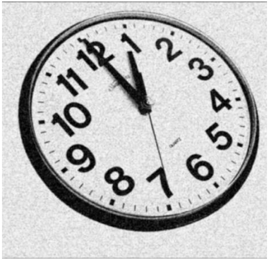


3x3-blur x 2

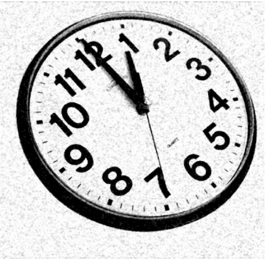


3x3-median x 2

Filtering of Grayscale Images

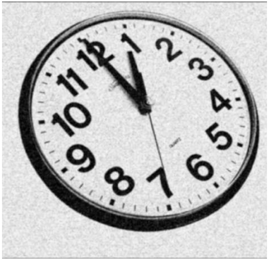


3x3-blur x 3

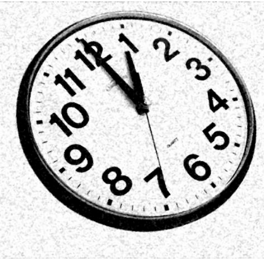


3x3-median x 3

Filtering of Grayscale Images

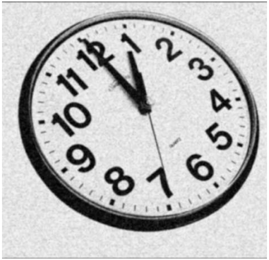


3x3-blur x 4

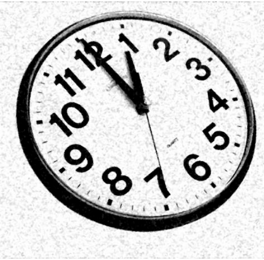


3x3-median x 4

Filtering of Grayscale Images

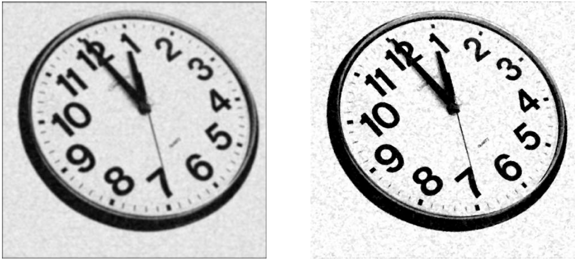


3x3-blur x 5



3x3-median x 5

Filtering of Grayscale Images


3x3-blur x 10

3x3-median x 10

Limit and Root Images

Fact: if you repeatedly filter an image with the same blurring filter or median filter, eventually the output does not change. That is, let

$$\mathbf{I}\left[*\mathbf{h}\right]^k \equiv \left(\left(\left(\mathbf{I}*\mathbf{h}\right)*\mathbf{h}\right)\cdots*\mathbf{h}\right), \quad k \text{ times, and}$$

$$\mathbf{I}\left[\text{med } \mathbf{Z}\right]^k \equiv \left(\left(\left(\mathbf{I} \text{ med } \mathbf{Z}\right) \text{ med } \mathbf{Z}\right)\cdots\text{med } \mathbf{Z}\right), \quad k \text{ times.}$$

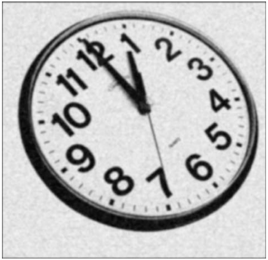
Then

$$\lim_{k\to\infty}\mathbf{I}\left[*\mathbf{h}\right]^k = \mathbf{I}\left[*\mathbf{h}\right]^n = \mathbf{I}_0, \quad \text{and}$$

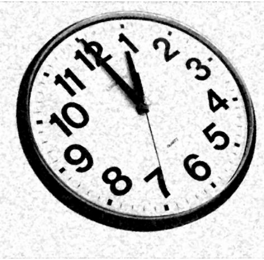
$$\lim_{k\to\infty}\mathbf{I}\left[\text{med } \mathbf{Z}\right]^k = \mathbf{I}\left[\text{med } \mathbf{Z}\right]^m = \mathbf{I}_\text{r},$$

where n and m are integers $(<\infty)$, $\mathbf{I}_0$ is a single-valued image and $\mathbf{I}_\text{r}$ is called the *median root* of **I**.

Limit and Root Images



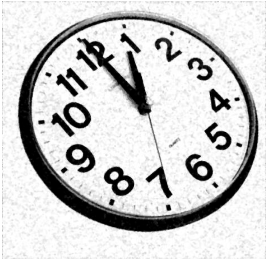
3x3-blur x 10



3x3-median x 10

Limit and Root Images

$3 \times 3\text{-blur } \times n \to \infty$



3x3-median root

Median Filter Algorithm in Matlab

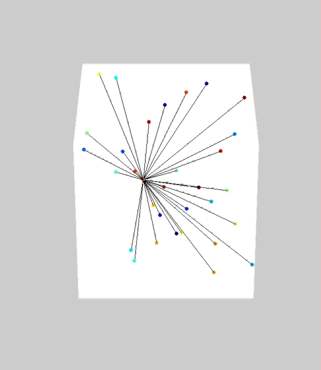
```
function D = median_filt(I,SE,origy,origx)
[R,C] = size(I);     % assumes 1-band image
[SER,SEC] = size(SE); % SE < 0 not in nbhd

N = sum(sum(SE>=0));  % no. of pixels in nbhd
A = -ones(R+SER-1,C+SEC-1,N); % accumulator
n=1; % copy I into band n of A for nbhd pix n
for j = 1 : SER % neighborhood is def’d in SE
   for i = 1 : SEC
      if SE(j,i) >= 0 % then is a nbhd pixel
         A(j:(R+j-1),i:(C+i-1),n) = I;
         n=n+1; % next accumulator band
      end
   end
end
% pixel-wise median across the bands of A
A = shiftdim(median(shiftdim(A,2)),1);
D = A( origy:(R+origy-1) , origx:(C+origx-1) );
return;
```

Vector Median Filter

A vector median filter selects from among a set of vectors, the one vector that is closest to all the others.

That is, if S is a set of vectors, in $\mathbb{F}^n$ the median, $\overline{\mathrm{v}}$, is

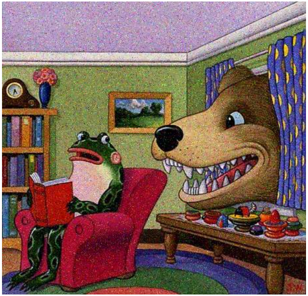
$$\overline{\mathbf{v}} = \arg\min_{k \neq j} \left\{ \left\| \mathbf{v}_k - \mathbf{v}_j \right\| \, \middle| \, \mathbf{v}_k, \mathbf{v}_j \in S \right\}.$$



$\trianglelefteq \mathbb{F}^n$ is an n-dimensional linear vector space over the field, $\mathbb{F}$.)

Color Median Filter

Jim Woodring – A Warm Shoulder



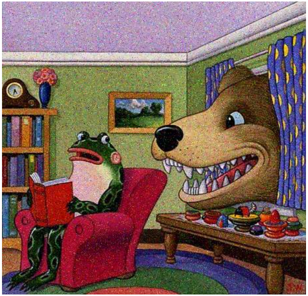
Sparse noise, 32% coverage in each band

Color Median Filter

3 × 3 color median filter applied once

3 × 3 color median filter applied twice

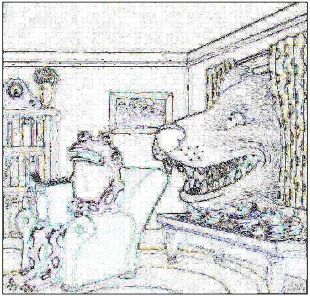
Color Median Filter


Sparse noise, 32% coverage in each band

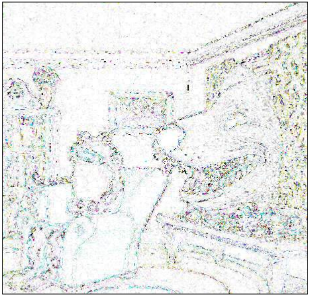

Jim Woodring – A Warm Shoulder

Color Median Filter

Absolute differences displayed as negatives to enhance visibility



(3 × 3 CMF^2 of noisy) – original



(3 × 3 CMF^2 of noisy) – (3 × 3 CMF^2 of original)

CMF vs. Standard Median on Individual Bands

A color median filter has to compute the distances between all the color vectors in the neighborhood of each pixel. That's expensive computationally.

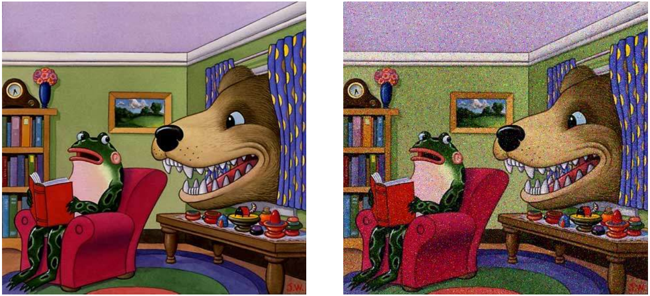
Q: Why not simply take the 1-band median of each color band individually?

A: The result at a pixel could be a color that did not exist in the pixel's neighborhood in the input image. The result is not the median of the colors – it is the median of the intensities of each color band treated independently.

Q: Is that a problem?

A: Maybe. Maybe not. It depends on the application. It may make little difference visually. If the colors need to be preserved, it could be problematic.

CMF vs. Standard Median on Individual Bands



Jim Woodring – A Warm Shoulder

Sparse noise, 32% coverage in each band

CMF vs. Standard Median on Individual Bands

3 × 3 color median filter applied once

3 × 3 color median filter applied twice

CMF vs. Standard Median on Individual Bands

3 × 3 median filter applied to each band once

3 × 3 median filter applied to each band twice

中国科学技术大学
University of Science and Technology of China

Gradient Image Processing

Juyong Zhang
School of Mathematics, USTC

Today: Gradient manipulation

Idea:

- Human visual system is very sensitive to gradient
- Gradient encode edges and local contrast quite well

- Do your editing in the gradient domain
- Reconstruct image from gradient

$$\nabla$$

- Various instances of this idea, I’ll mostly follow Perez et al. Siggraph 2003
  http://research.microsoft.com/vision/cambridge/papers/perez_siggraph03.pdf

Problems with direct cloning

sources/destinations

cloning

From Perez et al. 2003

Solution: clone gradient

sources/destinations

seamless cloning

Gradients and grayscale images

- Grayscale image: n × n scalars
- Gradient: n × n 2D vectors
- Overcomplete!
- What’s up with this?
- Not all vector fields are the gradient of an image!
- Only if they are curl-free (a.k.a. conservative)
  - But it does not matter for us

Today message I

- Manipulating the gradient is powerful

Today message II

- Optimization is powerful
  - In particular least square

- Good Least square optimization reduces to a big linear system
- Linear algebra is your friend
  - Big sparse linear systems can be solved efficiently

Today message III

- Toy examples are good to further understanding
- 1D can however be overly simplifying, n-D is much more complicated

Seamless Poisson cloning

- Given vector field *v* (pasted gradient), find the value of *f* in unknown region that optimize:

$$\min_{f} \iint_{\Omega} |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Poisson equation with Dirichlet conditions

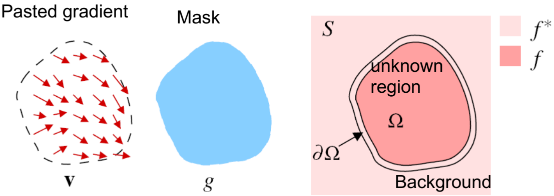
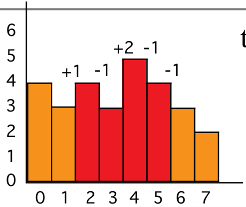


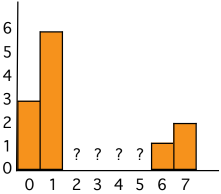
Figure 1: **Guided interpolation notations**. Unknown function f interpolates in domain Ω the destination function f^*, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function g.

Discrete 1D example: minimization

- Copy



to

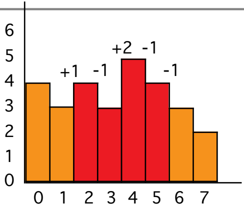


- Min $((f_2-f_1)-1)^2$
- Min $((f_3-f_2)-(-1))^2$
- Min $((f_4-f_3)-2)^2$
- Min $((f_5-f_4)-(-1))^2$
- Min $((f6-f_5)-(-1))^2$

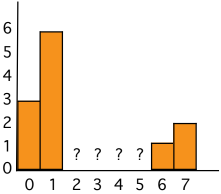
With
$f_1=6$
$f_6=1$

1D example: minimization

- Copy



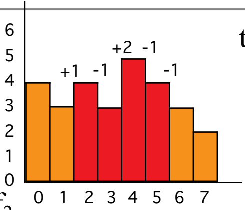
to



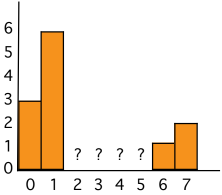
- Min $((f_2-6)-1)^2$ ==> $f_2{}^2+49-14f_2$
- Min $((f_3-f_2)-(-1))^2$ ==> $f_3{}^2+f_2{}^2+1-2f_3f_2 +2f_3-2f_2$
- Min $((f_4-f_3)-2)^2$ ==> $f_4{}^2+f_3{}^2+4-2f_3f_4 -4f_4+4f_3$
- Min $((f_5-f_4)-(-1))^2$ ==> $f_5{}^2+f_4{}^2+1-2f_5f_4 +2f_5-2f_4$
- Min $((1-f_5)-(-1))^2$ ==> $f_5{}^2+4-4f_5$

1D example: big quadratic

- Copy



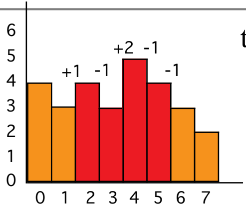
to



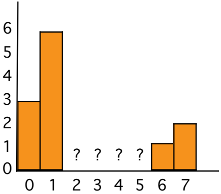
- Min $(f_2^2+49-14f_2$
  $+ f_3^2+f_2^2+1-2f_3f_2 +2f_3-2f_2$
  $+ f_4^2+f_3^2+4-2f_3f_4 -4f_4+4f_3$
  $+ f_5^2+f_4^2+1-2f_5f_4 +2f_5-2f_4$
  $+ f_5^2+4-4f_5)$
  Denote it Q

1D example: derivatives

- Copy



to



Min (f_2^2+49-14f_2
+ f_3^2+f_2^2+1-2$f_3$$f_2$ +2f_3-2f_2
+ f_4^2+f_3^2+4-2$f_3$$f_4$ -4f_4+4f_3
+ f_5^2+f_4^2+1-2$f_5$$f_4$ +2f_5-2f_4
+ f_5^2+4-4f_5)
Denote it Q

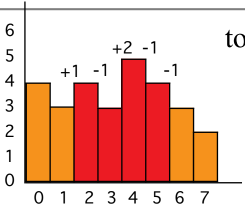
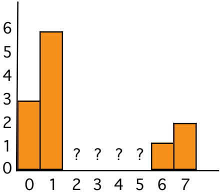
$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 16$$

$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4$$

$$\frac{dQ}{df_4} = 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2$$

$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2 + 2f_5 - 4$$

1D example: set derivatives to zero

- Copy  to 

$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 16$$

$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4$$

$$\frac{dQ}{df_4} = 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2$$

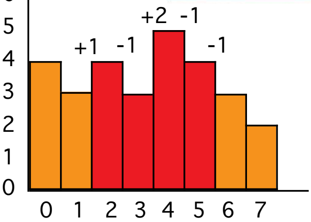
$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2 + 2f_5 - 4$$

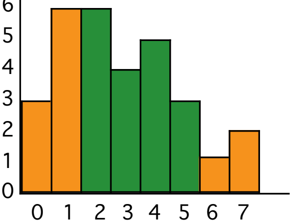
==> $$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

1D example

- Copy

to

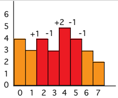


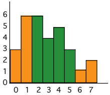


$$\begin{pmatrix}4&-2&0&0\\-2&4&-2&0\\0&-2&4&-2\\0&0&-2&4\end{pmatrix}\begin{pmatrix}f_2\\f_3\\f_4\\f_5\end{pmatrix}=\begin{pmatrix}16\\-6\\6\\2\end{pmatrix}$$

$$\begin{pmatrix}f_2\\f_3\\f_4\\f_5\end{pmatrix}=\begin{pmatrix}6\\4\\5\\3\end{pmatrix}$$

1D example: remarks

- Copy 

to 

$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

- Matrix is sparse
- Matrix is symmetric
- Everything is a multiple of 2
  - because square and derivative of square
- Matrix is a convolution (kernel -2 4 -2)
- Matrix is independent of gradient field. Only RHS is
- Matrix is a second derivative

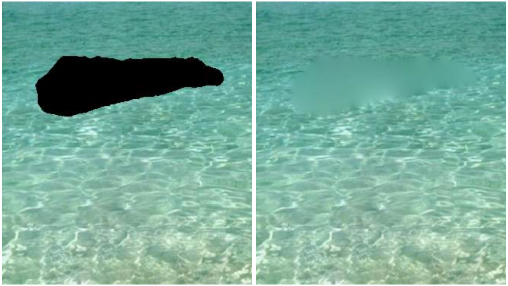
Let’s try to further analyze

- What is a simple case?

Membrane interpolation

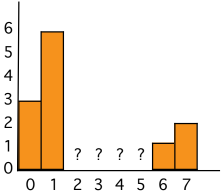
- What if *v* is null?
- Laplace equation (a.k.a. membrane equation)

$$\min_f \iint_\Omega |\nabla f|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$



1D example: minimization

- Minimize derivatives to interpolate

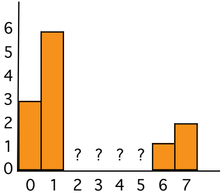


- Min $(f_2-f_1)^2$
- Min $(f_3-f_2)^2$
- Min $(f_4-f_3)^2$
- Min $(f_5-f_4)^2$
- Min $(f_6-f_5)^2$

With
$f_1=6$
$f_6=1$

1D example: derivatives

- Minimize derivatives to interpolate



Min (f_2^2+36-12f_2
+ f_3^2+f_2^2-2f_3f_2
+ f_4^2+f_3^2-2f_3f_4
+ f_5^2+f_4^2-2f_5f_4
+ f_5^2+1-2f_5)
Denote it Q

$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 12$$

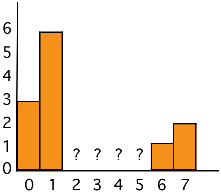
$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2f_3 - 2f_4$$

$$\frac{dQ}{df_4} = 2f_4 - 2f_3 + 2f_4 - 2f_5$$

$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2f_5 - 2$$

1D example: set derivatives to zero

- Minimize derivatives to interpolate



$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 12$$

$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2f_3 - 2f_4$$

$$\frac{dQ}{df_4} = 2f_4 - 2f_3 + 2f_4 - 2f_5$$

$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2f_5 - 2$$

==>

$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 12 \\ 0 \\ 0 \\ 2 \end{pmatrix}$$

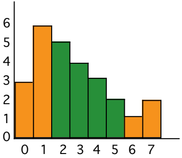
1D example

- Minimize derivatives to interpolate

- Pretty much says that second derivative should be zero

  (-1 2 -1)

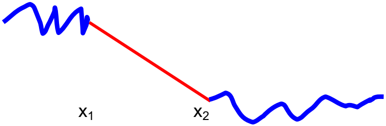
  is a second derivative filter



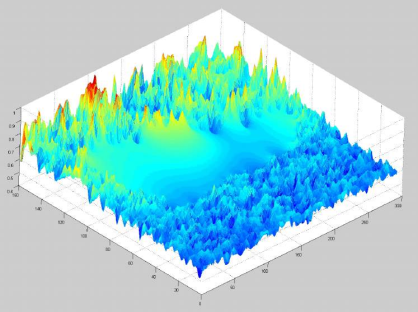
$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 12 \\ 0 \\ 0 \\ 2 \end{pmatrix} \qquad \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \end{pmatrix}$$

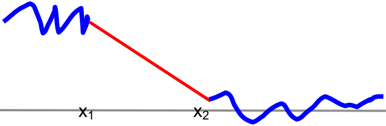
Intuition

- In 1D; just linear interpolation!
  - The min of $\int f'$ is the slope integrated over the interval
- Locally, if the second derivative was not zero, this would mean that the first derivative is varying, which is bad since we want $\int f'$ to be minimized
- Note that, in 1D: by setting f'', we leave two degrees of freedom. This is exactly what we need to control the boundary condition at x_1 and x_2



In 2D: membrane interpolation





Membrane interpolation

- What if *v* is null?
- Laplace equation (a.k.a. membrane equation)

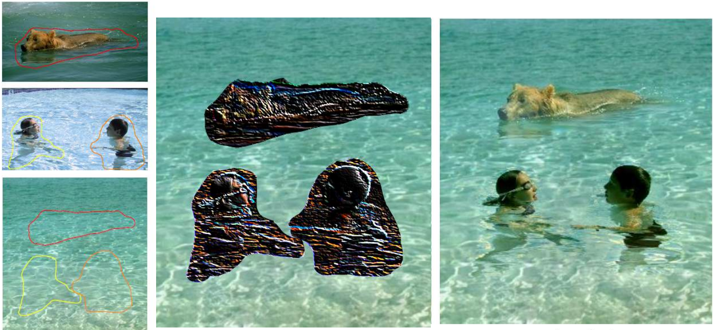
$$\min_f \iint_\Omega |\nabla f|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

- Mathematicians will tell you there is an Associated Euler-Lagrange equation:

$$\Delta f = 0 \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

  - Where the Laplacian Δ is similar to -1 2 -1in 1D
- Kind of the idea that we want a minimum, so we kind of derive and get a simpler equation

What is v is not null?



sources/destinations

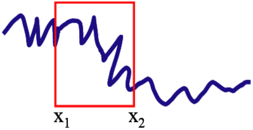
seamless cloning

What if *v* is not null?

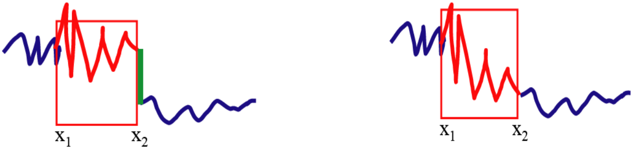
- 1D case

Seamlessly paste

onto



Just add a linear function so that the boundary condition is respected



(Review) Seamless Poisson cloning

- Given vector field *v* (pasted gradient), find the value of *f* in unknown region that optimize:

$$\min_{f} \iint_{\Omega} |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Poisson equation with Dirichlet conditions

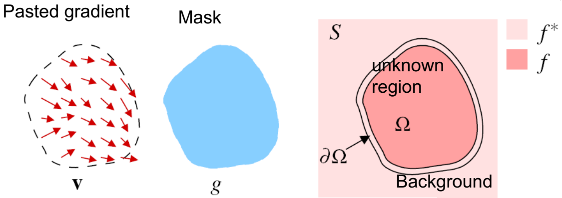


Figure 1: **Guided interpolation notations**. Unknown function f interpolates in domain Ω the destination function f^*, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function g.

What if *v* is not null: 2D

- Variational minimization (integral of a functional) with boundary condition

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega},$$

- Euler-Lagrange equation:

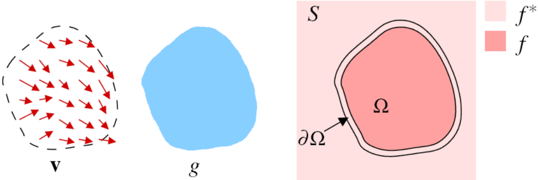
$$\Delta f = \text{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

where $\text{div}\mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $\mathbf{v} = (u, v)$

In 2D, if *v* is conservative

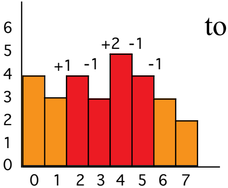
- If *v* is the gradient of an image g
- Correction function $\widehat{f}$ so that $f = g + \widehat{f}$
- $\widehat{f}$ performs membrane interpolation over Ω :

$$\Delta \tilde{f} = 0 \text{ over } \Omega, \;\; \tilde{f}|_{\partial\Omega} = (f^* - g)|_{\partial\Omega}$$

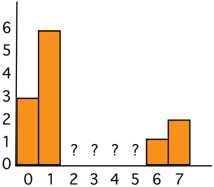


1D example

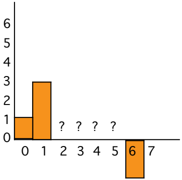
- Copy



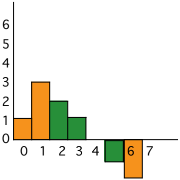
to



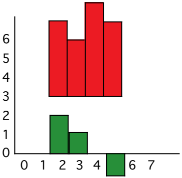
Difference



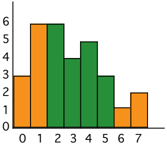
Solve Laplace



Add

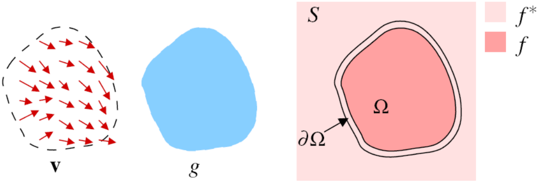


Result



In 2D, if *v* is NOT conservative

- Also need to project the vector field v to a conservative field
- And do the membrane thing
- Of course, we do not need to worry about it, it's all handled naturally by the least square approach

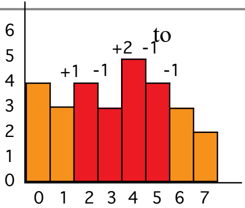


Recap

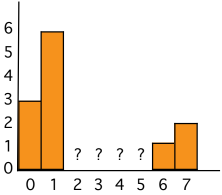
- Find image whose gradient best approximates the input gradient
  - least square Minimization
- Discrete case: turns into linear equation
  - Set derivatives to zero
  - Derivatives of quadratic ==> linear
- Continuous: turns into Euler-Lagrange form
  - $\Delta f = \text{div } v$
- When gradient is null, membrane interpolation
  - Linear interpolation in 1D

Discrete solver: Recall 1D

- Copy



to



$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 16$$

$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4$$

$$\frac{dQ}{df_4} = 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2$$

$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2 + 2f_5 - 4$$

==> $$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

Discrete Poisson solver

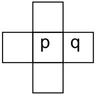
- Two approaches:
  - Minimize variational problem
  - Solve Euler-Lagrange equation

  In practice, variational is best

$$\min_{f} \iint_{\Omega} |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega},$$

$$\Delta f = \mathrm{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

- In both cases, need to discretize derivatives
  - Finite differences over 4 pixel neighbors
  - We are going to work using pairs
    - Partial derivatives are easy on pairs
    - Same for the discretization of v



Discrete Poisson solver

- Minimize variational problem $\min_{f} \iint_{\Omega} |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega},$

Discretized gradient

$$\min_{f|_{\Omega}} \sum_{\langle p,q\rangle \cap \Omega \neq \emptyset} (f_p - f_q - v_{pq})^2, \text{ with } f_p = f_p^*, \text{for all } p \in \partial\Omega$$

(all pairs that are in Ω)

Discretized v: g(p)-g(q)

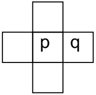
Boundary condition

- Rearrange and call N_p the neighbors of p

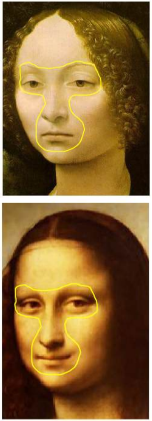
$$\text{for all } p \in \Omega, \quad |N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in \underbrace{N_p \cap \partial}\Omega} f_q^* + \sum_{q \in N_p} v_{pq}$$

Only for boundary pixels

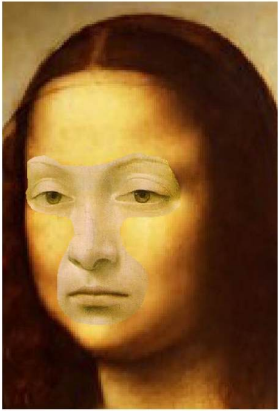
- Big yet sparse linear system



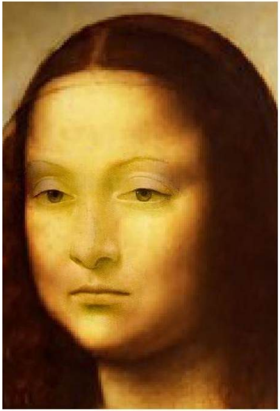
Result (eye candy)



source/destination



cloning



seamless cloning

Recap

- Find image whose gradient best approximates the input gradient
  - least square Minimization
- Discrete case: turns into big sparse linear equation
  - Set derivatives to zero
  - Derivatives of quadratic ==> linear

Solving big matrix systems

- Ax=b
- You can use Matlab’s \
  - (Gaussian elimination)
  - But not very scalable

Iterative solvers

Important ideas

- Do not inverse matrix
- Maintain a vector x’ that progresses towards the solution
- Updates mostly require to *apply* the matrix.
  - In many cases, it means you do no even need to store the matrix (e.g. for a convolution matrix you only need the kernel)
- Usually, you don’t even wait until convergence
- Big questions: in which direction do you walk?
  - Yes, very similar to gradient descent

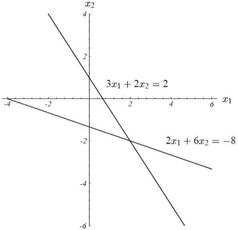
Solving big matrix systems

- Ax=b, where A is sparse (many zero entries)
- In Pset 3, we ask you to use conjugate gradient
  - http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf
  - http://www.library.cornell.edu/nr/bookcpdf/c10-6.pdf

Ax=b

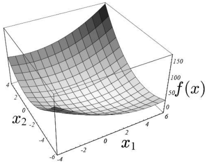
- A is square, symmetric and positive-definite
- When A is dense, you're stuck, use backsubstitution
- When A is sparse, iterative techniques (such as Conjugate Gradient) are faster and more memory efficient
- Simple example:

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

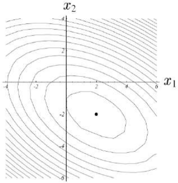


Turn Ax=b into a minimization problem

- Minimization is more logical to analyze iteration (gradient ascent/descent)
- Quadratic form $f(x) = \frac{1}{2}x^T Ax - b^T x + c$
  - c can be ignored because we want to minimize
- Intuition:
  - the solution of a linear system is always the intersection of n hyperplanes
  - Take the square distance to them
  - A needs to be positive-definite so that we have a nice parabola



Graph of quadratic form $f(x) = \frac{1}{2}x^T Ax - b^T x + c$. The minimum point of this surface is the solution to $Ax = b$.



Contours of the quadratic form. Each ellipsoidal curve has constant $f(x)$.

Conjugate gradient

- Smarter choice of direction
  - Ideally, step directions should be orthogonal to one another (no redundancy)
  - But tough to achieve
  - Next best thing: make them A-orthogonal (conjugate)
    That is, orthogonal when transformed by A: $d_{(i)}^T A d_{(j)} = 0$

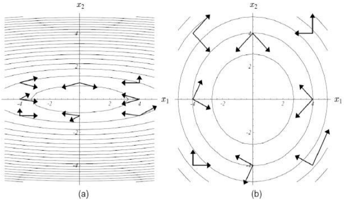


Figure 22: These pairs of vectors are A-orthogonal . . . because these pairs of vectors are orthogonal.

Recap

- Poisson image cloning: paste gradient, enforce boundary condition
- Variational formulation

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega},$$

- Also Euler-Lagrange formulation

$$\Delta f = \operatorname{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

- Discretize variational version,
  leads to big but sparse linear system
- Conjugate gradient is a smart iterative technique to solve it

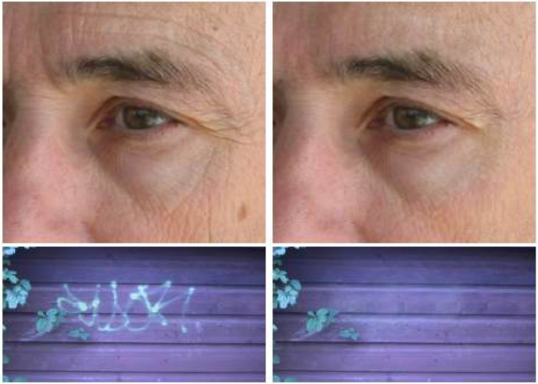
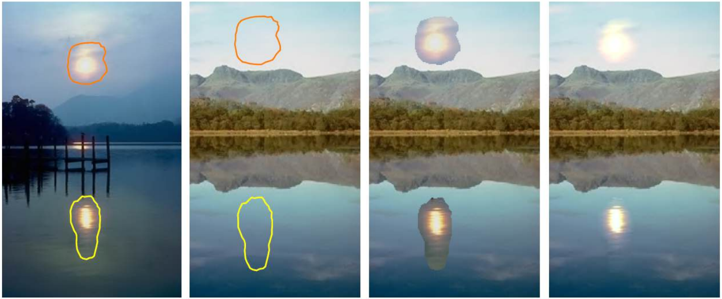


Figure 2: **Concealment**. By importing seamlessly a piece of the background, complete objects, parts of objects, and undesirable artifacts can easily be hidden. In both examples, multiple strokes (not shown) were used.



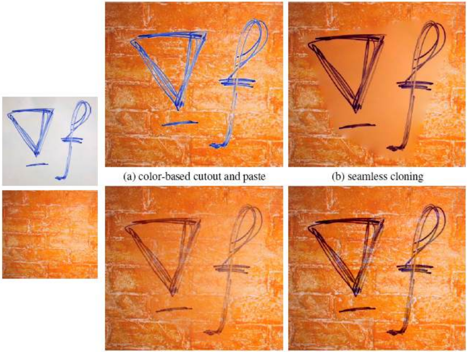
sources destinations cloning seamless cloning

Manipulate the gradient

- Mix gradients of g & f: take the max

source/destination seamless cloning mixed seamless cloning

Figure 8: **Inserting one object close to another**. With seamless cloning, an object in the destination image touching the selected region Ω bleeds into it. Bleeding is inhibited by using mixed gradients as the guidance field.



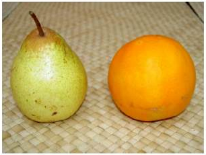
(a) color-based cutout and paste

(b) seamless cloning

(c) seamless cloning and destination averaged

(d) mixed seamless cloning

Figure 6: **Inserting objects with holes**. (a) The classic method, color-based selection and alpha masking might be time consuming and often leaves an undesirable halo; (b-c) seamless cloning, even averaged with the original image, is not effective; (d) mixed seamless cloning based on a loose selection proves effective.



swapped textures

source

destination

Figure 7: **Inserting transparent objects**. Mixed seamless cloning facilitates the transfer of partly transparent objects, such as the rainbow in this example. The non-linear mixing of gradient fields picks out whichever of source or destination structure is the more salient at each location.

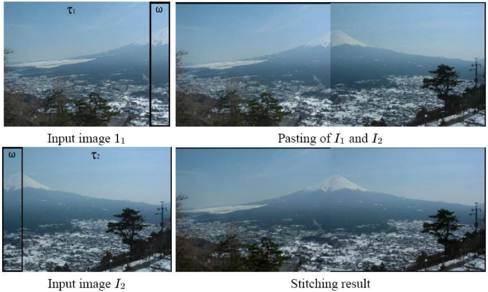
Reduce big gradients

- Dynamic range compression
- See Fattal et al. 2002

Figure 10: **Local illumination changes**. Applying an appropriate non-linear transformation to the gradient field inside the selection and then integrating back with a Poisson solver, modifies locally the apparent illumination of an image. This is useful to highlight under-exposed foreground objects or to reduce specular reflections.

Seamless Image Stitching in the Gradient Domain

- Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss
  http://www.cs.huji.ac.il/~alevin/papers/eccv04-blending.pdf
  http://eprints.pascal-network.org/archive/00001062/01/tips05-blending.pdf
- Various strategies (optimal cut, feathering)



Input image 1_1 Pasting of I_1 and I_2

Input image I_2 Stitching result

Fig. 1. Image stitching. On the left are the input images. ω is the overlap region. On top right is a simple pasting of the input images. On the bottom right is the result of the GIST1 algorithm.

Poisson Matting

- Sun et al. Siggraph 2004
- Assume gradient of F & B is negligible
- Plus various image-editing tools to refine matte

$$I = \alpha F + (1 - \alpha) B$$

$$\nabla I = (F - B)\nabla\alpha + \alpha\nabla F + (1 - \alpha)\nabla B$$

$$\nabla\alpha \approx \frac{1}{F - B}\nabla I$$

Figure 1: Pulling of matte from a complex scene. From left to right: a complex natural image for existing matting techniques where the color background is complex, a high quality matte generated by Poisson matting, a composite image with the extracted koala and a constant-color background, and a composite image with the extracted koala and a different background.

Poisson-ish mesh editing

- http://portal.acm.org/citation.cfm?id=1057432.1057456
- http://www.cad.zju.edu.cn/home/xudong/Projects/mesh_ec
- http://people.csail.mit.edu/sumner/research/deftransfer/

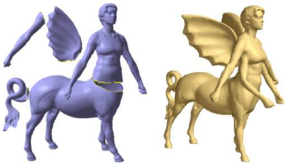


Figure 1: An unknown mythical creature. Left: mesh components for merging and deformation (the arm). Right: final editing result.

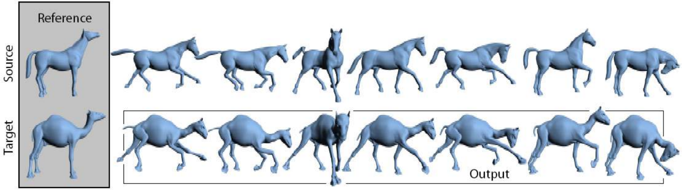
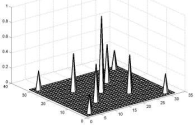


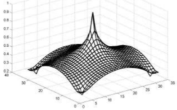
Figure 1: Deformation transfer copies the deformations exhibited by a source mesh onto a different target mesh. In this example, deformations of the reference horse mesh are transfered to the reference camel, generating seven new camel poses. Both gross skeletal changes as well as more subtle skin deformations are successfully reproduced.

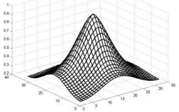
Alternative to membrane

- Thin plate:
  minimize *second* derivative

$$\text{min}_f \int\int f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2 \text{dxdy}$$


Data


Membrane interpolation


Thin-plate interpolation

Inpainting

- More elaborate energy functional/PDEs
- http://www-mount.ee.umn.edu/~guille/inpainting.htm

Key references

- Socolinsky, D. *Dynamic Range Constraints in Image Fusion and Visualization* 2000. http://www.equinoxsensors.com/news.html
- Elder, Image editing in the contour domain, 2001 http://elderlab.yorku.ca/~elder/publications/journals/ElderPAMI01.pdf
- Fattal et al. 2002
  Gradient Domain HDR Compression http://www.cs.huji.ac.il/%7Edanix/hdr/
- Poisson Image Editing Perez et al. http://research.microsoft.com/vision/cambridge/papers/perez_siggraph03.pdf
- Covariant Derivatives and Vision, Todor Georgiev (Adobe Systems) ECCV 2006

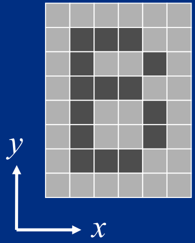
A Gentle Introduction
to Bilateral Filtering
and its Applications

SIGGRAPH2007

Naïve Image Smoothing: Gaussian Blur

Sylvain Paris – MIT CSAIL

Notation and Definitions



- Image = 2D array of pixels

- Pixel = intensity (scalar) or color (3D vector)

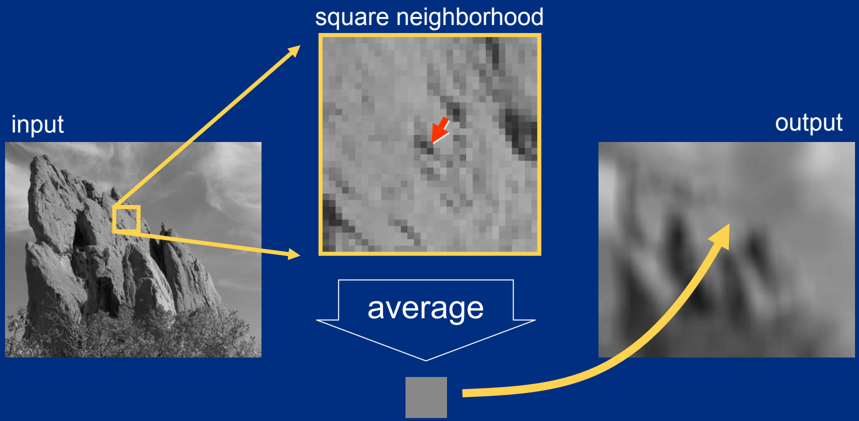
- $I_{\mathbf{p}}$ = value of image I at position: $\mathbf{p} = (\, p_x \,, p_y \,)$

- $F\,[\,I\,]$ = output of filter F applied to image I

Strategy for Smoothing Images

- Images are not smooth because adjacent pixels are different.
- Smoothing = making adjacent pixels
  look more similar.
- Smoothing strategy
  pixel → average of its neighbors

Box Average



input

square neighborhood

average

output

Equation of Box Average

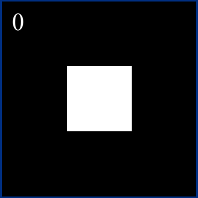
$$BA[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} B_{\sigma}(\mathbf{p}-\mathbf{q})\, I_{\mathbf{q}}$$

result at pixel $\mathbf{p}$

sum over all pixels $\mathbf{q}$

intensity at pixel $\mathbf{q}$

normalized box function



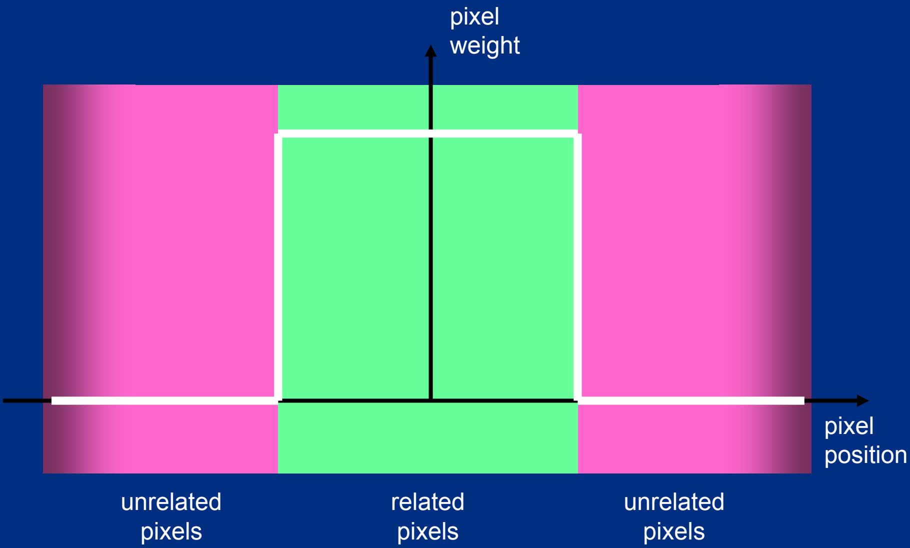
Square Box Generates Defects

- Axis-aligned streaks
- Blocky results

input

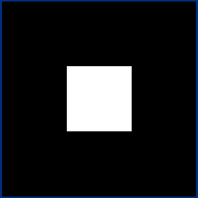
output

Box Profile

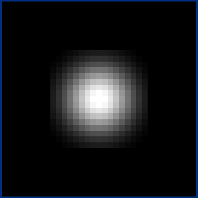

pixel
weight
pixel
position
unrelated
pixels
related
pixels
unrelated
pixels

Strategy to Solve these Problems

- Use an isotropic (*i.e.* circular) window.
- Use a window with a smooth falloff.

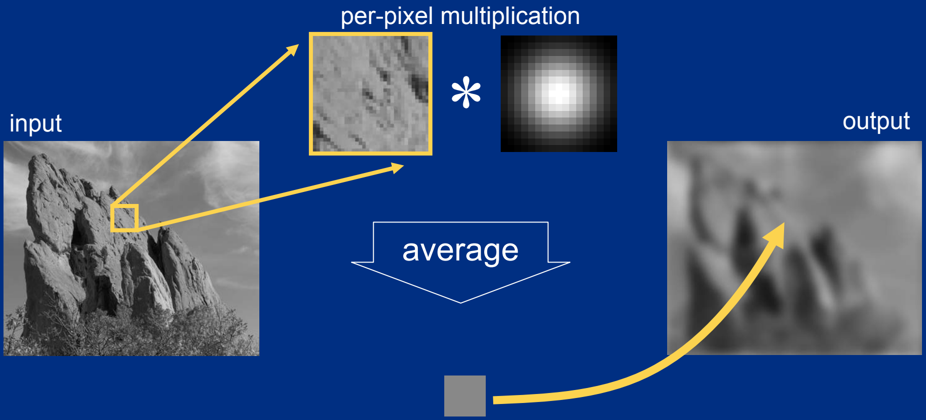


box window



Gaussian window

Gaussian Blur



per-pixel multiplication

input

*

output

average

input

box average

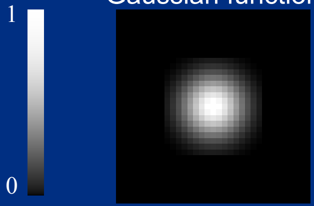
Gaussian blur

Equation of Gaussian Blur

Same idea: **weighted average of pixels.**

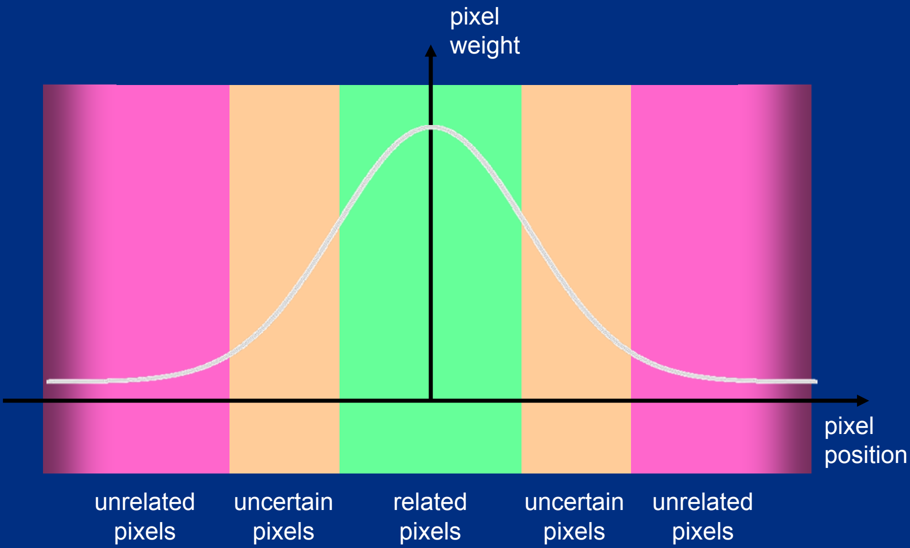
$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} \boxed{G_{\sigma}(\|\,\mathbf{p}-\mathbf{q}\,\|)}\, I_{\mathbf{q}}$$

normalized
Gaussian function



Gaussian Profile

$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

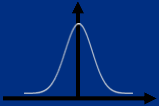


Spatial Parameter

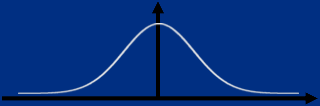
input

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} G_{\boxed{\sigma}}(\|\,\mathbf{p}-\mathbf{q}\,\|)\, I_{\mathbf{q}}$$

size of the window



small σ



large σ

limited smoothing

strong smoothing

How to set σ

- Depends on the application.

- Common strategy: proportional to image size
  - e.g. 2% of the image diagonal
  - property: independent of image resolution

Properties of Gaussian Blur

- Weights independent of spatial location
  - linear convolution
  - well-known operation
  - efficient computation (recursive algorithm, FFT)

Properties of Gaussian Blur

- Does smooth images
- But smoothes too much: **edges are blurred.**
  - Only spatial distance matters
  - No edge term

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma}(\|\mathbf{p}-\mathbf{q}\|)}_{\text{space}} I_{\mathbf{q}}$$

input

output

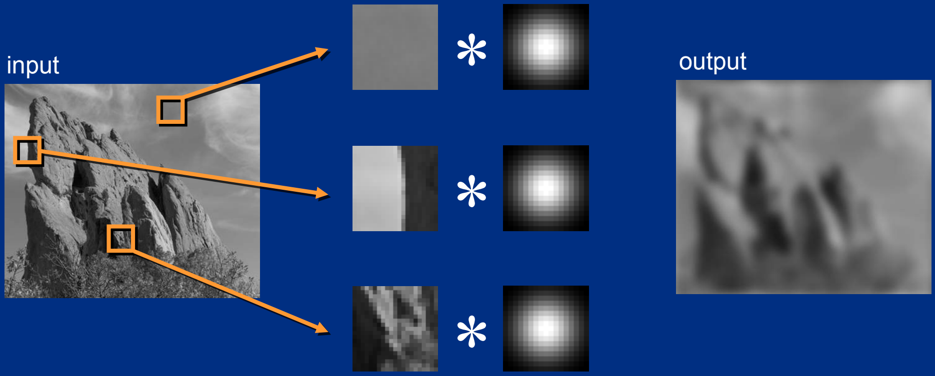
A Gentle Introduction
to Bilateral Filtering
and its Applications

SIGGRAPH2007

“Fixing the Gaussian Blur”: the Bilateral Filter

Sylvain Paris – MIT CSAIL

Blur Comes from Averaging across Edges



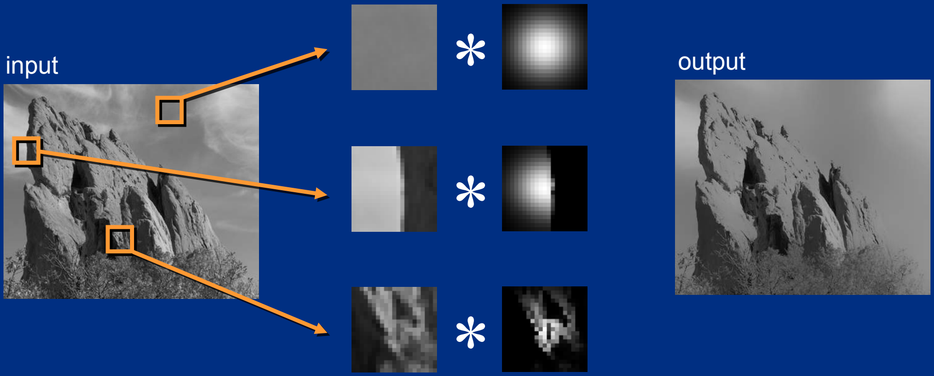
input

output

Same Gaussian kernel everywhere.

Bilateral Filter [Aurich 95, Smith 97, Tomasi 98]

No Averaging across Edges



The kernel shape depends on the image content.

Bilateral Filter Definition: an Additional Edge Term

Same idea: **weighted average of pixels.**

$$BF[I]_{\mathbf{p}} = \overset{\text{new}}{\underbrace{\boxed{\frac{1}{W_{\mathbf{p}}}}}_{\text{normalization factor}}} \sum_{\mathbf{q}\in S} \overset{\text{not new}}{\underbrace{\boxed{G_{\sigma_{\mathrm{s}}}\left(\|\mathbf{p}-\mathbf{q}\|\right)}}_{\textit{space}\text{ weight}}} \overset{\text{new}}{\underbrace{\boxed{G_{\sigma_{\mathrm{r}}}\left(|I_{\mathbf{p}}-I_{\mathbf{q}}|\right)}}_{\textit{range}\text{ weight}}} I_{\mathbf{q}}$$

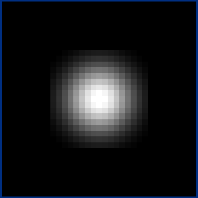
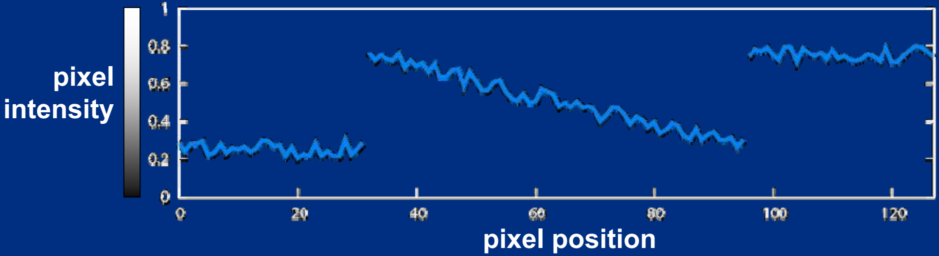


Illustration a 1D Image

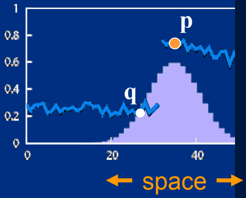
- 1D image = line of pixels

- Better visualized as a plot



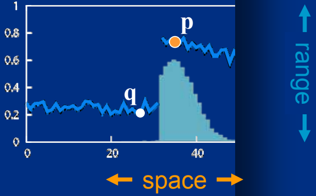
Gaussian Blur and Bilateral Filter

Gaussian blur

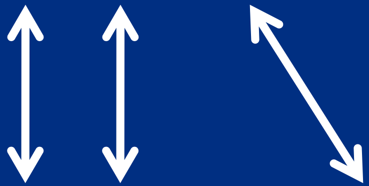


Bilateral filter

[Aurich 95, Smith 97, Tomasi 98]



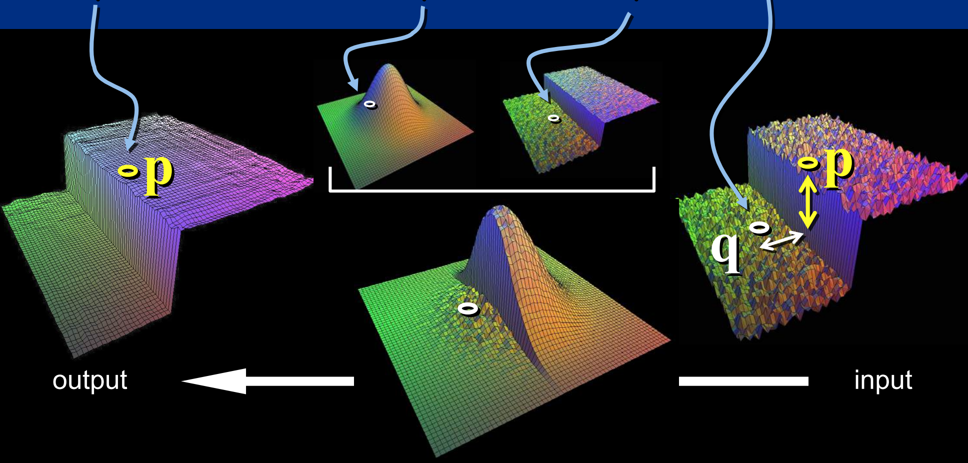
$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma}(\|\mathbf{p}-\mathbf{q}\|)}_{\text{space}} I_{\mathbf{q}}$$



$$BF[I]_{\mathbf{p}} = \underbrace{\frac{1}{W_{\mathbf{p}}}}_{\text{normalization}} \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)}_{\text{space}} \underbrace{G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}_{\text{range}} I_{\mathbf{q}}$$

Bilateral Filter on a Height Field

$$\underbrace{BF[I]_{\mathbf{p}}}_{} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma_s}(\|\,\mathbf{p}-\mathbf{q}\,\|)}_{} \; \underbrace{G_{\sigma_r}(|\,I_{\mathbf{p}} - I_{\mathbf{q}}\,|)}_{} \; I_{\mathbf{q}}$$



reproduced from [Durand 02]

Space and Range Parameters

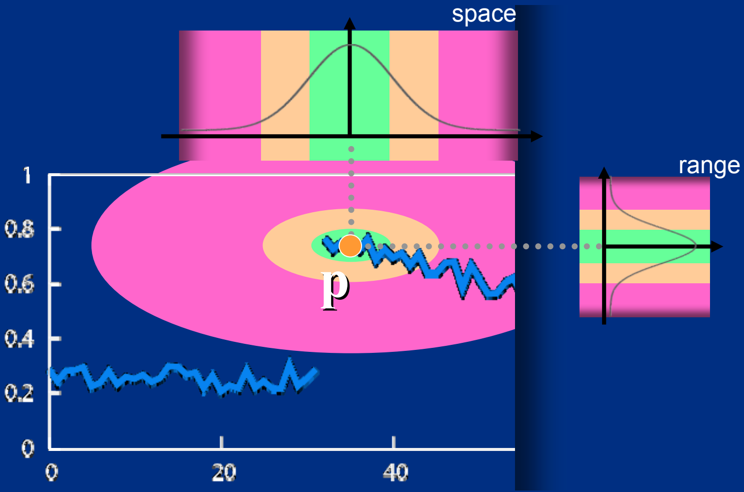
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_{\mathrm{s}}}(\| \mathbf{p} - \mathbf{q} \|) \, G_{\sigma_{\mathrm{r}}}(| I_{\mathbf{p}} - I_{\mathbf{q}} |) \, I_{\mathbf{q}}$$

- space σ_{s} : spatial extent of the kernel, size of the considered neighborhood.

- range σ_{r} : “minimum” amplitude of an edge

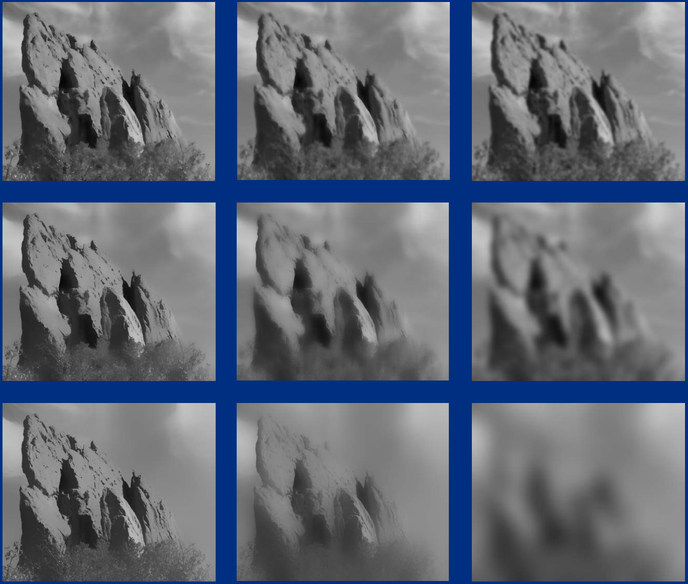
Influence of Pixels

Only pixels close in space and in range are considered.



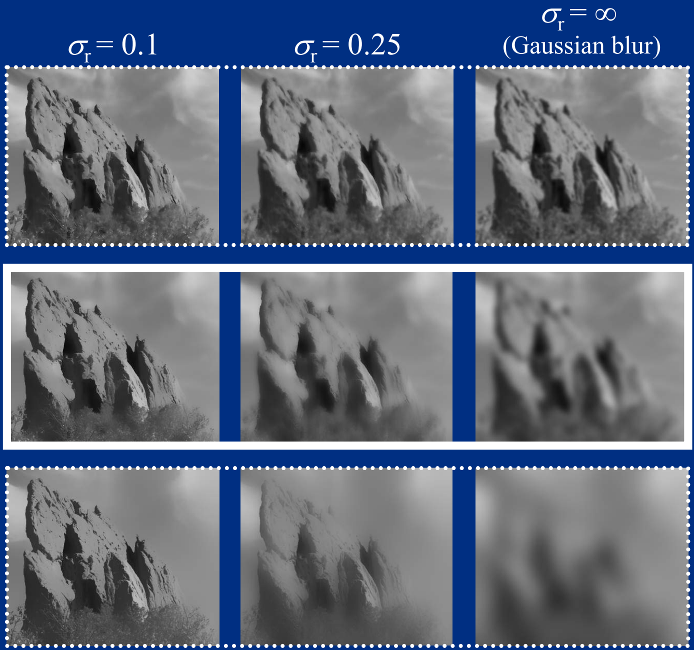
Exploring the Parameter Space

input



Varying the Range Parameter

input



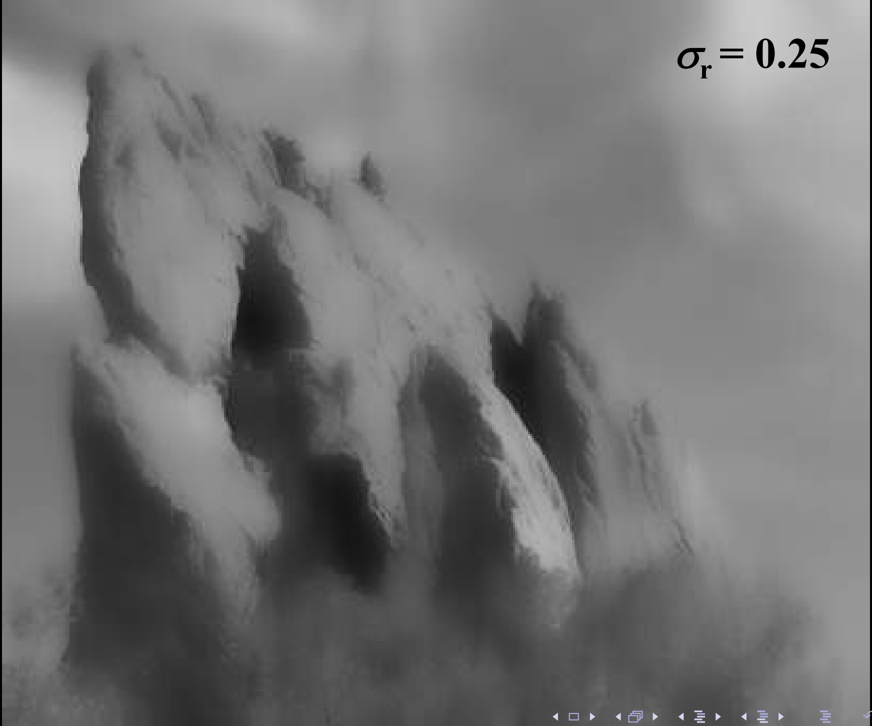
$\sigma_s = 2$

$\sigma_s = 6$

$\sigma_s = 18$

input

$\sigma_r = 0.1$

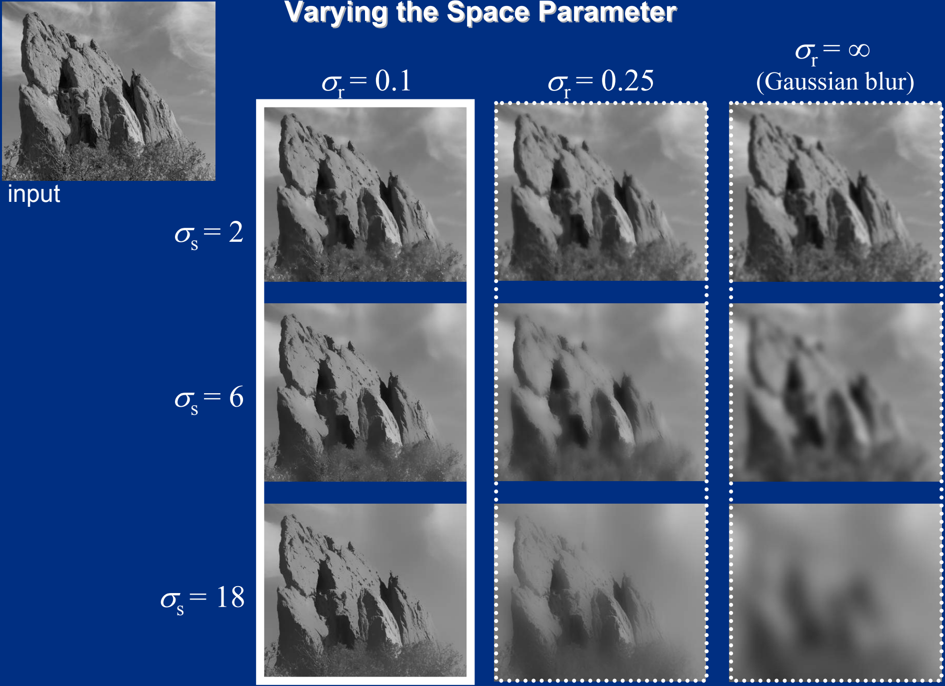


$\sigma_r = 0.25$

$\sigma_r = \infty$

(Gaussian blur)

Varying the Space Parameter




input

$\sigma_s = 2$

$\sigma_s = 6$

$\sigma_s = 18$

How to Set the Parameters

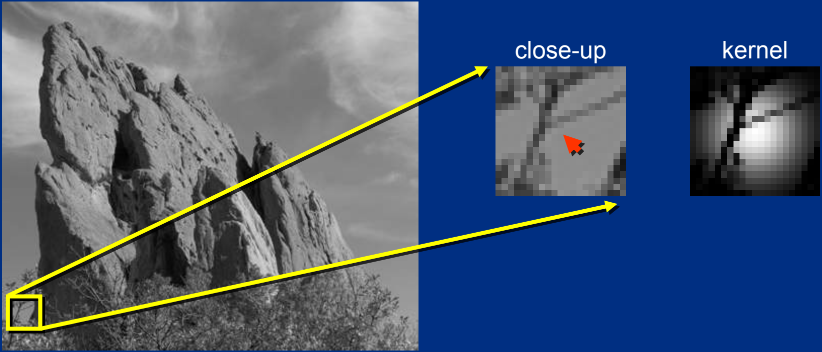
Depends on the application. For instance:

- space parameter: proportional to image size
  - e.g., 2% of image diagonal
- range parameter: proportional to edge amplitude
  - e.g., mean or median of image gradients
- independent of resolution and exposure

A Few
More Advanced
Remarks

Bilateral Filter Crosses Thin Lines

- Bilateral filter averages across features thinner than $\sim 2\sigma_s$
- Desirable for smoothing: more pixels = more robust
- Different from diffusion that stops at thin lines



Iterating the Bilateral Filter

$$I_{(n+1)} = BF\left[I_{(n)}\right]$$

- Generate more piecewise-flat images
- Often not needed in computational photo.

input

1 iteration

2 iterations

4 iterations

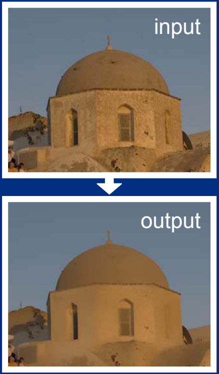
Bilateral Filtering Color Images

For gray-level images

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}\Big(\overbrace{|I_{\mathbf{p}} - I_{\mathbf{q}}|}^{\text{intensity difference}}\Big)\underbrace{I_{\mathbf{q}}}_{\text{scalar}}$$

For color images

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}\Big(\overbrace{\|\mathbf{C}_{\mathbf{p}} - \mathbf{C}_{\mathbf{q}}\|}^{\text{color difference}}\Big)\underbrace{\mathbf{C}_{\mathbf{q}}}_{\substack{\text{3D vector}\\ \text{(RGB, Lab)}}}$$



The bilateral filter is extremely easy to adapt to your need.

Hard to Compute

- Nonlinear $$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

- Complex, spatially varying kernels
  - Cannot be precomputed, no FFT

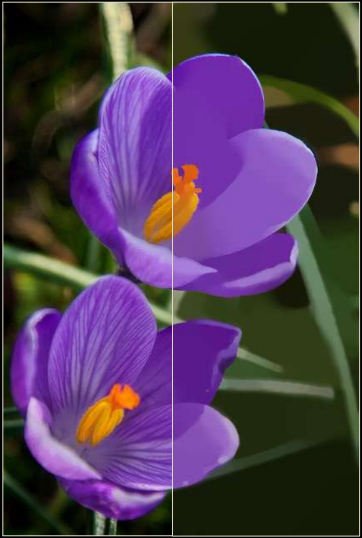
- Brute-force implementation is slow > 10min

Questions ?

Image Smoothing via L0 Gradient Minimization

Li Xu, Cewu Lu, Yi Xu, Jiaya Jia
The Chinese University of Hong Kong

Image Smoothing



- A fundamentally important tool

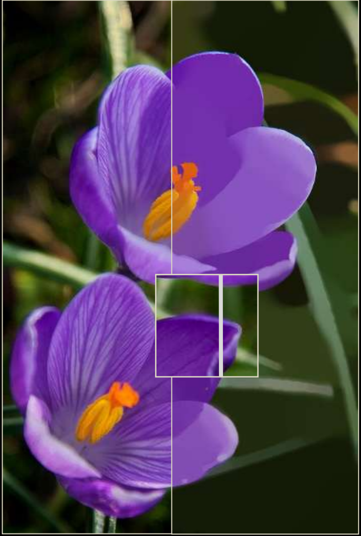
[DeCarlo and Santella 02]

[Fattal et al. 06]

[Kass and Solomon 10]

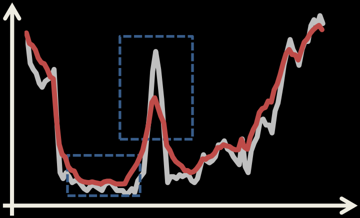
[Farbman et al. 08]

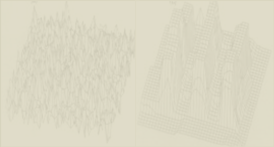
Image Smoothing



General goals:

- Suppress insignificant details
- Maintain major edges





[Rudin et al. 92]

[Tumblin and Turk 99]

[Durand and Dorsey 02]

[Chen et al. 07]

[Farbman et al. 08]

[Subr et al. 09]

[Kass and Solomon 10]

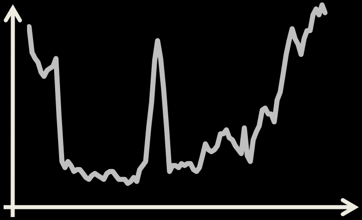
[Baek and Jacobs 10]

[Sylvain et al. 11]

Our New Smoothing Method

A general and effective global smoothing strategy based on a sparsity measure

$$c(f) := \#\{p \mid |\nabla f_p| \neq 0\}$$

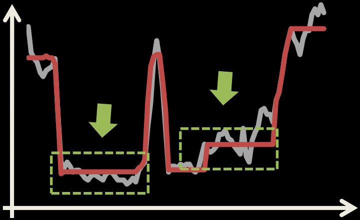


which corresponds to the L0-norm of gradient

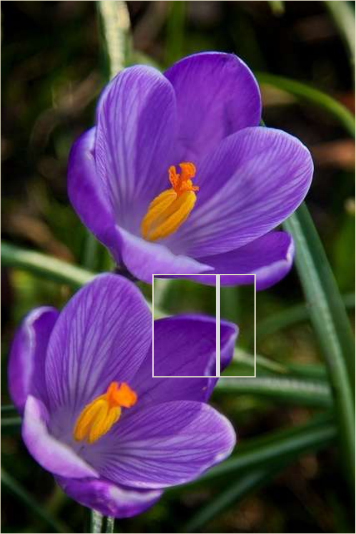
Two Features

1. **Flattening** insignificant details

By removing small non-zero gradients

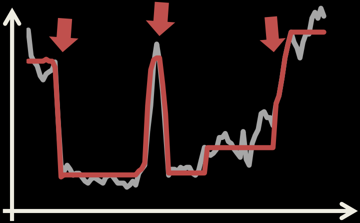


Two Features



2. **Enhancing** prominent edges

Because large gradients receive the same penalty as small ones



$$\#\{p \mid |\nabla f_p| \neq 0\} = \#\{p \mid |\alpha \nabla f_p| \neq 0\}$$

Our Framework in 1D

- Constrain # of non-zero gradients

$$c(f) = \#\{p \mid \left| f_p - f_{p+1} \right| \neq 0\} = k$$

- Make the result similar to the input g

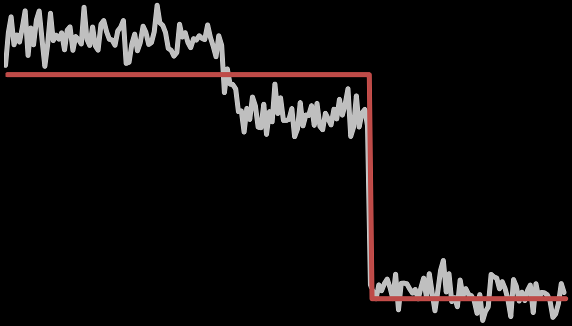
$$\min_f \sum_p (f_p - g_p)^2$$

- Objective function

$$\min_f \sum_p (f_p - g_p)^2 \qquad \text{s.t.} \qquad c(f) = k$$

Our Framework in 1D

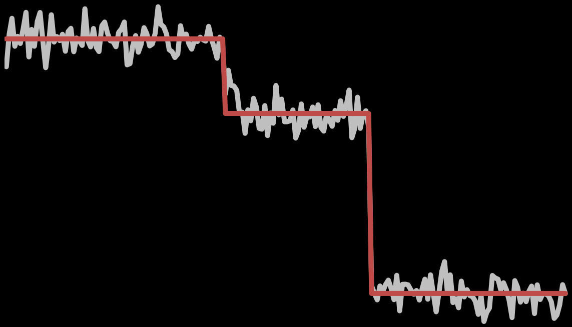
- Input 1D signal g



$$\min_{f} \sum_{p} (f_p - g_p)^2 \qquad \text{s.t.} \qquad c(f) = 1$$

Our Framework in 1D

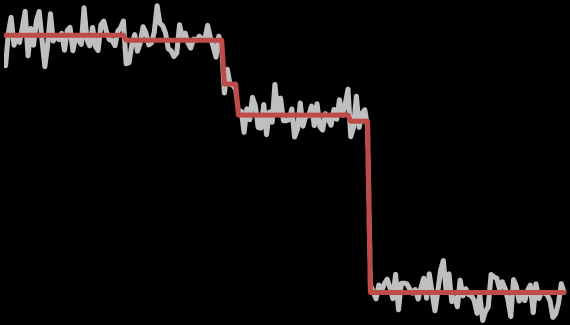
- Input 1D signal g



$$\min_f \sum_p (f_p - g_p)^2 \qquad s.t. \qquad c(f) = 2$$

Our Framework in 1D

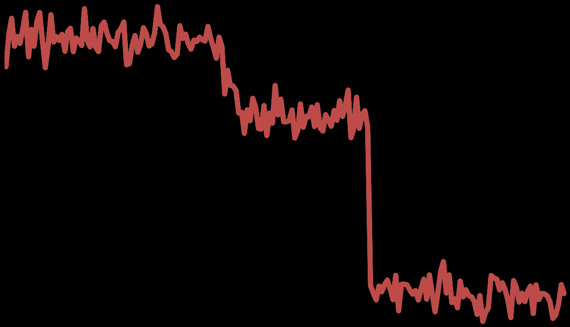
- Input 1D signal g



$$\min_f \sum_p (f_p - g_p)^2 \qquad s.t. \qquad c(f) = 5$$

Our Framework in 1D

- Input 1D signal g



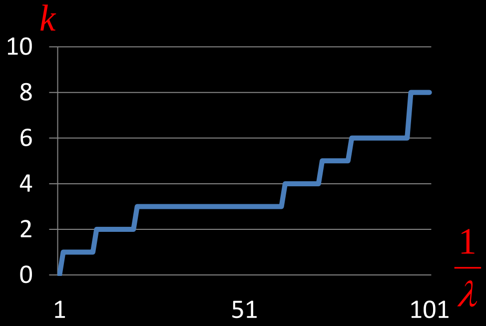
$$\min_{f} \sum_{p} (f_p - g_p)^2 \qquad s.t. \qquad c(f) = 200$$

Transformation

$$\min_{f} \sum_{p} (f_p - g_p)^2 \qquad \text{s.t.} \qquad c(f) = k$$

⬇

$$\min_{f} \sum_{p} (f_p - g_p)^2 + \lambda \cdot c(f)$$



2D Image

$$\min_{f} \sum_{p} (f_p - g_p)^2 + \lambda \cdot c(\partial_x f, \partial_y f)$$

$$c(\partial_x f, \partial_y f) = \#\{p \mid |\partial_x f_p| + |\partial_y f_p| \neq 0\}$$

Finding the global optimum is NP hard

Approximation

$$\min_{f} \sum_{p} (f_p - g_p)^2 + \lambda \cdot c(\quad \boldsymbol{h}, \boldsymbol{v} \quad)$$

$$+\beta \cdot \sum_{p} \left((\partial_x f_p - h_p)^2 + (\partial_y f_p - v_p)^2 \right)$$

Separately estimate f and (h, v)

Iterative Optimization

- Compute f given h, v

$$E(f) = \sum (f_p - g_p)^2 + \beta \cdot \left((\partial_x f_p - h_p)^2 + (\partial_y f_p - v_p)^2 \right)$$

Both the sub-problems are with closed-form solutions

-

$$E(h,v) = \sum_p \left((\partial_x f_p - h_p)^2 + (\partial_y f_p - v_p)^2 \right) + \frac{\lambda}{\beta} c(h,v)$$

- Gradually approximate the original problem

$$\beta \leftarrow 2\beta$$

One Example

Converge in 15 iterations

Iteration #[illegible]

Smoothing Strength

Input

Smoothing Strength

λ=0.01

Smoothing Strength

λ=0.02

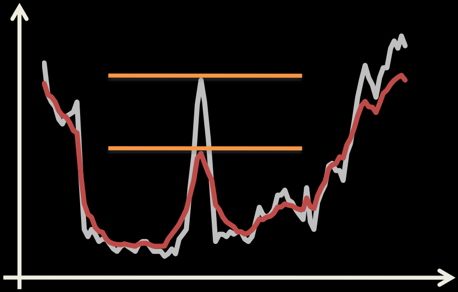
Smoothing Strength

λ=0.03

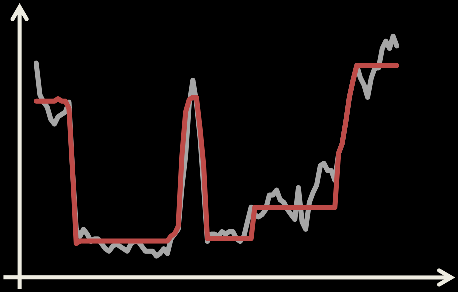
Comparison

Total Variation
Our Result

Comparison

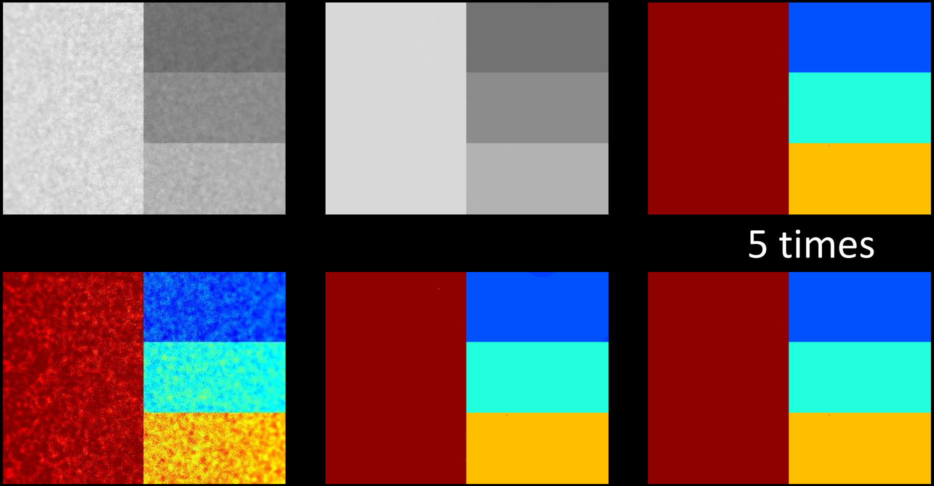


Total Variation



Our Result

Another Example



5 times

Input

1 times

20 times

Applications

Edge Enhancement and Extraction

Edge Enhancement and Extraction

Gradient Map

Edge Enhancement and Extraction

Extracted Edge

Edge Enhancement and Extraction

Smoothing result

Edge Enhancement and Extraction

Extracted Edge

Edge Enhancement and Extraction

Edge Enhancement and Extraction

Edge Enhancement and Extraction

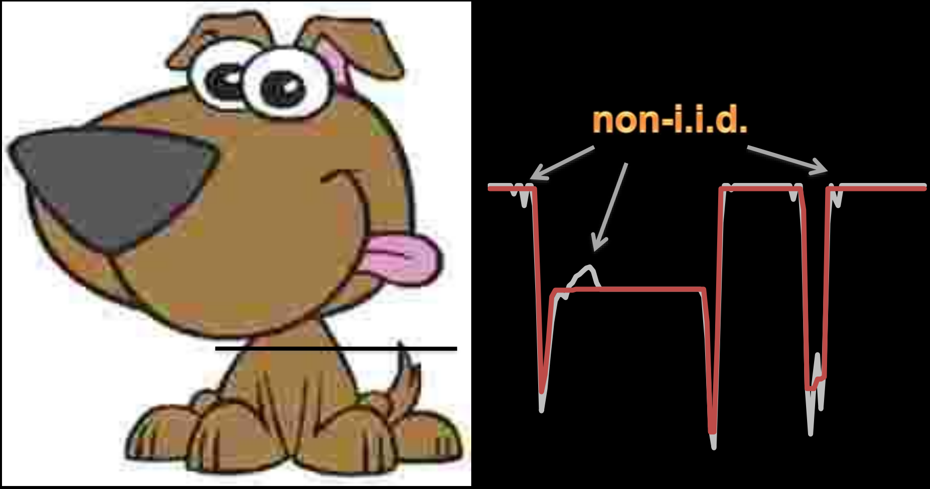
Edge Enhancement and Extraction

Without smoothing

With smoothing

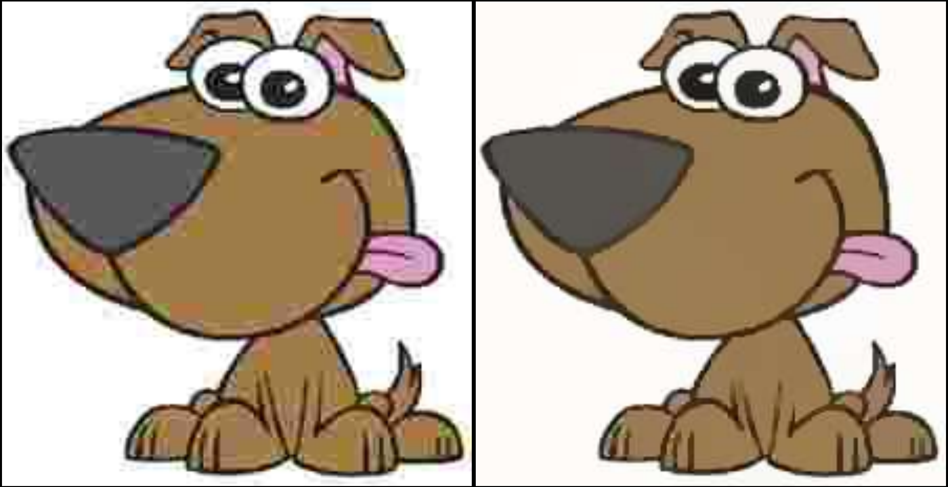
Clip-Art JPEG Artifact Removal

Clip-Art JPEG Artifact Removal



non-i.i.d.

Clip-Art JPEG Artifact Removal



Clip-Art JPEG Artifact Removal

Image Abstraction

Image Abstraction

Pencil Sketch

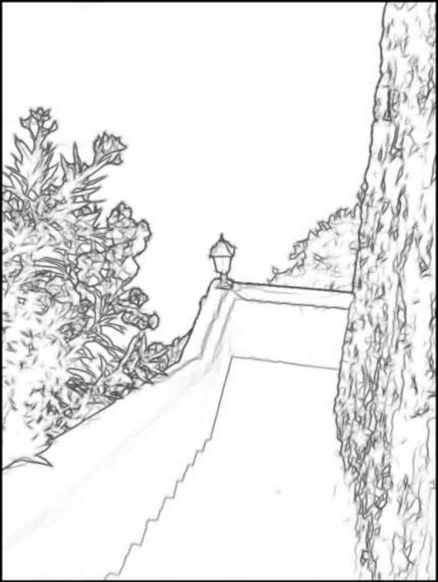
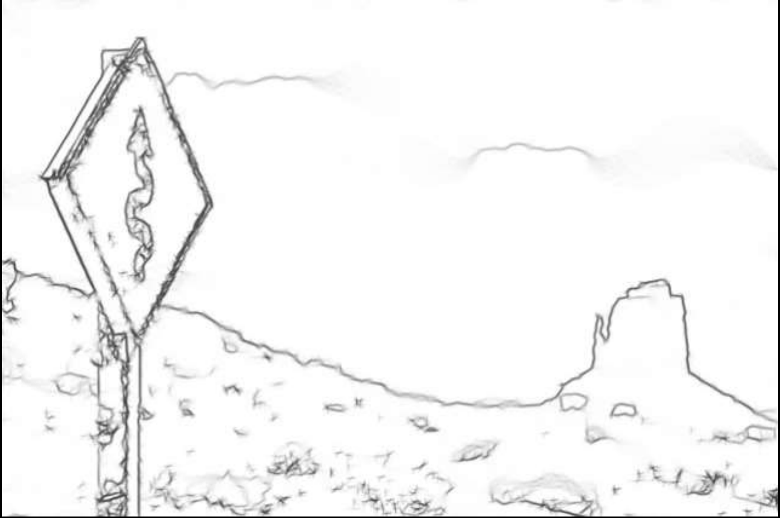


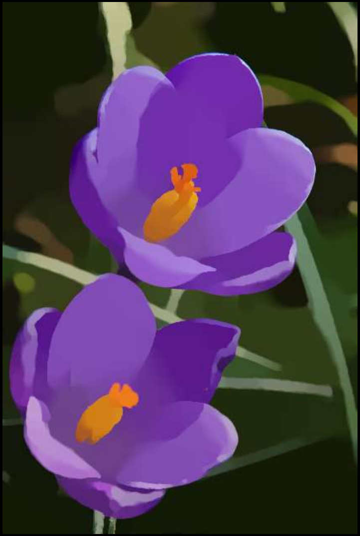
Image Abstraction

Image Abstraction

Pencil Sketch

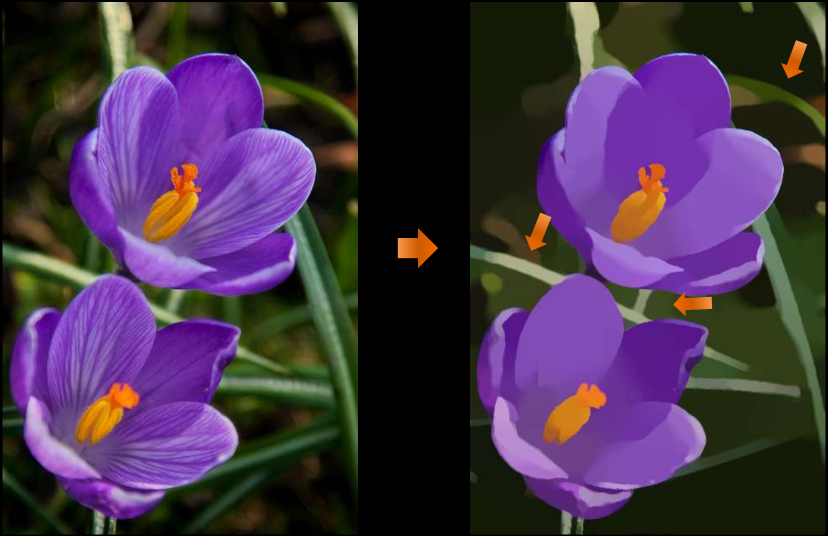


Detail Manipulation



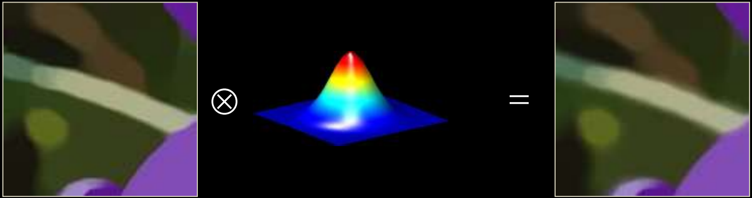
Base layer

Detail Manipulation



Base layer

Edge Adjustment



Spatially varying Gaussian blur in an optimization procedure

Edge Adjustment

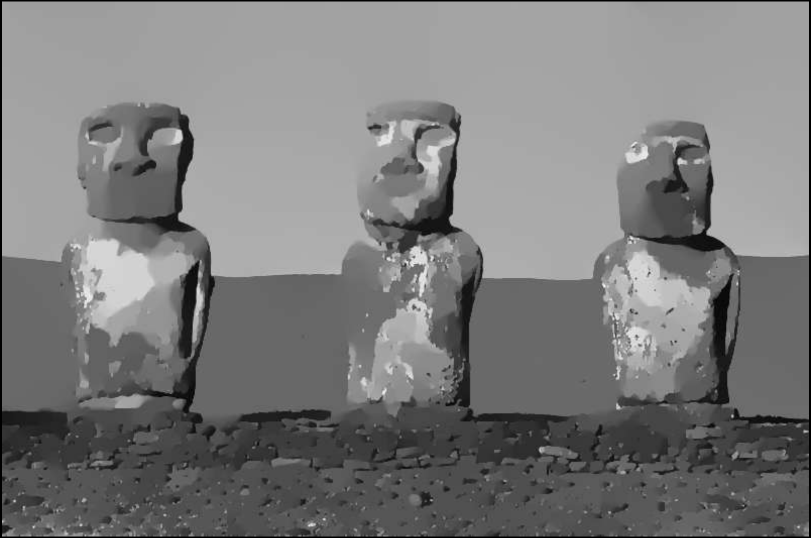
Input

Detail Boosting

Detail Manipulation

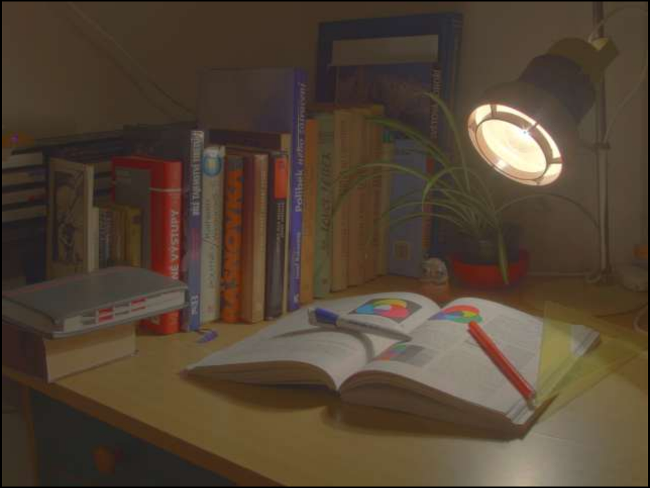
Image of [Farbman et al. 08]

Detail Manipulation



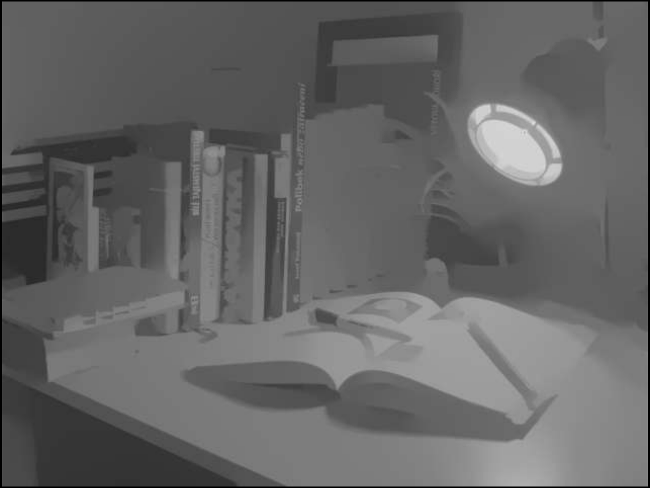
Detail Manipulation

HDR Tone Mapping



HDR Input (gamma adjusted)

HDR Tone Mapping

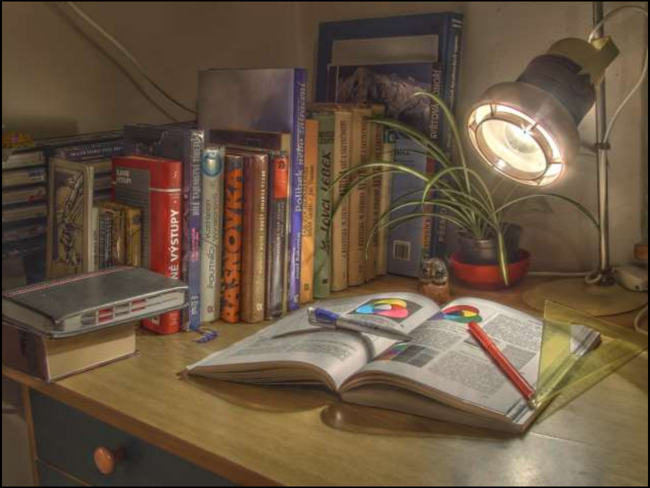


Log-base layer to be compressed

HDR Tone Mapping

Detail layer

HDR Tone Mapping



HDR Tone Mapping

HDR Input (gamma adjusted)

HDR Tone Mapping

HDR Tone Mapping

HDR Tone Mapping

Combined with other smoothing

Strong texture will be preserved

Combined with other smoothing

Our smoothing result

Combined with other smoothing

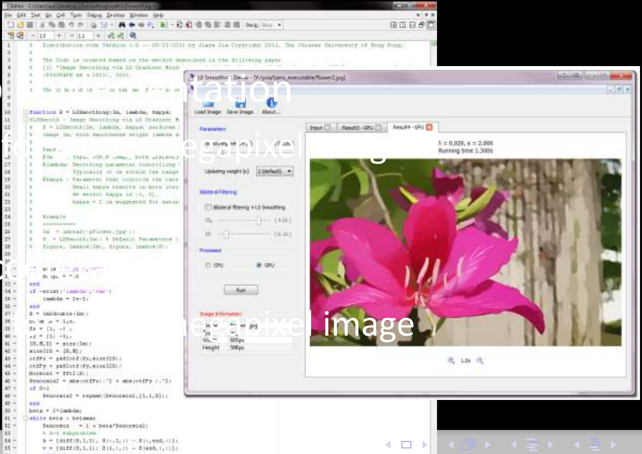
Bilateral Filter

Combined with other smoothing

Bilateral Filter + Ours

Implementation

- Matlab source code and Windows software are available
- Matlab implementation
  - 12s for a one megapixel image
- GPU acceleration
  - 0.4s for a one megapixel image


λ = 0.020, κ = 2.000
Running time 1.300s

Conclusion

- A simple and general smoothing framework
  - Approximate L0-norm gradient measure
  - Flatten low-amplitude details
  - Enhance prominent structures
- Possible extensions in graphics and vision
  - Video
  - 3D surface (modeling)
  - Depth

We wish to thank

- *Michael S. Brown* for narrating the video.
- The *anonymous reviewers* for constructive comments.
- Flicker Users: John McCormick, conner395, cyber-seb, T-KONI, Remi Longva, dms_a_jem for allowing us to use their pictures.

The End

中国科学技术大学
University of Science and Technology of China

Image Stitching

张举勇
中国科学技术大学

Overview

- Image stitching is to combine multiple photos to create a larger photo.
- This technology is now widely available. It’s on pretty much all smart phones that are in market today.
- It’s also used in other domains, such as medical imaging, and remote sensing.

Image Stitching

Image 1

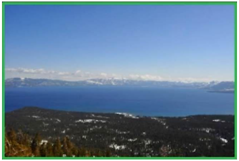


Image 2

Image 3

How would you align these images?

Image Stitching

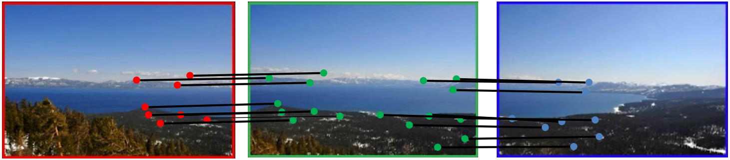


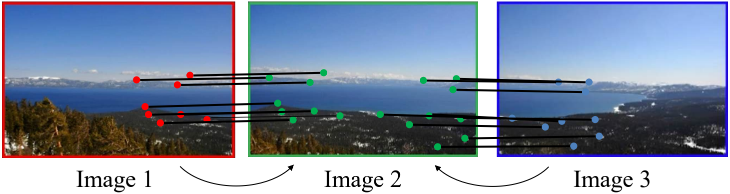
Image 1

Image 2

Image 3

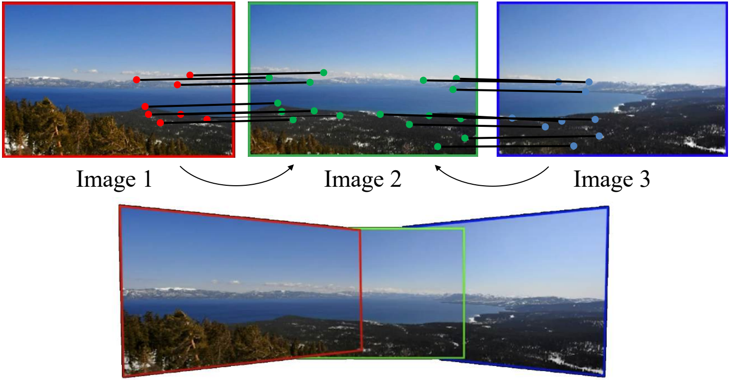
Find corresponding points
(using feature detectors like SIFT)

Image Stitching



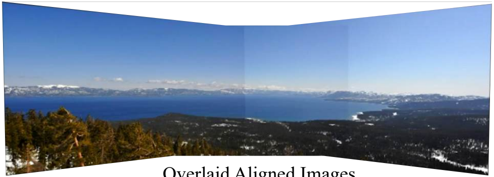
Find geometric relationship between the images

Image Stitching



Warp images so that corresponding points align

Image Stitching



Overlaid Aligned Images

Blended Image

Blend images to remove hard seams

Image Stitching

Topics:

- 2x2 Image Transformations
- 3x3 Image Transformations
- Computing Homography
- Dealing with Outliers:RANSAC
- Warping and Blending Images

2x2 Image Transformations

Image Filtering: Change range (brightness)

$$g(x,y) = T_r(f(x,y))$$

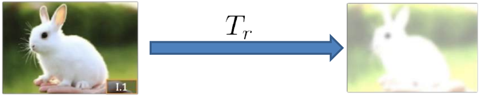
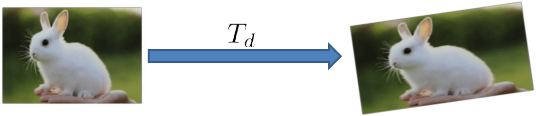


Image Warping: Change domain (location)

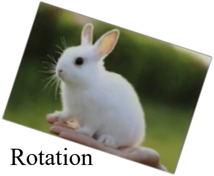
$$g(x,y) = f(T_d(x,y))$$

Transformation T_d is a coordinate changing operator



Global Warping/Transformation


Translation


Rotation


Scaling and Aspect

$$g(x, y) = f(T(x, y))$$

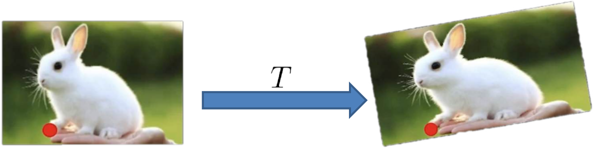

Affine


Projective


Barrel

Transformation T is the same over entire domain
often can be described by just a few parameters

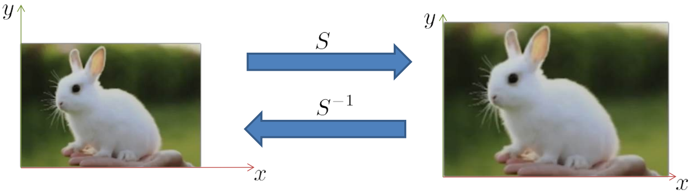
2x2 Linear Transformations



T can be represented by a matrix.

$$\mathbf{p}_2 = T\mathbf{p}_1 \qquad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \qquad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Scaling(Stretching and Squishing)



Forward:

$$x_2 = ax_1 \qquad y_2 = by_1$$

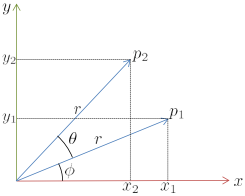
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Inverse:

$$x_1 = \frac{1}{a} x_2 \qquad y_1 = \frac{1}{b} y_2$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = S^{-1} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/b \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

2D Rotation



$$x_1 = r\cos(\phi)$$

$$y_1 = r\sin(\phi)$$

$$x_2 = r\cos(\phi + \theta)$$

$$x_2 = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$$

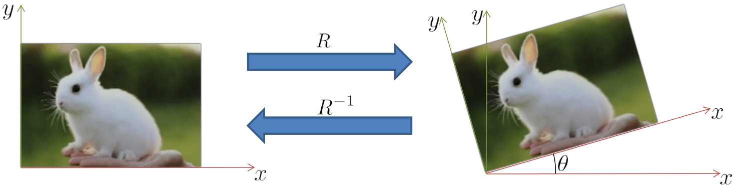
$$x_2 = x_1\cos(\theta) - y_1\sin(\theta)$$

$$y_2 = r\sin(\phi + \theta)$$

$$y_2 = r\cos(\phi)\sin(\theta) + r\sin(\phi)\cos(\theta)$$

$$y_2 = x_1\sin(\theta) + y_1\cos(\theta)$$

Rotation



Forward:

$$x_2 = x_1 \cos(\theta) - y_1 \sin(\theta)$$

$$y_2 = x_1 \sin(\theta) + y_1 \cos(\theta)$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = R \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

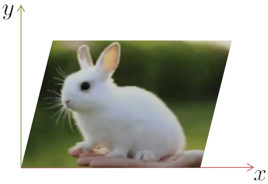
Inverse:

$$x_1 = x_2 \cos(\theta) + y_2 \sin(\theta)$$

$$y_1 = -x_2 \sin(\theta) + y_2 \cos(\theta)$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = R^{-1} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

Skew

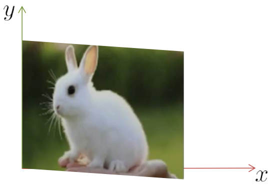


Horizontal Skew:

$$x_2 = x_1 + m_x y_1$$

$$y_2 = y_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_x \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & m_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$



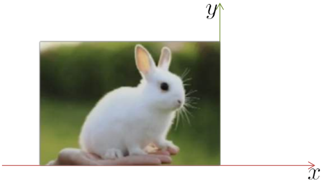
Vertical Skew:

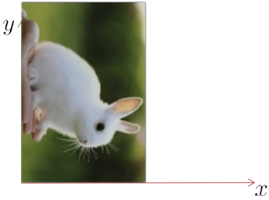
$$x_2 = x_1$$

$$y_2 = m_y x_1 + y_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_y \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ m_y & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Mirrow





Mirrow about Y-axis:

$$x_2 = -x_1$$

$$y_2 = y_1$$

$$M_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Mirrow about line $y = x$:

$$x_2 = y_1$$

$$y_2 = x_1$$

$$M_{xy} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

2x2 Matrix Transformations

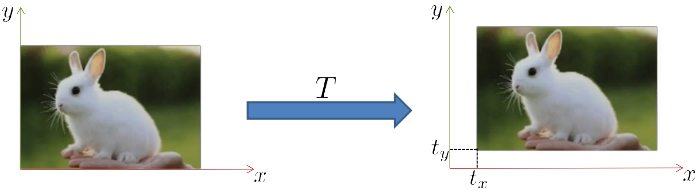
Any transformation of the form:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

- Origin maps to the origin
- Lines map to lines
- Parallel lines remain parallel
- Closed under composition

$$\left.\begin{aligned} \mathbf{p}_2 &= T_{21}\mathbf{p}_1 \\ \mathbf{p}_3 &= T_{32}\mathbf{p}_2 \\ \mathbf{p}_3 &= T_{31}\mathbf{p}_1 \end{aligned}\right\} \mathbf{p}_3 = T_{32}\mathbf{p}_2 = T_{32}T_{21}\mathbf{p}_1 \Rightarrow T_{31} = T_{32}T_{21}$$

Translation



$x_2 = x_1 + t_x$ $\qquad$ $y_2 = y_1 + t_y$

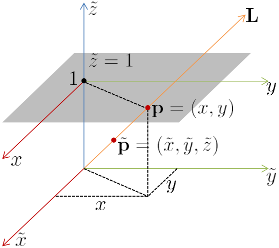
Can translation be expressed as a 2x2 matrix? No

Homogenous Coordinates

The homogenous representation of a 2D point $\mathbf{p} = (x, y)$ is a 3D point $\tilde{\mathbf{p}} = (\tilde{x}, \tilde{y}, \tilde{z})$. The third coordinate $\tilde{z} \neq 0$ is fictitious such that:

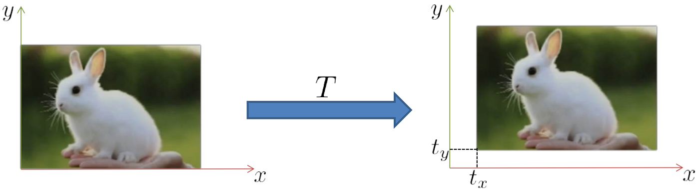
$$x = \frac{\tilde{x}}{\tilde{z}} \qquad y = \frac{\tilde{y}}{\tilde{z}}$$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{z}x \\ \tilde{z}y \\ \tilde{z} \end{bmatrix} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \tilde{\mathbf{p}}$$



Every point on line **L**(except origin) represent the homogenous coordinate of $\mathbf{p}(x, y)$

Translation



$$x_2 = x_1 + t_x \qquad y_2 = y_1 + t_y$$

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x_2} \\ \tilde{y_2} \\ \tilde{z_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Scaling, Rotation, Skew, Translation

$$\begin{bmatrix}\tilde{x_2}\\ \tilde{y_2}\\ \tilde{z_2}\end{bmatrix}=\begin{bmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

Scaling

$$\begin{bmatrix}\tilde{x_2}\\ \tilde{y_2}\\ \tilde{z_2}\end{bmatrix}=\begin{bmatrix}1 & m_x & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

Skew

$$\begin{bmatrix}\tilde{x_2}\\ \tilde{y_2}\\ \tilde{z_2}\end{bmatrix}=\begin{bmatrix}1 & 0 & t_x\\ 0 & 1 & t_y\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

Translation

$$\begin{bmatrix}\tilde{x_2}\\ \tilde{y_2}\\ \tilde{z_2}\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

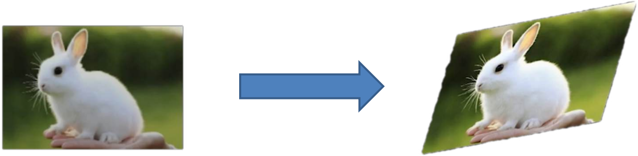
Rotation

Composition of these transformations?

Affine Transformation

Any transformation of the form:

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x_2} \\ \tilde{y_2} \\ \tilde{z_2} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{x_1} \\ \tilde{y_1} \\ \tilde{z_1} \end{bmatrix}$$



Affine Transformation

Any transformation of the form:

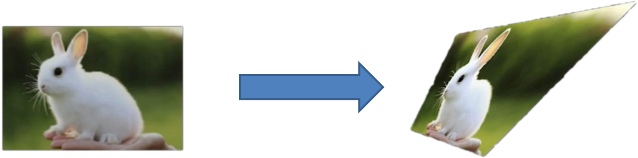
$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

- Origin does not necessarily map to the origin
- Lines map to lines
- Parallel lines remain parallel
- Closed under composition

Projective Transformation

Any transformation of the form:

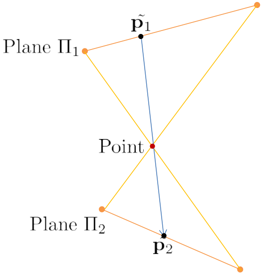
$$\begin{bmatrix} \tilde{x_2} \\ \tilde{y_2} \\ \tilde{z_2} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x_1} \\ \tilde{y_1} \\ \tilde{z_1} \end{bmatrix} \qquad \tilde{\mathbf{p}_2} = H\tilde{\mathbf{p}_1}$$



Also called Homography

Projective Transformation

Mapping of one plane to another through a point



$$\tilde{\mathbf{p}_2} = H\tilde{\mathbf{p}_1}$$

$$\begin{bmatrix} \tilde{x_2} \\ \tilde{y_2} \\ \tilde{z_2} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x_1} \\ \tilde{y_1} \\ \tilde{z_1} \end{bmatrix}$$

Same as imaging a plane through a pinhole

Projective Transformation

Homography can only be defined up to a scale.

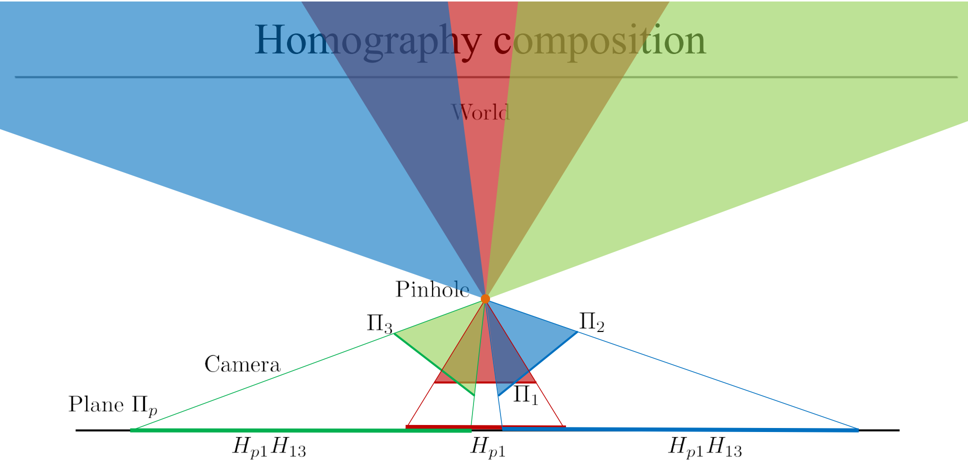
$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix} = \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = k \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

If we fix scale such that $\sqrt{\sum (h_{ij})^2}$ then 8 free parameters

- Origin does not necessarily map to the origin
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Closed under composition

Remember Vanishing Points?

Homography composition



Useful in stitching planar panoramas

Computing Homography

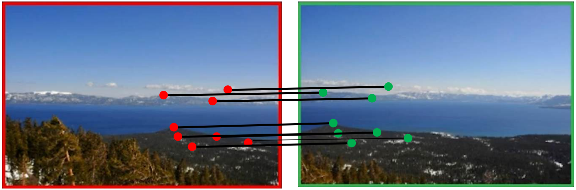
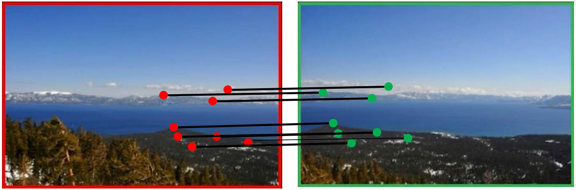


Image 1

Image 2

Given a set of matching features/points between image images 1 and 2, find the homography *H* that best "agrees" with the matches.

Computing Homography



$$\begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x}_d \\ \tilde{y}_d \\ \tilde{z}_d \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix}$$

How many unknows? 9 ...But 8 degrees of freedom

How many minimum pairs of matching points? 4

Computing Homography

For a given pair *i* of corresponding points:

$$x_d^{(i)} = \frac{\tilde{x}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}}$$

$$y_d^{(i)} = \frac{\tilde{y}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}}$$

Rearranging the terms:

$$x_d^{(i)}(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}$$

$$y_d^{(i)}(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}$$

Computing Homography

$$x_d^{(i)}(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}$$

$$y_d^{(i)}(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}$$

Rearranging the terms and writing as linear equation:

$$\underset{\text{(Known)}}{\boxed{\begin{bmatrix} x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \end{bmatrix}}} \underset{\substack{\mathbf{h} \\ \text{(Unknown)}}}{\boxed{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Computing Homography

Combining the equation for all corresponding points:

$$\underset{\substack{\boldsymbol{A}\\ \text{(Known)}}}{\boxed{\begin{bmatrix} x_s^{(1)} & y_s^{(1)} & 1 & 0 & 0 & 0 & -x_d^{(1)}x_s^{(1)} & -x_d^{(1)}y_s^{(1)} & -x_d^{(1)} \\ 0 & 0 & 0 & x_s^{(1)} & y_s^{(1)} & 1 & -y_d^{(1)}x_s^{(1)} & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)} \\ & & & & & \vdots & & & \\ x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \\ & & & & & \vdots & & & \\ x_s^{(n)} & y_s^{(n)} & 1 & 0 & 0 & 0 & -x_d^{(n)}x_s^{(n)} & -x_d^{(n)}y_s^{(n)} & -x_d^{(n)} \\ 0 & 0 & 0 & x_s^{(n)} & y_s^{(n)} & 1 & -y_d^{(n)}x_s^{(n)} & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)} \end{bmatrix}}} \underset{\substack{\mathbf{h}\\ \text{(Unknown)}}}{\boxed{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\text{Solve for } \mathbf{h}: \quad \boxed{\boldsymbol{A}\mathbf{h} = 0} \quad \text{such that} \quad \|\mathbf{h}\|^2 = 1$$

Constrained Least Squares

Solve for $\mathbf{h}$: $\boxed{\boldsymbol{A}\mathbf{h} = 0}$ such that $\|\mathbf{h}\|^2 = 1$

Define least squares problem:

$$\min_{\mathbf{h}} \|\boldsymbol{A}\mathbf{h}\|^2 \text{ such that } \|\mathbf{h}\|^2 = 1$$

We know that:

$$\|\boldsymbol{A}\mathbf{h}\|^2 = (\boldsymbol{A}\mathbf{h})^T \boldsymbol{A}\mathbf{h} = \mathbf{h}^T \boldsymbol{A}^T \boldsymbol{A}\mathbf{h} \quad \text{and} \quad \|\mathbf{h}\|^2 = \mathbf{h}^T \mathbf{h} = 1$$

$$\boxed{\min_{\mathbf{h}}(\mathbf{h}^T \boldsymbol{A}^T \boldsymbol{A}\mathbf{h}) \text{ such that } \|\mathbf{h}\|^2 = 1}$$

Constrained Least Squares

$$\min_{\mathbf{h}}(\mathbf{h}^T \boldsymbol{A}^T \boldsymbol{A}\mathbf{h}) \text{ such that } \|\mathbf{h}\|^2 = 1$$

Define Loss function $L(\mathbf{h}, \lambda)$:

$$L(\mathbf{h}, \lambda) = \mathbf{h}^T \boldsymbol{A}^T \boldsymbol{A}\mathbf{h} - \lambda(\mathbf{h}^T \mathbf{h} - 1)$$

Taking derivatives of $L(\mathbf{h}, \lambda)$ w.r.t $\mathbf{h}$: $2\boldsymbol{A}^T \boldsymbol{A}\mathbf{h} - 2\lambda\mathbf{h} = 0$

$$\boxed{\boldsymbol{A}^T \boldsymbol{A}\mathbf{h} = \lambda\mathbf{h}}$$ Eigenvalue Problem

Eigenvector $\mathbf{h}$ with smallest eigenvalue λ of matrix $\boldsymbol{A}^T \boldsymbol{A}$ minimizes the loss function $L(\mathbf{h})$.

Matlab: eig(A’*A) returns eigenvalues and vectors of $\boldsymbol{A}^T \boldsymbol{A}$

What could go wrong?

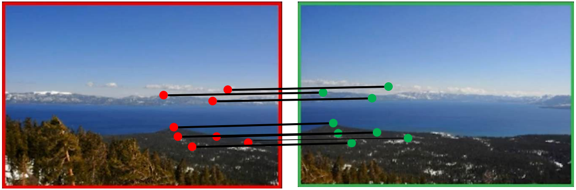


Image 1

Image 2

What could go wrong?

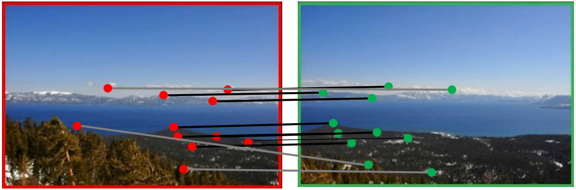


Image 1

Image 2

Outliers!

We need to robustly compute transformation in the presence of wrong matches.

If number of outliers < 50%, then RANSAC to the rescue!

RANdom SAmple Consensus

General RANSAC algorithm:

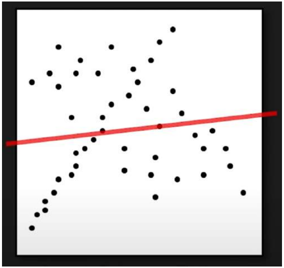
1. Randomly choose s samples. Typically s is the minimum samples to fit the model.
2. Fit the model to the randomly chosen samples.
3. Count the number M of data points(inliers) that fit the model within a measure of error ε.
4. Repeat Steps 1-3 N times
5. Choose the model that has the largest number M of inliers.

For homography:

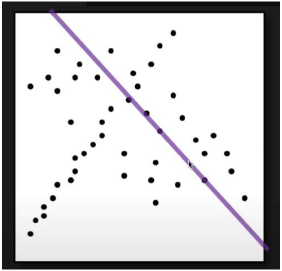
$s = 4$ points. ε is acceptable alignment error in pixels.

RANSAC Example: Line Fitting

Robust line fitting:



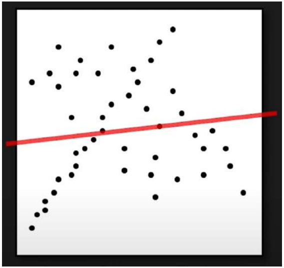
Least Squares Fitting
Inliers:2



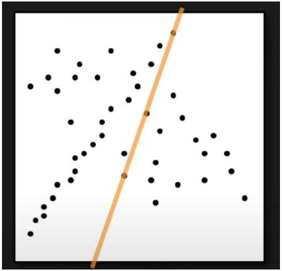
RANSAC Interation 1
Inliers:4

RANSAC Example: Line Fitting

Robust line fitting:



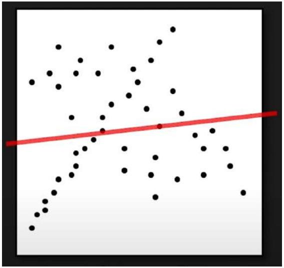
Least Squares Fitting
Inliers:2



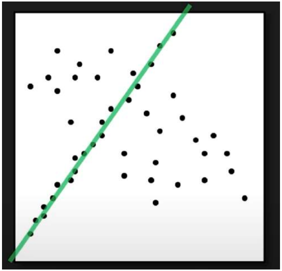
RANSAC Interation 2
Inliers:3

RANSAC Example: Line Fitting

Robust line fitting:



Least Squares Fitting
Inliers:2

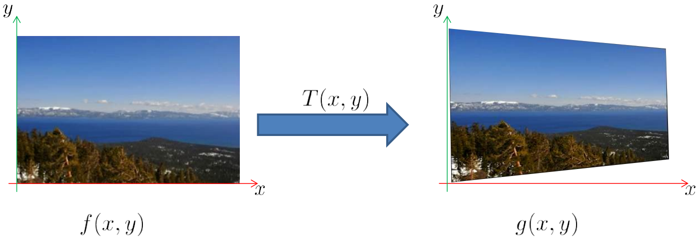


RANSAC Interation i
Inliers:20

Warping Images

Given a transformation *T* and a image *f*(*x*,*y*), compute the transformed image *g*(*x*,*y*)

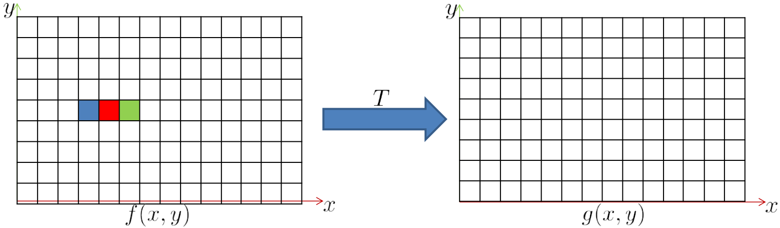
$$\boxed{g(x,y) = f(T(x,y))}$$



Forward Warping

Send each pixel (x,y) in $f(x,y)$ to its corresponding location $T(x,y)$ in $g(x,y)$

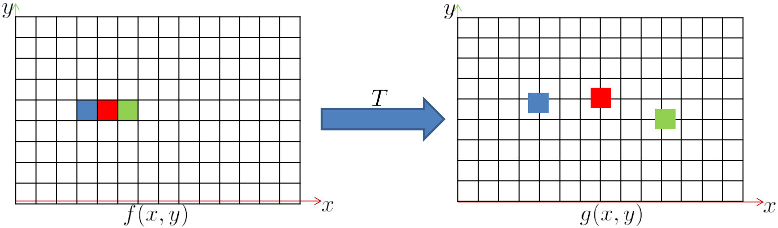
$$\boxed{g(x,y) = f(T(x,y))}$$



Forward Warping

Send each pixel (*x*,*y*) in *f*(*x*,*y*) to its corresponding location *T*(*x*,*y*) in *g*(*x*,*y*)

$$\boxed{g(x,y) = f(T(x,y))}$$

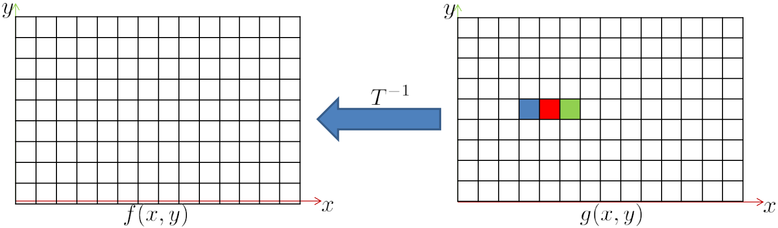


What if pixel lands in between pixels?
What if not all pixels in *g*(*x*,*y*) are filled?
Can result in holes!

Backward Warping

Get each pixel (*x*,*y*) in *g*(*x*,*y*) from its corresponding location $T^{-1}(x,y)$ in *f*(*x*,*y*)

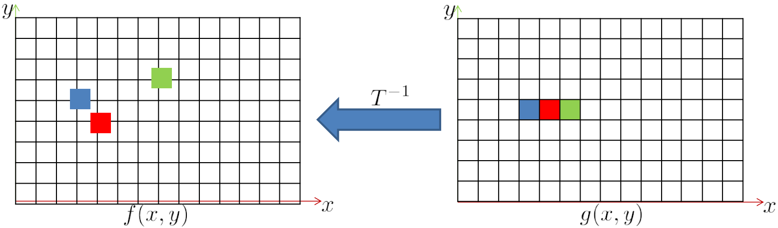
$$\boxed{g(x,y) = f(T(x,y))}$$



Backward Warping

Get each pixel (*x*,*y*) in *g*(*x*,*y*) from its corresponding location $T^{-1}(x,y)$ in *f*(*x*,*y*)

$$\boxed{g(x,y) = f(T(x,y))}$$



What if pixel lands between pixels?
Use Nearest Neighbor or Interpolate

Image Alignment Process

Image 1

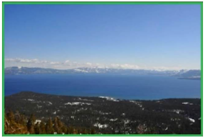
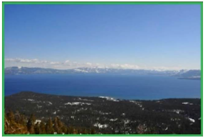


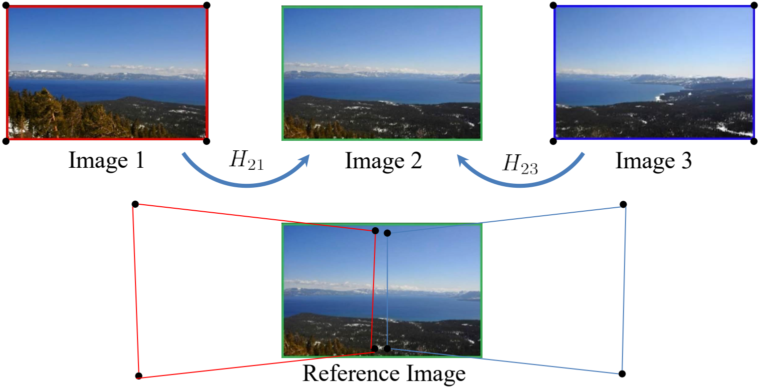
Image 2

Image 3



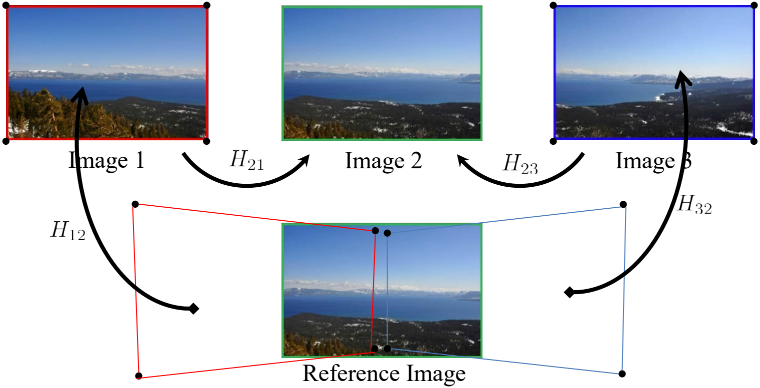
Reference Image
(Image 2)

Image Alignment Process



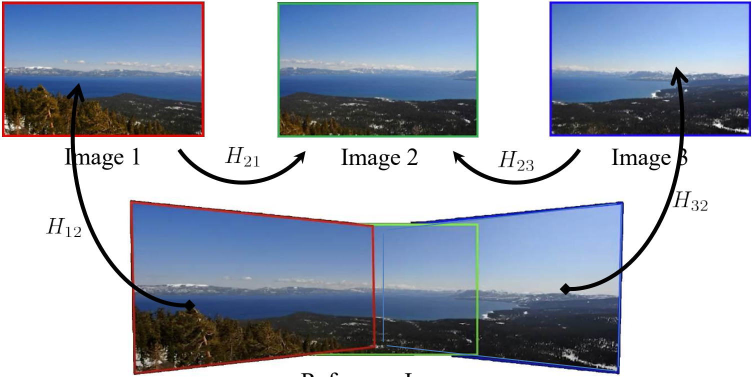
Compute the bounds of Image 1 and Image 3 in reference image space

Image Alignment Process



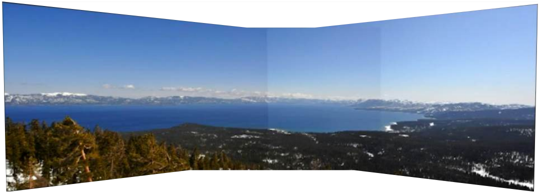
For each pixel within bounds, compute its location in captured image

Image Alignment Process



For each pixel within bounds, compute its location in captured image

Blending Images



Overlaid Aligned Images

Hard seams due to vignetting, exposure differences, etc.

Blending Images: Averaging

Averaged Images

Seams still visible

Blending Images

Say we want to blend images I_1 and I_2 at the center

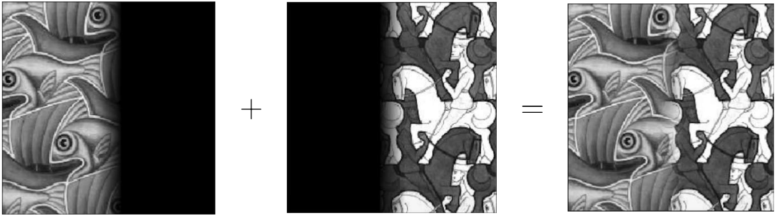
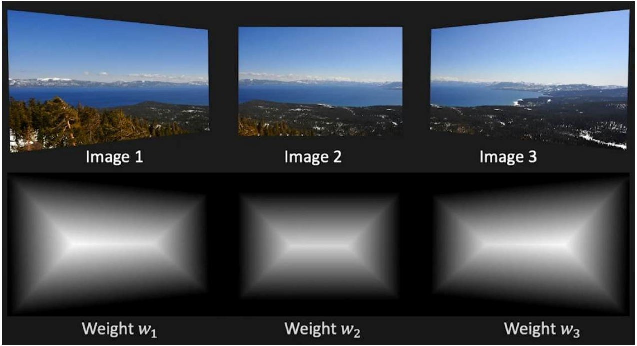


Image I_1 Image I_2 Blended Image I_{blend}

Weight w_1 Weight w_2

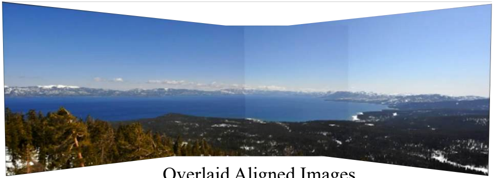
$$I_{blend} = \frac{w_1 I_1 + w_2 I_2}{w_1 + w_2}$$

Computing Weighting Functions



Pixels closer to the edge get a lower weight.
Ex: Distance Transform (bwdist in MATLAB)

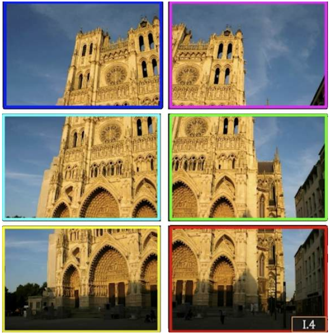
Weighted Blending



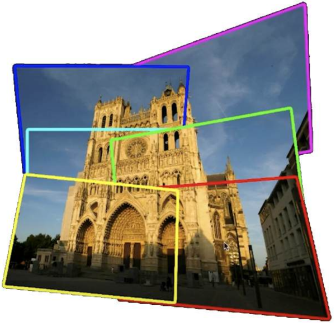
Overlaid Aligned Images

Blended Image

Image Stitching Example

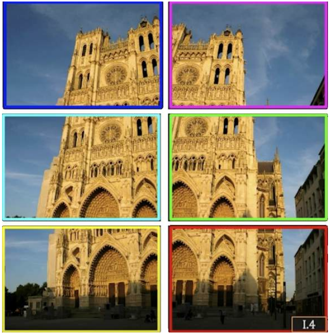


Source Images



Aligned Images

Image Stitching Example



Source Images

Blended Images

Face Detection

张举勇
中国科学技术大学

What is Face Detection?

Locate human faces in images

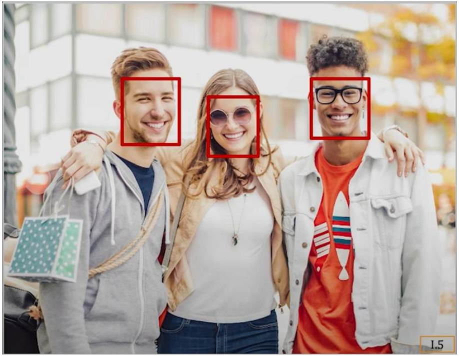


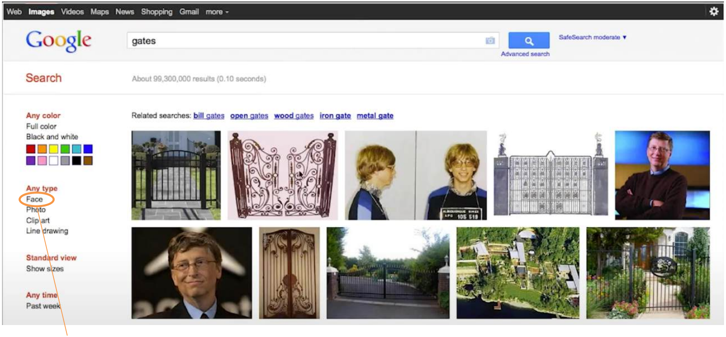
Image Stitching

Locate human faces in images.

Topics:

- Uses of Face Detection
- Haar Features for Face Detection
- Integral Image
- Nearest Neighbor Classifier
- Support Vector Machine

Where is Face Detection Used?



gates

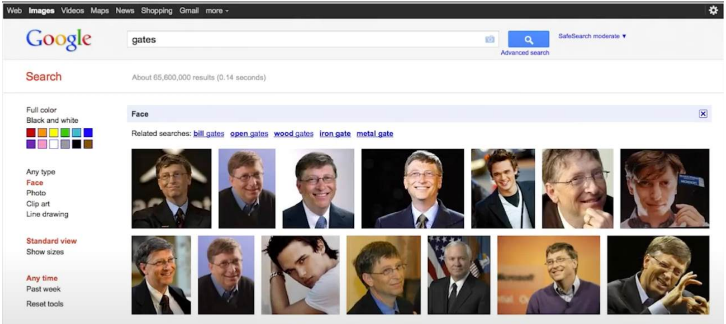
About 99,300,000 results (0.10 seconds)

Related searches: bill gates open gates wood gates iron gate metal gate

Face Detection

Finding People using Search Engines

Where is Face Detection Used?



gates

About 65,600,000 results (0.14 seconds)

Face

Related searches: bill gates open gates wood gates iron gate metal gate

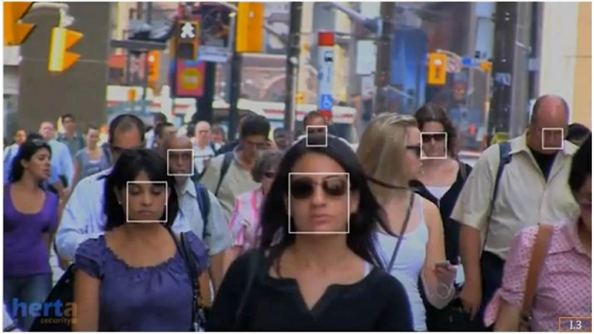
Only faces of people named "Gates"

Finding People using Search Engines

Where is Face Detection Used?

Intelligent Marketing

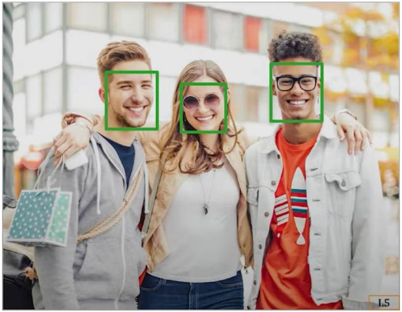
Where is Face Detection Used?



Biometrics, Surveillance, Monitoring

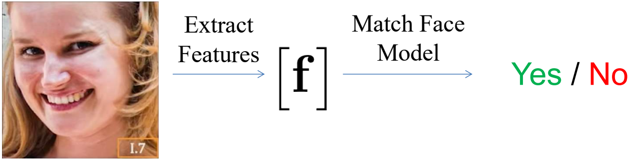
Face Detection in Computers

Slide windows of different sizes across image.
At each location match window to face model.



Face Detection Framework

For each window:



Extract Features

$[\mathbf{f}]$

Match Face Model

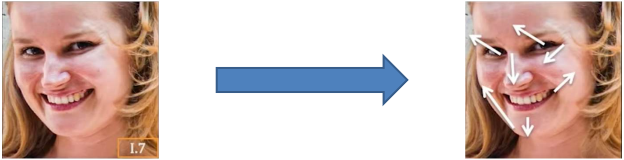
Yes / No

Features: Which features represent faces well?

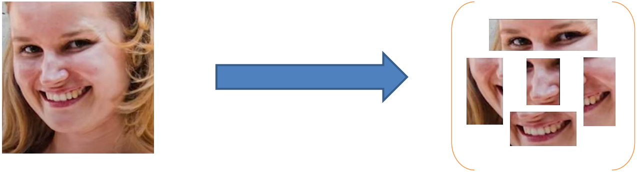
Classifier: How to construct a face model and efficiently classify features as face or not?

What are Good Features?

Interest Points (Edges, Corners, SIFT)?



Facial Components (Templates)?



Charateristics of Good Features

Discriminate Face/Non-Face

$\neq$

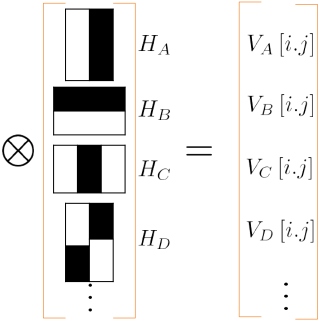
Extremely Fast to Compute

Need to evaluate millions of windows in an image

Haar Features

Set of Correlation Responses to Haar Filters

Input Image



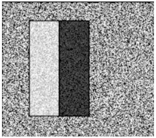
Haar Filters

Haar Features

$\mathbf{f}[i,j]$

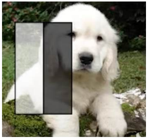
Discriminative Ability of Haar Feature

$V_A = 64$



$V_A \approx 0$

$V_A = 16$

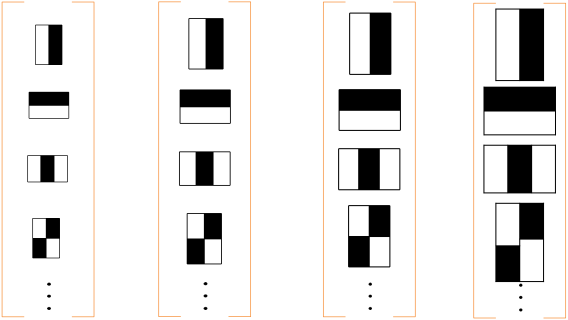


$V_A = -127$

Haar Features are Sensitive to Directionality of Patterns

Haar Features

Compute Haar Features at different scales to detect faces of different sizes.



Haar Features

$\otimes$

$\text{White} = 1, \text{Black} = -1$

Response to Filter H_A at location (i, j) :

$$V_A[i,j] = \sum_m \sum_n I[m-i, n-j] \, H_A[m,n]$$

$$V_A[i,j] = \sum (\text{pixel intensities in white area}) - \sum (\text{pixels intensities in black area})$$

Haar Features: Computation Cost

$$Value = \sum(pixel\ intensities\ in\ white\ area) - \sum(pixels\ intensities\ in\ black\ area)$$

Computation cost = $(N \times M - 1)$ additions per pixel per filter per scale.

Can We Do Better?

Integral Image

A table that holds the sum of all pixel values to the left and top of a given pixel, inclusive.

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

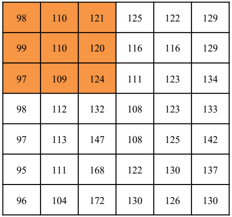
Image I

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

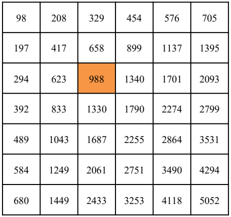
Integral Image

A table that holds the sum of all pixel values to the left and top of a given pixel, inclusive.



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I

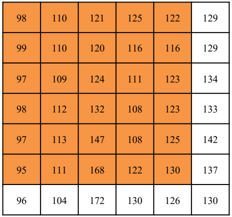


98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

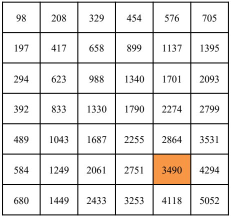
Integral Image

A table that holds the sum of all pixel values
to the left and top of a given pixel, inclusive.



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I

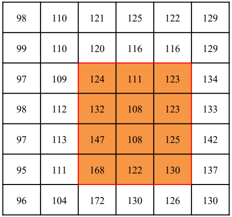


98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

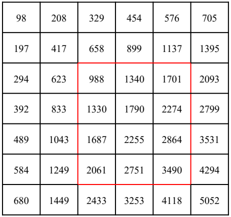
Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I

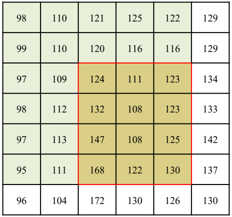


98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

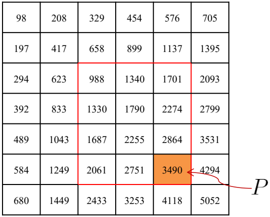
Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I



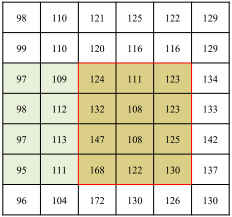
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

$$\begin{aligned} Sum &= II_P + \dots \\ &= 3490 + \dots \end{aligned}$$

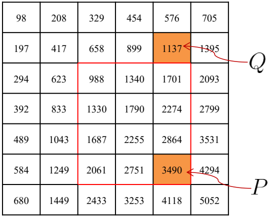
Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I



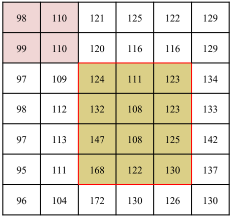
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

$$\begin{aligned} Sum &= II_P - II_Q + \dots \\ &= 3490 - 1137 + \dots \end{aligned}$$

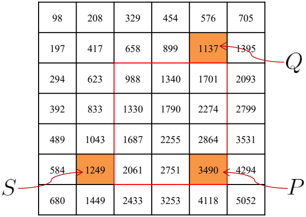
Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I



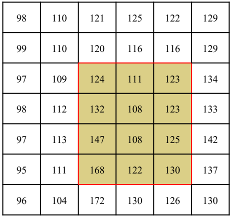
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

$$\begin{aligned} Sum &= II_P - II_Q - II_S + \dots \\ &= 3490 - 1137 - 1249 + \dots \end{aligned}$$

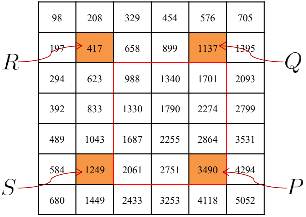
Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I



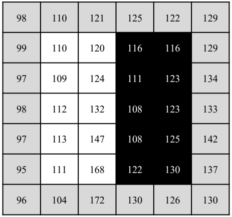
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

$$\begin{aligned} Sum &= II_P - II_Q - II_S + II_R \\ &= 3490 - 1137 - 1249 + 417 = 1521 \end{aligned}$$

Computation Cost: Only 3 additions

Haar Response using Integral Image



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

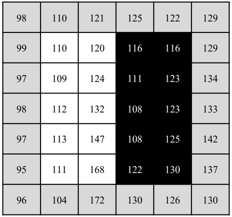
Image I

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

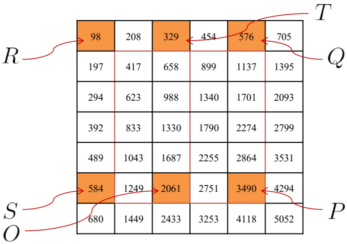
$$V_A = \sum(pixels\ in\ white) - \sum(pixels\ in\ black)$$

Haar Response using Integral Image



98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I



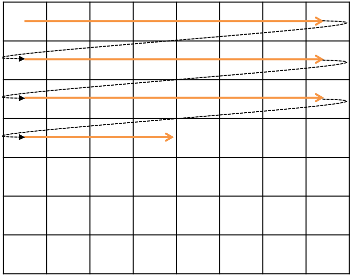
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

$$\begin{aligned} V_A &= \sum(\textit{pixel intensities in white}) - \sum(\textit{pixel intensities in black}) \\ &= (II_O - II_T + II_R - II_S) - (II_P - II_Q + II_T - II_O) \\ &= (2061 - 329 + 98 - 584) - (3490 - 576 + 329 - 2061) = 64 \end{aligned}$$

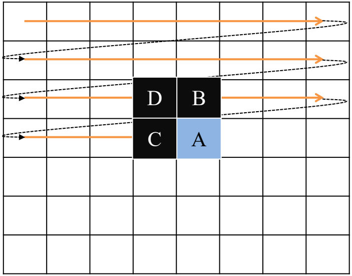
Computation Cost: Only 7 additions

Computing Integral Image



Raster Scanning

Computing Integral Image



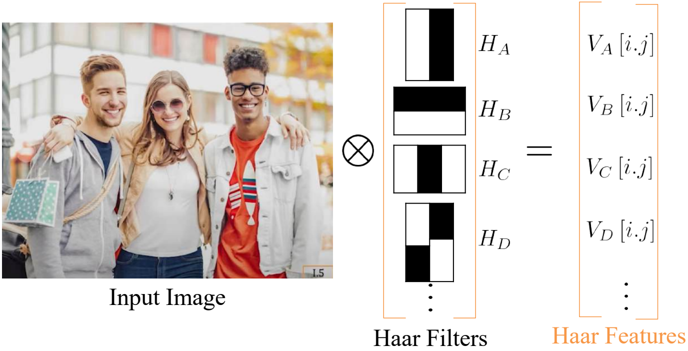
Raster Scanning

Let I_A and II_A be the values of Image and Integral Image, respectively, at pixel A.

$$\boxed{II_A = II_B + II_C - II_D + I_A}$$

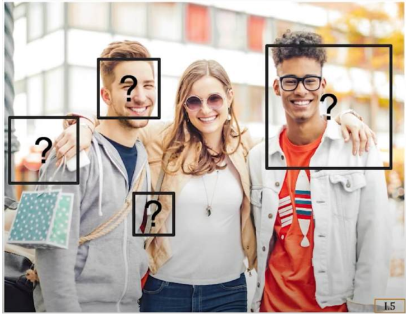
Haar Features using Integral Images

Integral image needs to be computed once per test image.
Allows fast computations of Haar features.



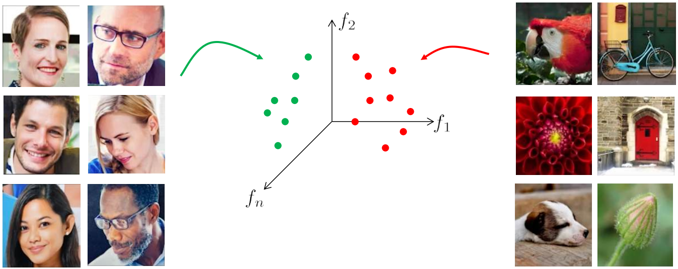
Classifier for Face Detection?

Given the features for a window, how to decide whether it contains a face or not?



Feature Space

Haar Features **f** (a vector) at a pixel
is a point in an n-D space,$\mathbf{f} \in \mathbf{R}^n$

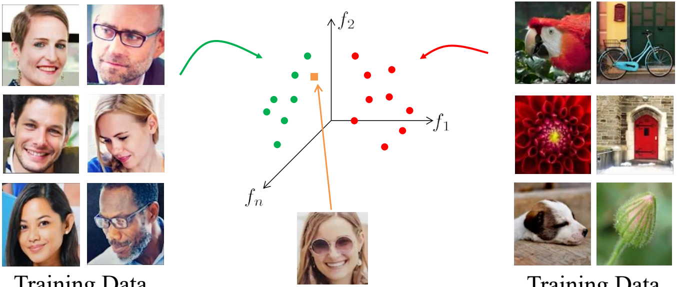


Training Data
of Face

Training Data
of Non-Face

Feature Space

Haar Features **f** (a vector) at a pixel
is a point in an n-D space,$\mathbf{f} \in \mathbf{R}^n$



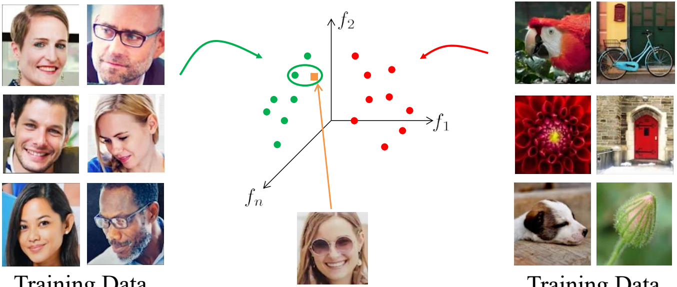
Training Data
of Face

Test Image

Training Data
of Non-Face

Feature Space

Find the Nearest training sample using L^2 distance and assign its label.



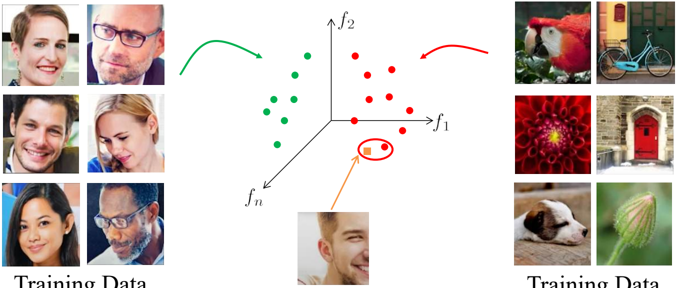
Training Data of Face

Face

Training Data of Non-Face

Feature Space

Find the Nearest training sample using L^2 distance and assign its label.



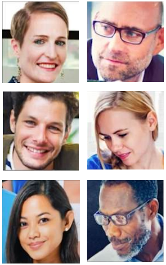
Training Data of Face

Not Face

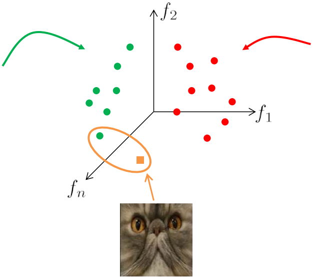
Training Data of Non-Face

Feature Space

Find the Nearest training sample using L^2 distance and assign its label.

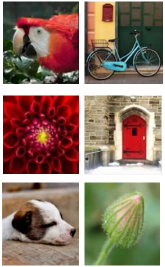


Training Data of Face



Face

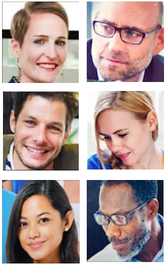
False Positive



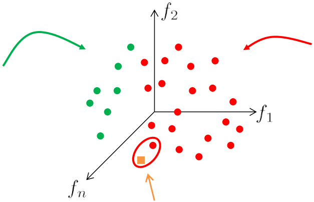
Training Data of Non-Face

Feature Space

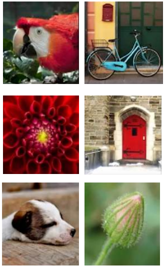
Larger the training set, more robust the NN classifier



Training Data
of Face



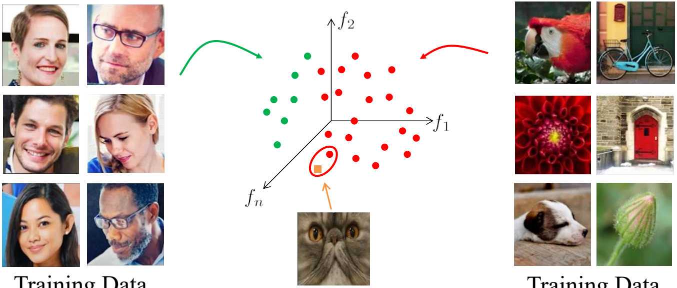
Non-Face



Training Data
of Non-Face

Feature Space

Larger the training set, slower the NN classifier



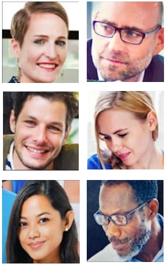
Training Data
of Face

Non-Face

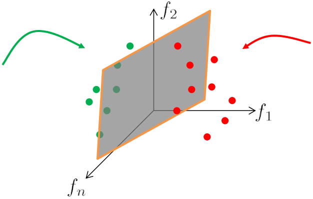
Training Data
of Non-Face

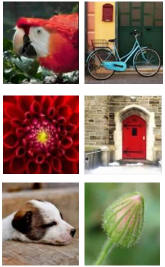
Feature Space

A simple dicision boundary separating
face and non-face classes will suffice



Training Data
of Face

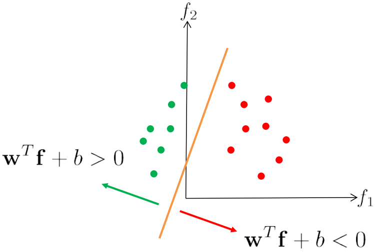




Training Data
of Non-Face

Linear Dicision Boundaries

A Linear Decision Boundary in 2-D space is a 1-D Line



Equation of Line:

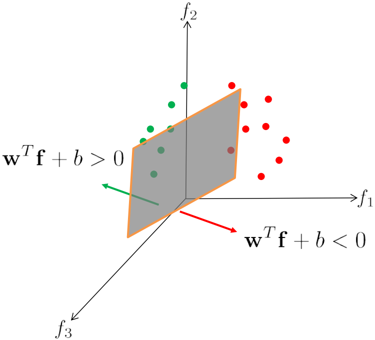
$$w_1 f_1 + w_2 f_2 + b = 0$$

$$\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + b = 0$$

$$\boxed{\mathbf{w}^T\mathbf{f} + b = 0}$$

Linear Dicision Boundaries

A Linear Decision Boundary in 3-D space is a 2-D Plane



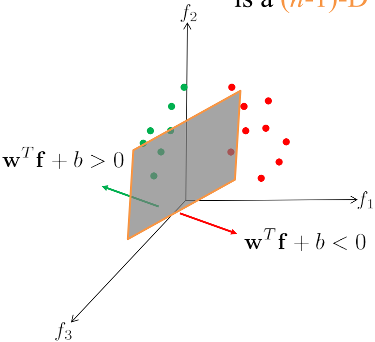
Equation of Plane:

$$w_1 f_1 + w_2 f_2 + w_3 f_3 + b = 0$$

$$\mathbf{w}^T\mathbf{f} + b = 0$$

Linear Dicision Boundaries

A Linear Decision Boundary in *n*-D space
is a (*n*-1)-D Hyperplane



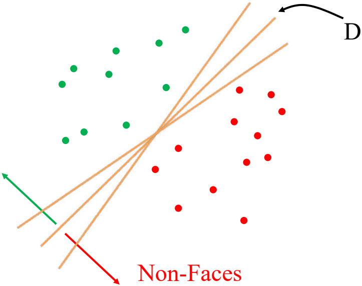
Equation of Hyperplane:

$$w_1 f_1 + w_2 f_2 + \cdots + w_n f_n + b = 0$$

$$\mathbf{w}^T\mathbf{f} + b = 0$$

Decision Boundary ($\mathbf{w}$,b)

What is the optimal decision boundary?



Decision Boundary

$\mathbf{w}^T\mathbf{f} + b = 0$

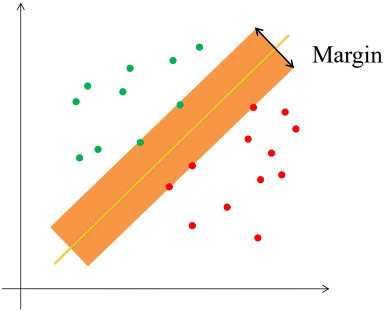
Faces

$\mathbf{w}^T\mathbf{f} + b > 0$

Non-Faces

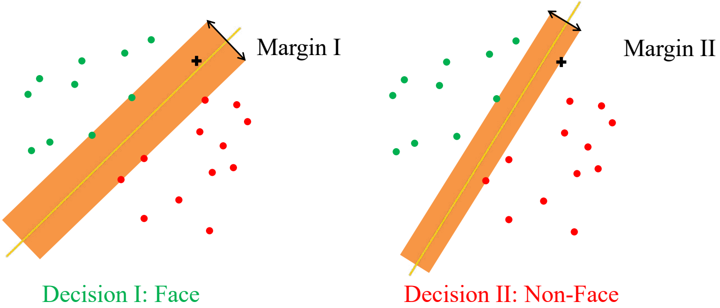
$\mathbf{w}^T\mathbf{f} + b < 0$

Evaluating a Decision Boundary



Margin or Safe Zone: The width that the boundary could be increased by, before hitting a feature point.

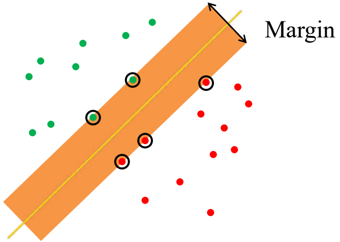
Evaluating a Decision Boundary



Choose Decision Boundary with Maximum Margin!

Support Vector Machine (SVM)

Classifier optimized to Maximum Margin



Support Vectors: Closest data samples to the boundary

Decision Boundary & Margin depend only on Support Vectors

Support Vector Machine (SVM)

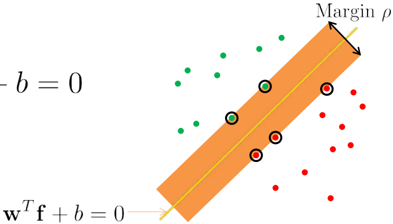
Given:

- k training images $\{I_1, I_2, \ldots, I_k\}$ and their Haar features $\{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_k\}$.
- k corresponding labels $\{\lambda_1, \lambda_2, \ldots, \lambda_k\}$, where $\lambda_j = +1$ if I_j is a face and $\lambda_j = -1$ if I_j is not a face.

Find:

Decision Boundary $\mathbf{w}^T\mathbf{f} + b = 0$

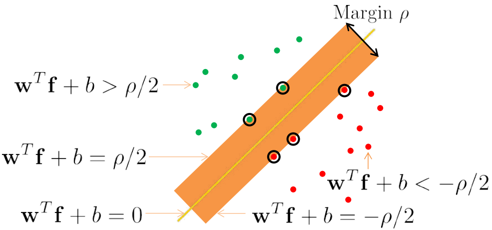
with Maximum Margin ρ



Finding Decision Boundary (**w**,*b*)

For each training sample $(\mathbf{f}_i, \lambda_i)$:

$$\left.\begin{array}{ll}\text{If } \lambda_i = +1: & \mathbf{w}^T\mathbf{f}_i + b \geq \rho/2 \\ \text{If } \lambda_i = -1: & \mathbf{w}^T\mathbf{f}_i + b \leq -\rho/2\end{array}\right\} \quad \boxed{\lambda_i(\mathbf{w}^T\mathbf{f}_i + b) \geq \rho/2}$$



Finding Decision Boundary (**w**,*b*)

For each training sample $(\mathbf{f}_i, \lambda_i)$:

$$\left.\begin{array}{ll} \text{If } \lambda_i = +1 : & \mathbf{w}^T\mathbf{f}_i + b \geq \rho/2 \\ \text{If } \lambda_i = -1 : & \mathbf{w}^T\mathbf{f}_i + b \leq -\rho/2 \end{array}\right\} \boxed{\lambda_i(\mathbf{w}^T\mathbf{f}_i + b) \geq \rho/2}$$

If ***S*** is the set of support vectors,

Then for every support vector $s \in \boldsymbol{S}$: $\boxed{\lambda_s(\mathbf{w}^T\mathbf{f}_s + b) = \rho/2}$

Numerical methods exist to find
w,*b* and ***S*** that maximize ρ

MATLAB: svmtrain

Support Vector Machine (SVM)

Given: Haar features $\mathbf{f}$ for an image window and
SVM parameters $\mathbf{w}, b, \rho, \boldsymbol{S}$

Classification:

Compute $d = \mathbf{w}^T \mathbf{f} + b$

$$\text{If: } \begin{cases} d \geq \rho/2 & \text{Face} \\ d > 0 \text{ and } d < \rho/2 & \text{Probably Face} \\ d < 0 \text{ and } d > -\rho/2 & \text{Probably Not-Face} \\ d \leq -\rho/2 & \text{Not-Face} \end{cases}$$

Face Detection Results

```
OpenCL detector (c) 2011 Michal Hruby

Device: GeForce GTX 285
Execution of 6 kernels: 54843 us
Total frame time: 68.5756 ms (14 fps)
Frame grab: 4.5565 ms
```

Remarks

- Current face detection systems are mature but not perfect.
- Frontal and side poses usually require different face models.
- Successful vision technology used in cameras, surveillance, biometrics, search.
- Performance continues to improve.

Camera Calibration

张举勇
中国科学技术大学

Camera Calibration

- Method to find a camera's internal and external parameters.

Topics:

- (1) Linear Camera Model
- (2) Camera Calibration
- (3) Extracting Intrinsic and Extrinsic Matrices
- (4) Example Application: Simple Stereo

Forward Imaging Model: 3D to 2D

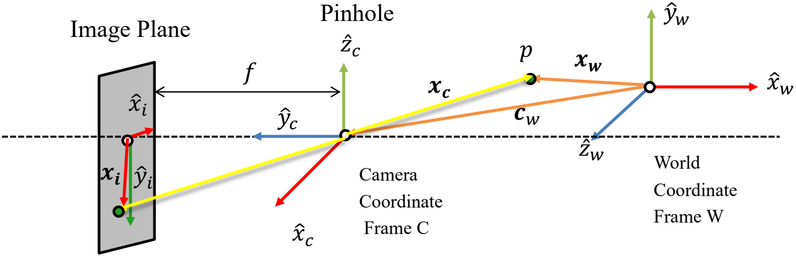


Image Coordinates

$$\boldsymbol{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Camera Coordinates

$$\boldsymbol{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

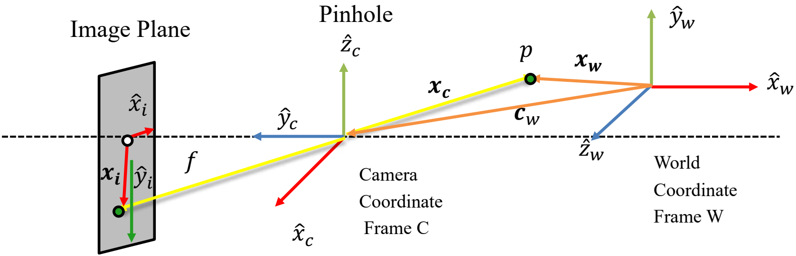
Perspective Projection

World Coordinates

$$\boldsymbol{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

Coordinate Transformation

Forward Imaging Model: 3D to 2D



We know that $\frac{x_i}{f} = \frac{x_c}{z_c}$ and $\frac{y_i}{f} = \frac{y_c}{z_c}$

Therefore: $x_i = f\frac{x_c}{z_c}$ and $y_i = f\frac{y_c}{z_c}$

Image Plane to Image Sensor Mapping

Image Plane

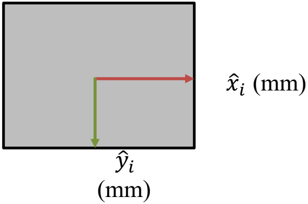
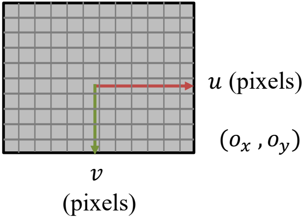


Image Sensor

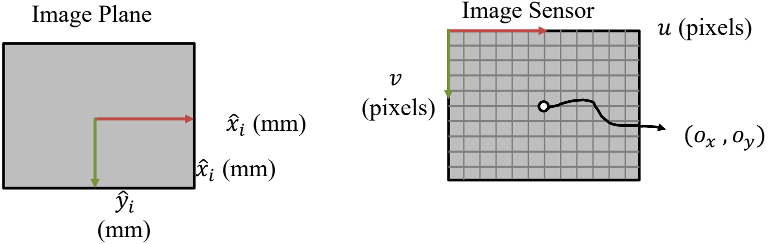


Pixels may be rectangular.

If m_x and m_y are the pixel densities (pixels/mm) in x and y directions, respectively, then pixel coordinates are:

$$u = m_x x_i = m_x f \frac{x_c}{z_c} \qquad v = m_y y_i = m_y f \frac{y_c}{z_c}$$

Image Plane to Image Sensor Mapping



We usually treat the top-left corner of the image sensor as its origin (easier for indexing). If pixel (o_x, o_y) is the **Principle Point** where the optical axis pierces the sensor, then:

$$u = m_x f \frac{x_c}{z_c} + o_x \qquad v = m_y f \frac{y_c}{z_c} + o_y$$

Perspective Projection

$$u = m_x f \frac{x_c}{z_c} + o_x \qquad v = m_y f \frac{y_c}{z_c} + o_y$$

$$\boxed{u = f_x \frac{x_c}{z_c} + o_x \qquad v = f_y \frac{y_c}{z_c} + o_y}$$

where: $(f_x, f_y) = (m_x f, m_y f)$ are the focal lengths in pixels in the x and y directions.

(f_x, f_y, o_x, o_y): **Intrinsic parameters** of the camera. They represent the **camera's internal geometry.**

Perspective Projection

$$u = m_x f \frac{x_c}{z_c} + o_x \qquad v = m_y f \frac{y_c}{z_c} + o_y$$

$$\boxed{u = f_x \frac{x_c}{z_c} + o_x \qquad v = f_y \frac{y_c}{z_c} + o_y}$$

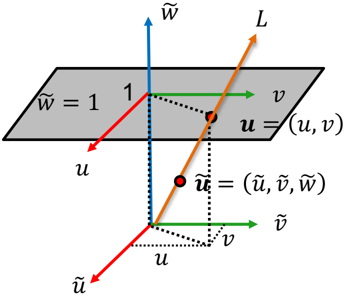
Equations for perspective projection are **Non-Linear**.
It is convenient to express them as linear equations.

Homogenous Coordinates

The homogenous representation of a 2D point $\boldsymbol{u} = (u, v)$ is a 3D point $\tilde{\boldsymbol{u}} = (\tilde{u}, \tilde{v}, \tilde{w})$. The third coordinate $\tilde{w} \neq 0$ is fictitious such that:

$$u = \frac{\tilde{u}}{\tilde{w}} \qquad v = \frac{\tilde{v}}{\tilde{w}}$$

$$\boxed{\boldsymbol{u} \equiv \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}u \\ \tilde{w}v \\ \tilde{u} \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{u} \end{bmatrix} = \tilde{\boldsymbol{u}}}$$



Every point on line L (except origin) represents the homogenous coordinate of u(u, v)

Homogenous Coordinates

The **homogenous** representation of a 3D point $\boldsymbol{x} = (x, y, z) \in \mathcal{R}^3$ is a 4D point $\widetilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{z}, \widetilde{w}) \in \mathcal{R}^4$.The fourth coordinate w ≠0 is fictitious such that:

$$x = \frac{\tilde{x}}{\widetilde{w}} \quad y = \frac{\tilde{y}}{\widetilde{w}} \quad z = \frac{\tilde{z}}{\widetilde{w}}$$

$$\boxed{\boldsymbol{x} \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \widetilde{w}x \\ \widetilde{w}y \\ \widetilde{w}z \\ \widetilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \widetilde{w} \end{bmatrix} = \widetilde{\boldsymbol{x}}}$$

Perspective Projection

Perspective projection equations:

$$u = f_x \frac{x_c}{z_c} + o_x \quad v = f_y \frac{y_c}{z_c} + o_y$$

Homogenous coordinates of (u, v) :

$$\boxed{\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}}$$

where: $(u, v) = (\tilde{u}/\tilde{w}, \tilde{v}/\tilde{w})$

Linear Model for Perspective Projection

Intrinsic Matrix

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Calibration Matrix:

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

Upper Right Triangular Matrix

Intrinsic Matrix:

$$M_{int} = [\, K \mid 0 \,] = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boxed{\tilde{u} = [\, K \mid 0 \,]\tilde{\boldsymbol{x}}_c = M_{int}\widetilde{x_c}}$$

Forward Imaging Model: 3D to 2D

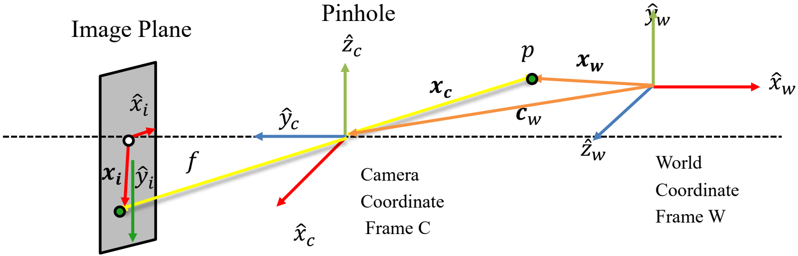


Image Coordinates

$$\boldsymbol{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \xleftarrow{\boldsymbol{M}_{int}}$$

Perspective Projection

Camera Coordinates

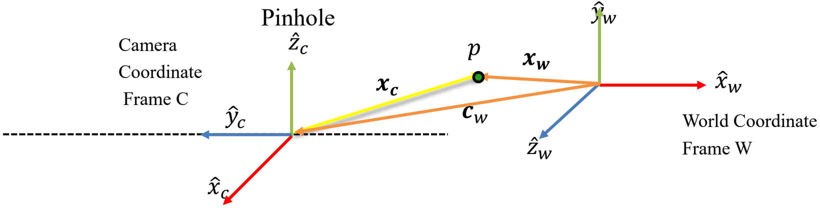
$$\boldsymbol{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \xleftarrow{?}$$

World Coordinates

$$\boldsymbol{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

Coordinate Transformation

Extrinsic Parameters



Position c_w and **Orientation** R of the camera in the world coordinate frame W are the camera's **Extrinsic Parameters**.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{matrix} \rightarrow \text{Row 1: Direction of } \hat{x}_c \text{ in world coordinate frame} \\ \rightarrow \text{Row 2: Direction of } \hat{y}_c \text{ in world coordinate frame} \\ \rightarrow \text{Row 3: Direction of } \hat{z}_c \text{ in world coordinate frame} \end{matrix}$$

Orientation/Rotation Matrix R is **Orthonormal**

Orthonormal Vectors and Matrices

Orthonormal Vectors: Two vectors **u** and **v** are orthonormal if and only if:

$$dot(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^T \boldsymbol{v} = 0 \quad \text{and} \quad \boldsymbol{u}^T \boldsymbol{u} = \boldsymbol{v}^T \boldsymbol{v} = 1$$

(Orthogonality) **(Unit length)**

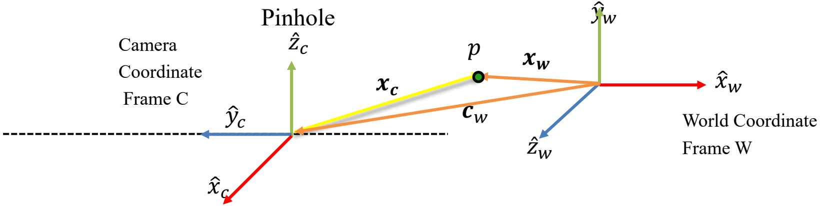
Example: The x-, y- and z-axes of R^3 Euclidean space

Orthonormal Matrix: A square matrix R whose row (or column) vectors are orthonormal. For such a matrix:

$$R^{-1} = R^T \qquad R^T R = R R^T = I$$

A Rotation Matrix is an Orthonormal Matrix

World-to-Camera Transformation



Given the **extrinsic parameters** (R, c_w) of the camera, the camera-centric location of the point P in the world coordinate frame is:

$$\boldsymbol{x}_c = R(\boldsymbol{x_w} - \boldsymbol{c}_w) = R\boldsymbol{x}_w - R\boldsymbol{c}_w = R\boldsymbol{x}_w + \boldsymbol{t}$$

$$\boxed{\boldsymbol{t} = -R\boldsymbol{c}_w}$$

$$\boldsymbol{x_c} = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Extrinsic Matrix

Rewriting using homogenous coordinates:

$$\boxed{x_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}}$$

Extrinsic Matrix: $M_{ext} = \begin{bmatrix} R_{3\times 3} & t \\ 0_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$$\boxed{\widetilde{x_c} = M_{\text{ext}}\, \widetilde{x_w}}$$

Forward Imaging Model: 3D to 2D

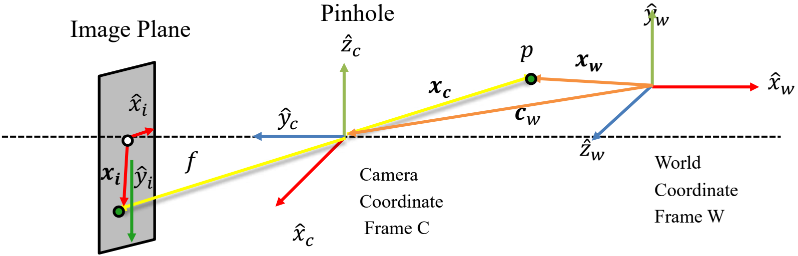


Image Coordinates

Camera Coordinates

World Coordinates

$$\boldsymbol{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \xleftarrow{\boldsymbol{M}_{int}} \boldsymbol{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \xleftarrow{\boldsymbol{M}_{\text{ext}}} \boldsymbol{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

Perspective Projection

Coordinate Transformation

Projection Matrix P

Camera to Pixel

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \widetilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$$\tilde{u} = \boldsymbol{M}_{\boldsymbol{int}} \widetilde{x_c}$$

World to Camera

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$\widetilde{x_c} = \boldsymbol{M}_{\mathbf{ext}} \widetilde{x_w}$$

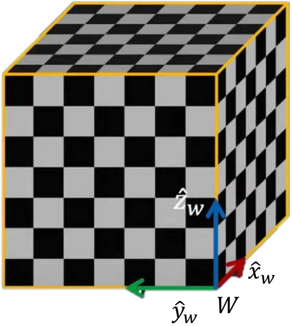
Combining the above two equations, we get the full **projection matrix P**:

$$\tilde{u} = M_{int} M_{\text{ext}} \widetilde{x_w} = P \widetilde{x_w}$$

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \widetilde{w} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Camera Calibration Procedure

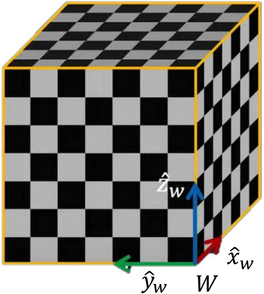
Step1: Capture an image of an object with known geometry.



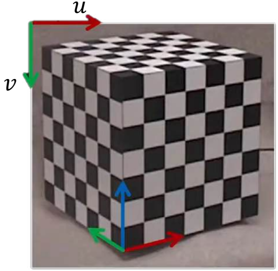
Object of Known Geometry

Camera Calibration Procedure

Step 2: Identify correspondences between 3D scene points and image points.



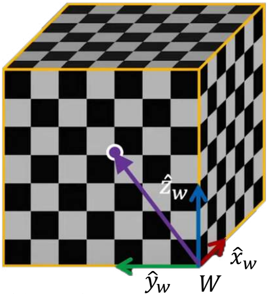
Object of Known Geometry



Captured Image

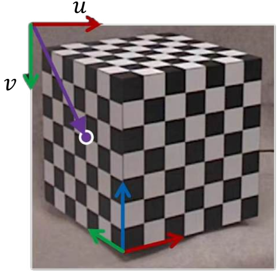
Camera Calibration Procedure

Step 2: Identify correspondences between 3D scene points and image points.



Object of Known Geometry

- $x_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 4 \end{bmatrix}$ (inches)

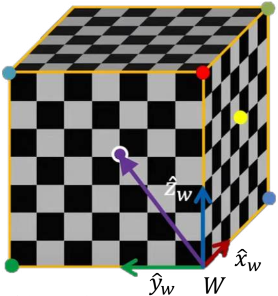


Captured Image

- $u = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 56 \\ 115 \end{bmatrix}$ (pixels)

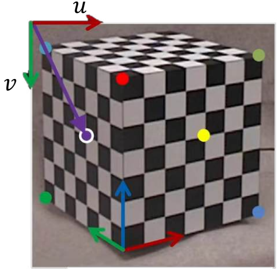
Camera Calibration Procedure

Step 2: Identify correspondences between 3D scene points and image points.



Object of Known Geometry

- $x_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 4 \end{bmatrix}$ (inches)



Captured Image

- $u = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 56 \\ 115 \end{bmatrix}$ (pixels)

Camera Calibration Procedure

Step 3: For each corresponding point i in scene and image:

$$\underbrace{\begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix}}_{\text{Known}} \equiv \underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_{\text{Unknown}} \underbrace{\begin{bmatrix} x_w^{(i)} \\ y_w^{(i)} \\ z_w^{(i)} \\ 1 \end{bmatrix}}_{\text{Known}}$$

Expanding the matrix as linear equations:

$$u^{(i)} = \frac{p_{11}x_w^{(i)} + p_{12}y_w^{(i)} + p_{13}z_w^{(i)} + p_{14}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

$$v^{(i)} = \frac{p_{21}x_w^{(i)} + p_{22}y_w^{(i)} + p_{23}z_w^{(i)} + p_{24}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

Camera Calibration Procedure

Step4: Rearranging the terms

$$\underbrace{\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1x_w^{(1)} & -u_1y_w^{(1)} & -u_1z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1x_w^{(1)} & -v_1y_w^{(1)} & -v_1z_w^{(1)} & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & -u_ix_w^{(i)} & -u_iy_w^{(i)} & -u_iz_w^{(i)} & -u_i \\ 0 & 0 & 0 & 0 & x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & -v_ix_w^{(i)} & -v_iy_w^{(i)} & -v_iz_w^{(i)} & -v_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_nx_w^{(n)} & -u_ny_w^{(n)} & -u_nz_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_nx_w^{(n)} & -v_ny_w^{(n)} & -v_nz_w^{(n)} & -v_n \end{bmatrix}}_{\substack{A \\ \text{Known}}} \underbrace{\begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}}_{\substack{\boldsymbol{p} \\ \text{Unknown}}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Step5: Solve for P

$$\boxed{A\,\boldsymbol{p} = 0}$$

Scale of Projection Matrix

Projection matrix acts on homogenous coordinates.

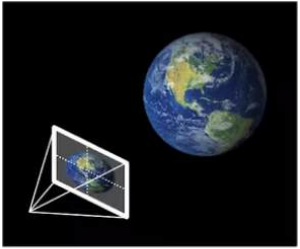
We know that: $$\begin{bmatrix}\tilde{u}\\ \tilde{v}\\ \widetilde{w}\end{bmatrix} \equiv k\begin{bmatrix}\tilde{u}\\ \tilde{v}\\ \widetilde{w}\end{bmatrix}$$ (k ≠0 is any constant)

That is: $$\begin{bmatrix}p_{11} & p_{12} & p_{13} & p_{14}\\ p_{21} & p_{22} & p_{23} & p_{24}\\ p_{31} & p_{32} & p_{33} & p_{34}\end{bmatrix}\begin{bmatrix}x_w\\ y_w\\ z_w\\ 1\end{bmatrix} \equiv k\begin{bmatrix}p_{11} & p_{12} & p_{13} & p_{14}\\ p_{21} & p_{22} & p_{23} & p_{24}\\ p_{31} & p_{32} & p_{33} & p_{34}\end{bmatrix}\begin{bmatrix}x_w\\ y_w\\ z_w\\ 1\end{bmatrix}$$

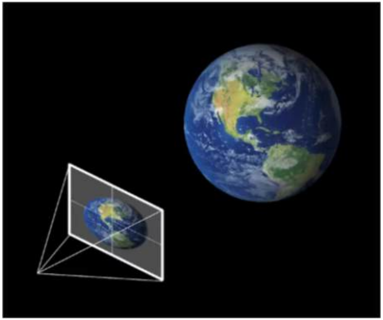
Therefore, Projection Matrices P and kP produce the same homogenous pixel coordinates.

Projection Matrix P is defined only up to a scale.

Scale of Projection Matrix



Scale = k_1



Scale = k_2

Scaling projection matrix, implies simultaneously scaling the world and camera, which does not change the image.

Set projection matrix to some arbitrary scale!

Least squares Solution for P

Option 1: Set scale so that: $p_{34} = 1$

Option 2: Set scale so that: $\boxed{||p||^2 = 1}$

We want Ap as close to 0 as possible and $||p||^2 = 1$:

$$\min_{\mathrm{p}} \ ||\mathrm{Ap}||^2 \text{ such that } ||\mathrm{p}||^2 = 1$$

$$\min_{p}(p^T A^T A p) \ \text{ such that } \ p^T p = 1$$

Define **Loss function** $L(\boldsymbol{p}, \lambda)$:

$$L(\boldsymbol{p}, \lambda) = \boldsymbol{p}^{\boldsymbol{T}} A^T A \boldsymbol{p} - \lambda\left(\boldsymbol{p}^{\boldsymbol{T}} \boldsymbol{p} - 1\right)$$

(Similar to Solving Homography in Image Stitching)

Constrained Least Squares Solution

Taking derivatives of $L(\boldsymbol{p}, \lambda)$ w.r.t $\boldsymbol{p}$: $2A^T A\boldsymbol{p} - 2\lambda\boldsymbol{p} = 0$

$$\boxed{A^T A\boldsymbol{p} = \lambda\boldsymbol{p}}$$ **Eigenvalue Problem**

Eigenvector $\boldsymbol{p}$ with **smallest eigenvalue** λ of matrix $A^T A$ minimizes the loss function $L(\boldsymbol{p})$.

Rearrange solution $\boldsymbol{p}$ to form the projection matrix $\boldsymbol{P}$.

Extracting Intrinsic/Extrinsic Parameters

We know that:

$$P=\begin{bmatrix}p_{11} & p_{12} & p_{13} & p_{14}\\ p_{21} & p_{22} & p_{23} & p_{24}\\ p_{31} & p_{32} & p_{33} & p_{34}\end{bmatrix}=\underbrace{\begin{bmatrix}f_x & 0 & o_x & 0\\ 0 & f_y & o_y & 0\\ 0 & 0 & 1 & 0\end{bmatrix}}_{M_{\text{int}}}\underbrace{\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\ r_{21} & r_{22} & r_{23} & t_y\\ r_{31} & r_{32} & r_{33} & t_z\\ 0 & 0 & 0 & 1\end{bmatrix}}_{M_{\text{ext}}}$$

That is:

$$\begin{bmatrix}p_{11} & p_{12} & p_{13}\\ p_{21} & p_{22} & p_{23}\\ p_{31} & p_{32} & p_{33}\end{bmatrix}=\begin{bmatrix}f_x & 0 & o_x\\ 0 & f_y & o_y\\ 0 & 0 & 1\end{bmatrix}\boxed{\begin{bmatrix}r_{11} & r_{12} & r_{13}\\ r_{21} & r_{22} & r_{23}\\ r_{31} & r_{32} & r_{33}\end{bmatrix}}=KR$$

Given that K is an **Upper Right Triangular** matrix and R is an **Orthonormal** matrix, it is possible to uniquely "**decouple**" K and R from their product using "**QR factorization**".

Extracting Intrinsic/Extrinsic Parameters

We know that:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{M_{\text{int}}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{M_{\text{ext}}}$$

That is:

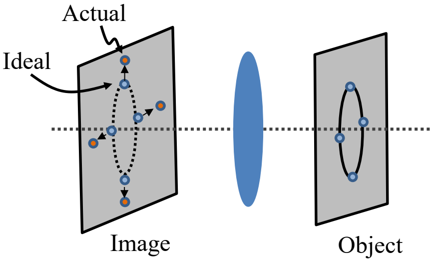
$$\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = K\boldsymbol{t}$$

Therefore:

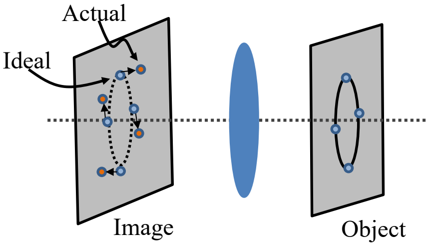
$$\boxed{\boldsymbol{t} = K^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}}$$

Camera Calibration

Pinholes do not exhibit image distortions. But, lenses do!



Tangential Distortion

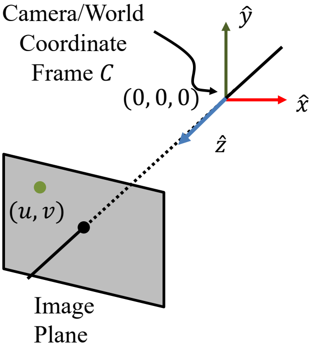


Radial Distortion

The intrinsic model of the camera will need to include
the distortion coefficients. We ignore distortions here.

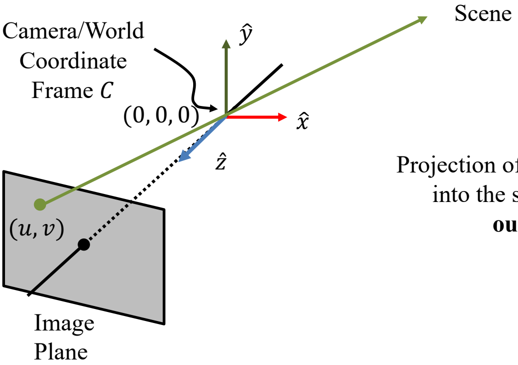
Backward Projection: From 2D to 3D

Given a calibrated camera, can we find the 3D scene point from a single 2D image?



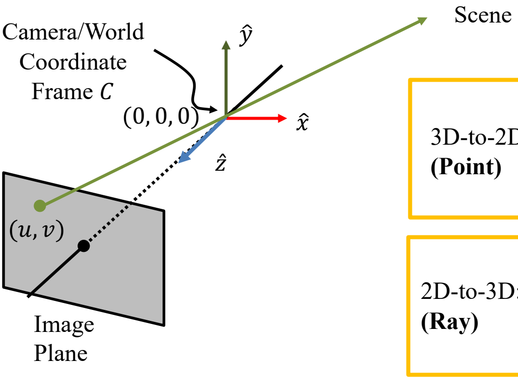
Backward Projection: From 2D to 3D

Given a calibrated camera, can we find the 3D scene point from a single 2D image?



Projection of an image point back into the scene results in an **outgoing ray.**

Computing 2D-to-3D Outgoing Ray



3D-to-2D:
(Point)

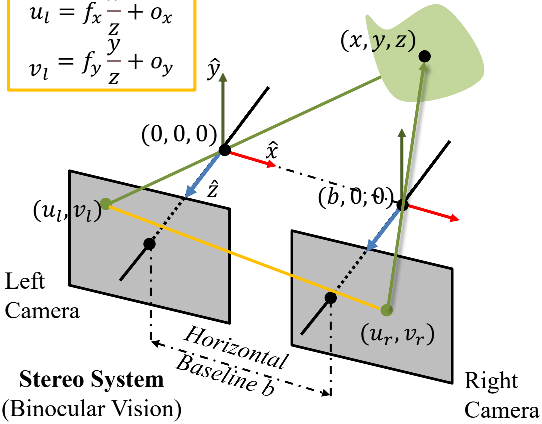
$$u = f_x \frac{x_c}{z_c} + o_x$$
$$v = f_y \frac{y_c}{z_c} + o_y$$

2D-to-3D:
(Ray)

$$\mathrm{x} = \mathrm{z}/f_x(u - o_x)$$
$$\mathrm{y} = z/f_y\left(v - o_y\right)$$
$$z > 0$$

Triangulation using Two Cameras

$$u_l = f_x \frac{x}{z} + o_x$$
$$v_l = f_y \frac{y}{z} + o_y$$



Stereo System
(Binocular Vision)

$$u_r = f_x \frac{x - b}{z} + o_x$$
$$v_r = f_y \frac{y}{z} + o_y$$

f_x, f_y, b, o_x, o_y are known.

Simple Stereo: Depth and Disparity

From perspective projection:

$$(u_l, v_l) = \left(f_x \frac{x}{z} + o_x, f_y \frac{y}{z} + o_y\right) \quad (u_r, v_r) = \left(f_x \frac{x-b}{z} + o_x, f_y \frac{y}{z} + o_y\right)$$

Solving for (x, y, z):

$$x = \frac{b(u_l - o_x)}{(u_l - u_r)} \quad y = \frac{bf_x(v_l - o_y)}{f_y(u_l - u_r)} \quad \boxed{z = \frac{bf_x}{(u_l - u_r)}}$$

where $(u_l - u_r)$ is called **Disparity**.

Depth z is inversely proportional to Disparity.

Disparity is proportional to Baseline.

A Simple Stereo Camera

Fujifilm FinePix REAL 3D W3

Stereo Matching: Finding Disparities

Goal: Find the disparity between left and right stereo pairs.

Left/Right Camera Images

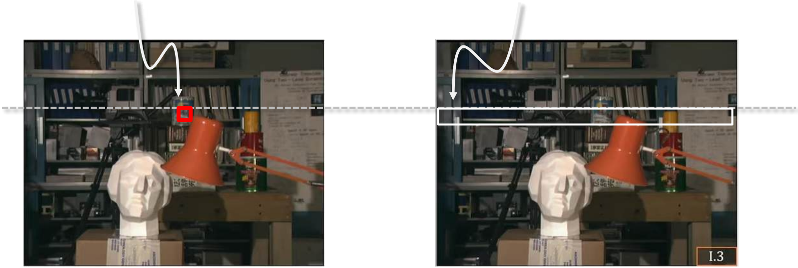
Disparity Map(Ground Truth)

From perspective projection: $v_l = v_r = f_y \frac{y}{z} + o_y$

Corresponding scene points lie on the same horizontal scan line.

Window Based Methods

Determine Disparity using **Template Matching**



Template Window T

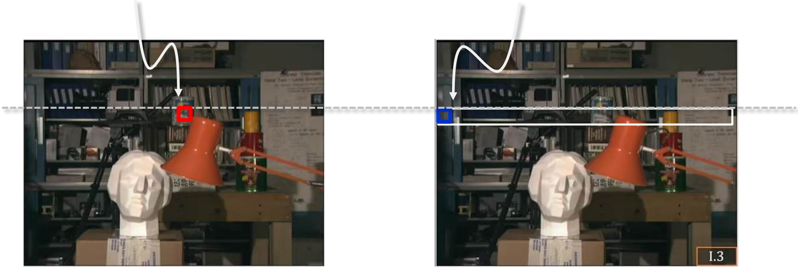
Search Scan Line L

Left Camera Image E_l

Right Camera Image E_r

Window Based Methods

Determine Disparity using **Template Matching**



Template Window T

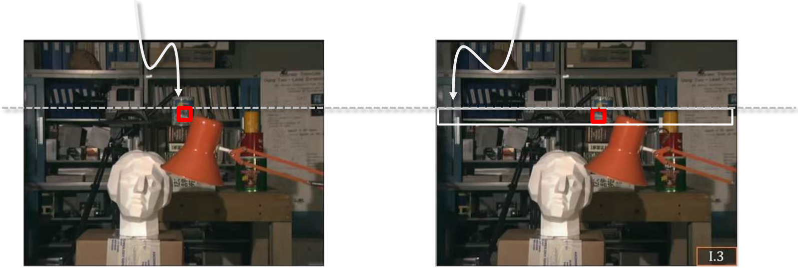
Search Scan Line L

Left Camera Image E_l

Right Camera Image E_r

Window Based Methods

Determine Disparity using **Template Matching**



Template Window T

Search Scan Line L

Left Camera Image E_l

Right Camera Image E_r

Disparity: $d = u_l - u_r$

Depth: $z = \dfrac{bf_x}{(u_l - u_r)}$

Similarity Metrics for Template Matching

Find pixel $(k, l) \in L$ with Minimum **Sum of Absolute Differences:**

$$SAD(k,l) = \sum\nolimits_{(i,j)\in T} |E_l(i,j) - E_r(i+k, j+l)|$$

Find pixel $(k, l) \in L$ with Minimum **Sum of Squared Differences:**

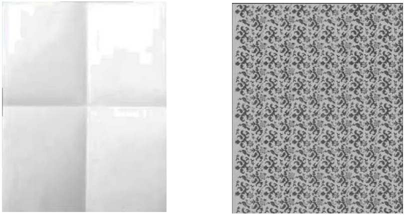
$$SSD(k,l) = \sum\nolimits_{(i,j)\in T} |E_l(i,j) - E_r(i+k, j+l)|^2$$

Find pixel $(k, l) \in L$ with Maximum **Normalized Cross-Correlation:**

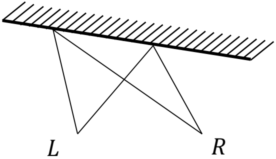
$$NCC(k,l) = \frac{\sum_{(i,j)\in T} E_l(i,j) E_r(i+k, j+l)}{\sqrt{\sum_{(i,j)\in T} E_l(i,j)^2 \sum_{(i,j)\in T} E_r(i+k, j+l)^2}}$$

Issues with Stereo Matching

- Surface must have (non-repetitive) texture



- Foreshortening effect makes matching challenging



How Large Should Window Be?

Window size = 5 pixels
(Sensitive to noise)

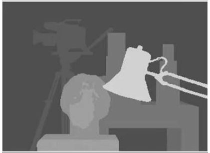
Window size = 30 pixels
(Poor localization)

Adaptive Window Method Solution: For each point, match using windows of multiple sizes and use the disparity that is a result of the best similarity measure (minimize SSD per pixel).

Window Based Methods: Results

Left Image

Right Image



Ground Truths

SSD - Adaptive Window

SD (Window size=21)

State of the Art

http://vision.middlebury.edu/stereo

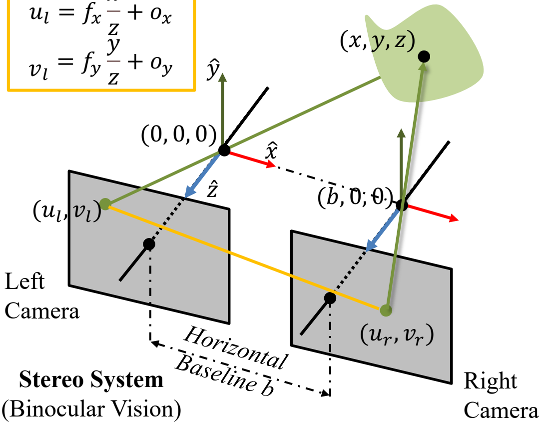
中国科学技术大学
University of Science and Technology of China

Uncalibrated Stereo

张举勇
中国科学技术大学

Simple (Calibrated) Stereo

$$u_l = f_x \frac{x}{z} + o_x$$
$$v_l = f_y \frac{y}{z} + o_y$$



$$u_r = f_x \frac{x - b}{z} + o_x$$
$$v_r = f_y \frac{y}{z} + o_y$$

f_x, f_y, b, o_x, o_y are in pixel units.

Depth and Disparity

Solving for (x, y, z):

$$x = \frac{b(u_l - o_x)}{(u_l - u_r)} \quad y = \frac{bf_x(v_l - o_y)}{f_y(u_l - u_r)} \quad \boxed{z = \frac{bf_x}{(u_l - u_r)}}$$

where $(u_l - u_r)$ is called **Disparity**.

Uncalibrated stereo

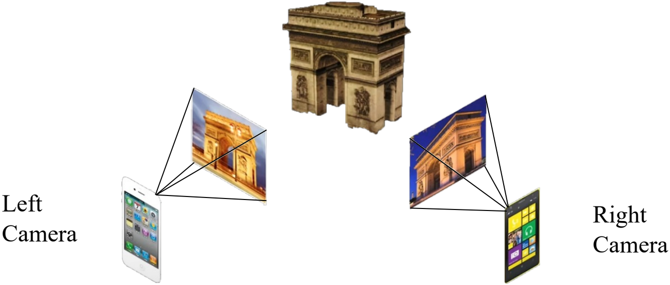
- Method to estimate 3D structure of a static scene from two arbitrary views.

Topics:

- (1) Problem of Uncalibrated Stereo
- (2) Epipolar Geometry
- (3) Estimating Fundamental Matrix
- (4) Finding Dense Correspondences
- (5) Computing Depth

Uncalibrated Stereo

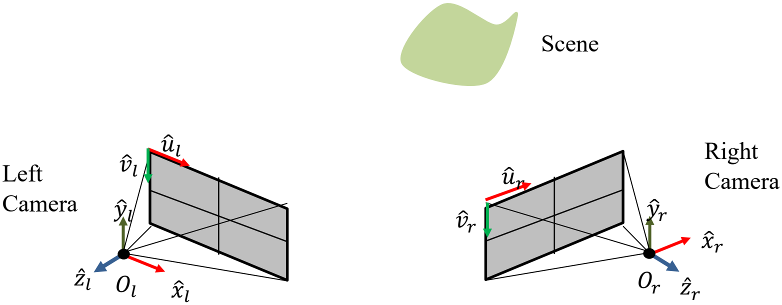
Compute **3D structure of static scene** from **two arbitrary views**



Instrinsics (f_x, f_y, o_x, o_y) are **known** for both views/cameras.

Extrinsics (relative position/orientation of cameras) are **unknown**.

Uncalibrated Stereo



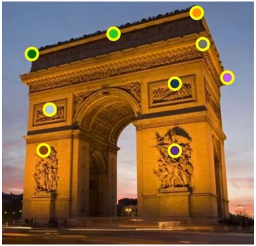
✓ 1. Assume Camera Matrix K is known for each camera

2. Find a few Reliable Corresponding Points

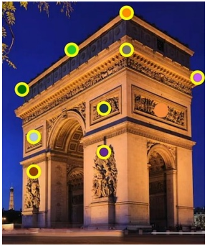
Initial Correspondence

Find a set of **corresponding features (at least 8)** in left and right images (e.g. using SIFT or hand-picked).

Left image

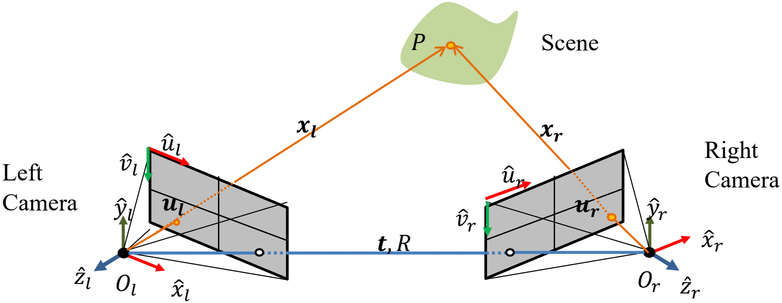


Right image



● $\left(u_l^{(1)}, v_l^{(1)}\right)$	● $\left(u_r^{(1)}, v_r^{(1)}\right)$
$\vdots$	$\vdots$
● $\left(u_l^{(m)}, v_l^{(m)}\right)$	● $\left(u_r^{(m)}, v_r^{(m)}\right)$

Uncalibrated Stereo

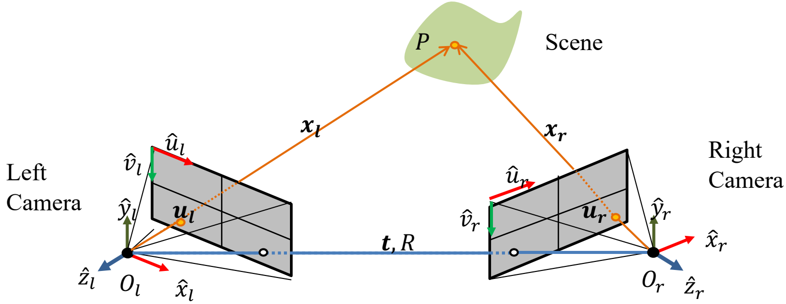


✓ 1. Assume Camera Matrix K is known for each camera

✓ 2. Find a few Reliable Corresponding Points

3. Find Relative Camera Position t and Orientation R
4. Find Dense Correspondence
5. Compute Depth using Triangulation

Epipolar Geometry: Epipoles

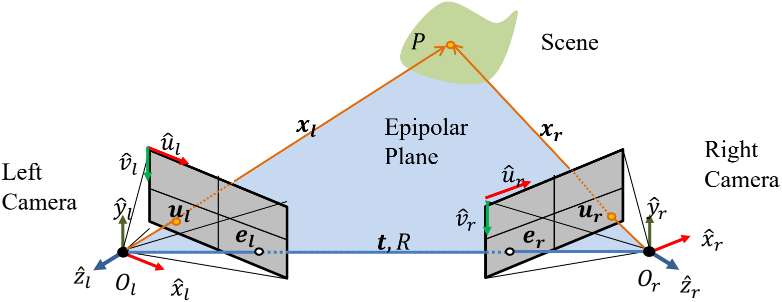


Epipole: Image point of origin/pinhole of one camera as viewed by the other camera.

$\boldsymbol{e_l}$ and $\boldsymbol{e_r}$ are the epipoles.

$\boldsymbol{e_l}$ and $\boldsymbol{e_r}$ are unique for a given stereo pair.

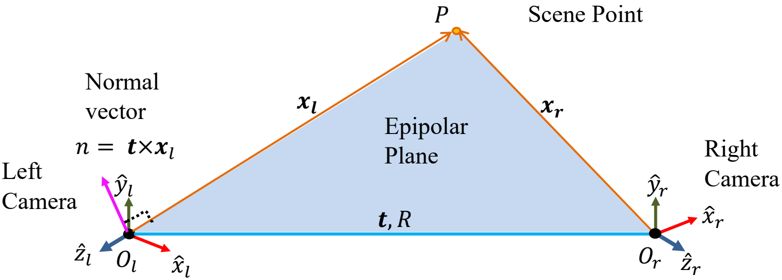
Epipolar Geometry: Epipolar Plane



Epipolar Plane of Scene Point P: The plane formed by camera origins (O_l and O_r), epipoles ($\boldsymbol{e}_l$ and $\boldsymbol{e}_r$) and scene point P.

Every scene point lies on a **unique epipolar plane**.

Epipolar Constraint



Vector normal to the epipolar plane: $n = \boldsymbol{t} \times \boldsymbol{x}_l$

Dot product of $\boldsymbol{n}$ and $\boldsymbol{x}_l$ (perpendicular vectors) is zero:

$$\boxed{\boldsymbol{x}_l \cdot (\boldsymbol{t} \times \boldsymbol{x}_l) = 0}$$

Epipolar Constraint

Writing the epipolar constraint in matrix form:

$$\boldsymbol{x}_l \cdot (\boldsymbol{t}\times\boldsymbol{x}_l) = 0$$

$$[x_l \quad y_l \quad z_l]\begin{bmatrix} t_y z_l - t_z y_l \\ t_z x_l - t_x z_l \\ t_x y_l - t_y x_l \end{bmatrix} = 0$$

Cross-product definition

$$\boxed{[x_l \quad y_l \quad z_l]\underbrace{\begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}}_{T_\times}\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = 0}$$

Matrix-vector form

$\boldsymbol{t}_{3\times1}$: Position of Right Camera in Left Camera's Frame

$R_{3\times3}$: Orientation of Left Camera in Right Camera's Frame

$$\boxed{\boldsymbol{x}_l = R\boldsymbol{x}_r + \boldsymbol{t}}$$

$$\boxed{\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}}$$

Epipolar Constraint

Substituting into the epipolar constraint gives:

$$\begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \left(\boxed{\begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} + \boxed{\begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}} \right) = 0$$

$$\boldsymbol{t} \times \boldsymbol{t} = 0$$

$$\begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \boxed{\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0$$

Essential Matrix E

$$\boxed{E = T_{\times} R}$$

[Longuet-Higgins 1981]

Essential Matrix E: Decomposition

$$\boxed{E = T_{\times} R}$$

$$\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \boxed{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}$$

Given that Tx is a **Skew-Symmetric** matrix ($\boldsymbol{a_{ij} = -a_{ji}}$) and R is an **Orthonormal** matrix, it is possible to "**decouple**" $T_{\times}$ and R from their product using **"Singular Value Decomposition".**

Take Away: If E is known, we can calculate **t** and R.

How do we find E?

Relates 3D position (x_l, y_l, z_l) of scene point w.r.t left camera to its 3D position (x_r, y_r, z_r) w.r.t right camera

$$\boxed{\boldsymbol{x}_l^T E \boldsymbol{x}_r = 0}$$

$$\begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0$$

3D position in left camera coordinates

3x3 Essential Matrix

3D position in right camera coordinates

Unfortunately, **we don't have $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$**.

But we do know corresponding points in image coordinates.

Incorporating the Image Coordinate

Perspective projection equations for left camera:

$$u_l = f_x^{(l)} \frac{x_l}{z_l} + o_x^{(l)} \qquad v_l = f_y^{(l)} \frac{y_l}{z_l} + o_y^{(l)}$$

$$z_l u_l = f_x^{(l)} x_l + z_l o_x^{(l)} \qquad z_l v_l = f_y^{(l)} y_l + z_l o_y^{(l)}$$

Representing in matrix form:

$$z_l \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = \begin{bmatrix} z_l u_l \\ z_l v_l \\ z_l \end{bmatrix} = \begin{bmatrix} f_x^{(l)} x_l + z_l o_x^{(l)} \\ f_y^{(l)} y_l + z_l o_y^{(l)} \\ z_l \end{bmatrix} = \underbrace{\begin{bmatrix} f_x^{(l)} & 0 & o_x^{(l)} \\ 0 & f_y^{(l)} & o_y^{(l)} \\ 0 & 0 & 1 \end{bmatrix}}_{\substack{\text{Known} \\ \text{Camera Matrix } K_i}} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix}$$

Incorporating the Image Coordinate

Left camera:

$$z_l \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x^{(l)} & 0 & o_x^{(l)} \\ 0 & f_y^{(l)} & o_y^{(l)} \\ 0 & 0 & 1 \end{bmatrix}}_{K_l} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix}$$

Right camera:

$$z_r \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x^{(r)} & 0 & o_x^{(r)} \\ 0 & f_y^{(r)} & o_y^{(r)} \\ 0 & 0 & 1 \end{bmatrix}}_{K_r} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix}$$

$$\boxed{\boldsymbol{x}_l^T = [u_l \quad v_l \quad 1] z_l K_l^{-1^T}}$$

$$\boxed{\boldsymbol{x}_r K_r^{-1} Z_r = \begin{bmatrix} c u_r \\ v_r \\ 1 \end{bmatrix}}$$

Incorporating the Image Coordinate

Epipolar constraint:

$$[x_l \quad y_l \quad z_l]\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0$$

Rewriting in terms of image coordinates:

$$[u_l \quad v_l \quad 1]\cancel{z_l} K_l^{-1}\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} K_r^{-1}\cancel{z_r}\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

$$\boxed{\begin{array}{c} z_l \neq 0 \\ z_r \neq 0 \end{array}}$$

Incorporating the Image Coordinate

Epipolar constraint:

$$[x_l \quad y_l \quad z_l]\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0$$

Rewriting in terms of image coordinates:

$$\boxed{[u_l \quad v_l \quad 1]K_l^{-1}\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}K_r^{-1}\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0}$$

Fundamental Matrix F

Epipolar constraint:

$$\begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0$$

Rewriting in terms of image coordinates:

$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} \boxed{\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

Fundamental Matrix F

$$\boxed{E = K_l^T F K_r} \qquad \boxed{E = T_\times R}$$

[Fagueras 1992, Luong 1992]

Stereo Calibration Procedure

Find a set of corresponding features in left and right images(e.g. using SIFT or hand-picked)

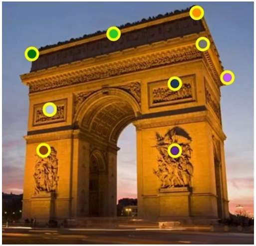
Left image

Right image

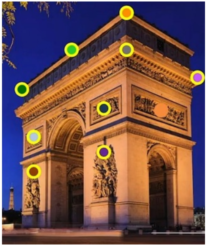
Stereo Calibration Procedure

Find a set of corresponding features in left and right images(e.g. using SIFT or hand-picked)

Left image



Right image



● $\left(u_l^{(1)}, v_l^{(1)}\right)$

$\vdots$

● $\left(u_l^{(m)}, v_l^{(m)}\right)$

● $\left(u_r^{(1)}, v_r^{(1)}\right)$

$\vdots$

● $\left(u_r^{(m)}, v_r^{(m)}\right)$

Stereo Calibration Procedure

Step A: For each correspondence i, write out epipolar constraint.

$$\underbrace{\begin{bmatrix} u_l^{(i)} & v_l^{(i)} & 1 \end{bmatrix}}_{\text{Known}} \underbrace{\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}}_{\text{Unknown}} \underbrace{\begin{bmatrix} u_r^{(i)} \\ v_r^{(i)} \\ 1 \end{bmatrix}}_{\text{Known}} = 0$$

Expand the matrix to get linear equation:

$$\left(f_{11}u_r^{(i)} + f_{12}v_r^{(i)} + f_{13}\right)u_l^{(i)} + \left(f_{21}u_r^{(i)} + f_{22}v_r^{(i)} + f_{23}\right)v_l^{(i)} + f_{31}u_r^{(i)} + f_{32}v_r^{(i)} + f_{33} = 0$$

Stereo Calibration Procedure

Step B: Rearrange terms to form a linear system.

$$\boxed{\begin{bmatrix} u_l^{(1)}u_r^{(1)} & u_l^{(1)}v_r^{(1)} & u_l^{(1)} & v_l^{(1)}u_r^{(1)} & v_l^{(1)}v_r^{(1)} & v_l^{(1)} & u_r^{(1)} & v_r^{(1)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_l^{(i)}u_r^{(i)} & u_l^{(i)}v_r^{(i)} & u_l^{(i)} & v_l^{(i)}u_r^{(i)} & v_l^{(i)}v_r^{(i)} & v_l^{(i)} & u_l^{(i)} & u_r^{(i)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_l^{(m)}u_r^{(m)} & u_l^{(m)}v_r^{(m)} & u_l^{(m)} & v_l^{(m)}u_r^{(m)} & v_l^{(m)}v_r^{(m)} & v_l^{(m)} & u_l^{(m)} & u_r^{(m)} & 1 \end{bmatrix}} \boxed{\begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

A
(Known)

$\boldsymbol{f}$
(Unknown)

$$\boxed{A\boldsymbol{f} = 0}$$

The Tale of Missing Scale

Fundamental matrix acts on homogenous coordinates.

$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0 = \begin{bmatrix} u_l & v_l & 1 \end{bmatrix} \begin{bmatrix} kf_{11} & kf_{12} & kf_{13} \\ kf_{21} & kf_{22} & kf_{23} \\ kf_{31} & kf_{32} & kf_{33} \end{bmatrix} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix}$$

Fundamental Matrix F and kF describe the same epipolar geometry. That is, F is defined only up to a scale.

Set Fundamental Matrix to some arbitrary scale.

$$\boxed{\|\boldsymbol{f}\|^2 = 1}$$

Solving for *F*

Step C: Find least squares solution for fundamental matrix F.

We want $A\boldsymbol{f}$ as close to 0 as possible and $\|\boldsymbol{f}\|^2 = 1$:

$$\boxed{\min_{f} \ \|A\boldsymbol{f}\|^2 \text{ such that } \|\boldsymbol{f}\|^2 = 1}$$

Constrained linear least squares problem

Like solving Projection Matrix during Camera Calibration.
Or, Homography Matrix for Image Stitching.

Rearrange solution f to form the fundamental matrix F.

Extracting Rotation and Translation

Step D: Compute essential matrix E from known left and right intrinsic camera matrices and fundamental matrix F.

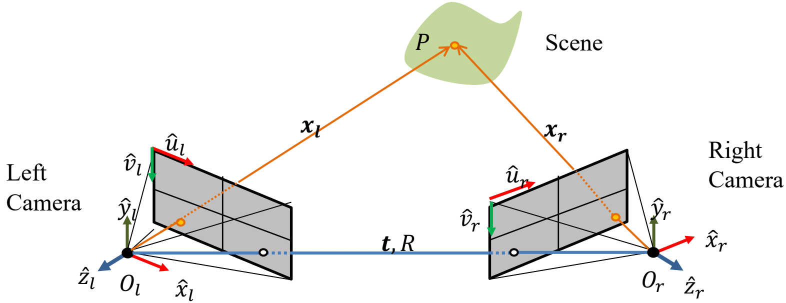
$$\boxed{E = K_l^T F K_r}$$

Step E: Extract R and **t** from E.

$$\boxed{E = T_\times R}$$

(Using Singular Value Decomposition)

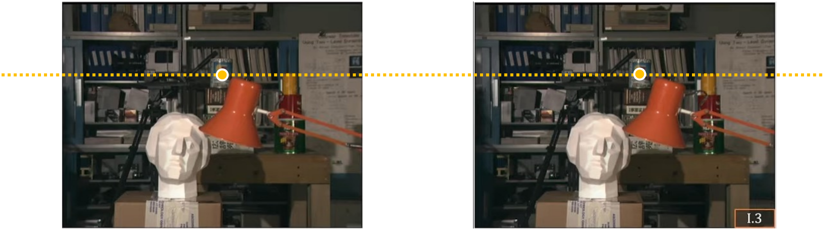
Uncalibrated Stereo



- ✓ 1. Assume Camera Matrix K is known for each camera
- ✓ 2. Find a few Reliable Corresponding Points
- ✓ 3. Find Relative Camera Position t and Orientation R
- 4. Find Dense Correspondence
- 5. Compute Depth using Triangulation

Simple Stereo: Finding Correspondences

Goal: Find the disparity between left and right stereo pairs.

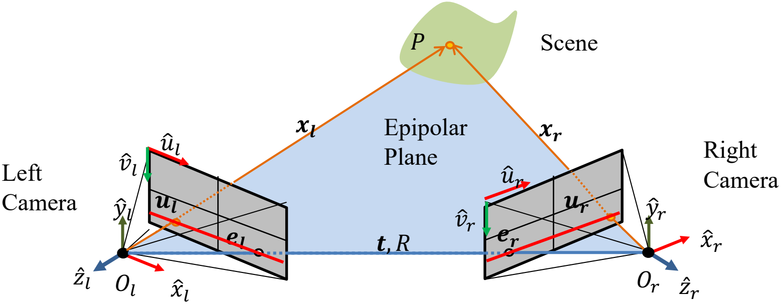


Left/Right Camera Images

Disparity Map(Ground Truth)

Corresponding scene points lie on the **same horizontal scan-line**
Finding correspondence is a **1D search**.

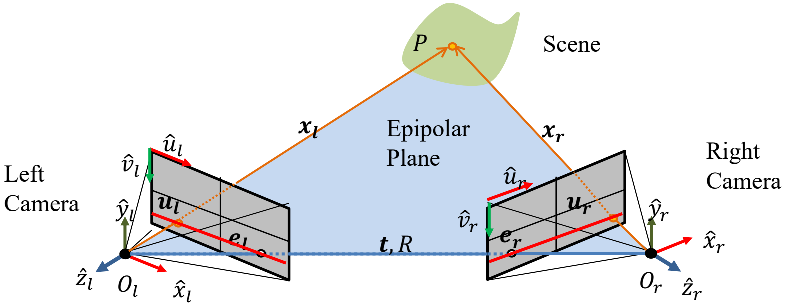
Epipolar Geometry: Epipolar Line



Epipolar Line: Intersection of image plane and epipolar plane.

Every scene point has **two corresponding epipolar lines**, one each on the two image planes.

Epipolar Geometry: Epipolar Line



Given a point in one image, the corresponding point in the other image must lie on the epipolar line.

Finding correspondence reduces to a 1D search.

Finding Epipolar Lines

Given: Fundamental matrix F and point on left image (u, v)
Find: Equation of Epipolar line in the right image

Epipolar Constraint Equation:

$$[u_l \quad v_l \quad 1]\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}\begin{bmatrix} \boldsymbol{u_r} \\ \boldsymbol{v_r} \\ \mathbf{1} \end{bmatrix} = 0$$

Expanding the matrix equation gives:

$$(f_{11}u_l + f_{21}v_l + f_{31})\boldsymbol{u_r} + (f_{12}u_l + f_{22}v_l + f_{32})\boldsymbol{v_r} + (f_{13}u_l + f_{23}v_l + f_{33}) = 0$$

Equation for **right epipolar line**: $\boxed{a_l\boldsymbol{u_r} + b_l\boldsymbol{v_r} + c_l = 0}$

Similarly we can calculate epipolar line in left image for a point in right image.

Finding Epipolar Lines: Example

Given the Fundamental matrix,

$$F = \begin{bmatrix} -.003 & -.028 & 13.19 \\ -.003 & -.008 & -29.2 \\ 2.97 & 56.38 & -9999 \end{bmatrix}$$

Left Image

Right Image

Finding Epipolar Lines: Example

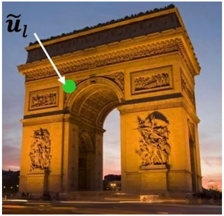
Given the Fundamental matrix,

$$F = \begin{bmatrix} -.003 & -.028 & 13.19 \\ -.003 & -.008 & -29.2 \\ 2.97 & 56.38 & -9999 \end{bmatrix}$$

and the **left** image point

$$\tilde{u}_l = \begin{bmatrix} 343 \\ 221 \\ 1 \end{bmatrix}$$

Left Image



Right Image

The equation for the epipolar line in the **right** image is:

$$\begin{bmatrix} u_r & v_r & 1 \end{bmatrix} \begin{bmatrix} -.003 & -.003 & 2.97 \\ -.028 & -.008 & 56.38 \\ 13.19 & -29.2 & -9999 \end{bmatrix} \begin{bmatrix} 343 \\ 221 \\ 1 \end{bmatrix} = 0$$

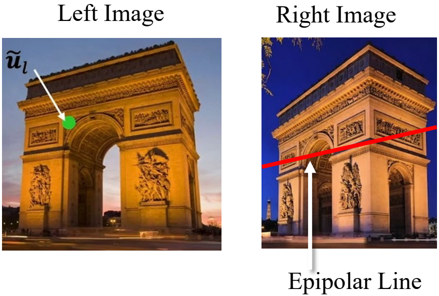
Finding Epipolar Lines: Example

Given the Fundamental matrix,

$$F = \begin{bmatrix} -.003 & -.028 & 13.19 \\ -.003 & -.008 & -29.2 \\ 2.97 & 56.38 & -9999 \end{bmatrix}$$

and the **left** image point

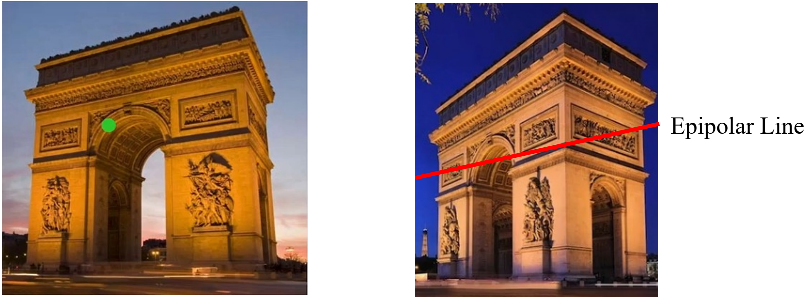
$$\tilde{u}_l = \begin{bmatrix} 343 \\ 221 \\ 1 \end{bmatrix}$$



The equation for the epipolar line in the **right** image is:

$$.03u_r + .99v_r - 265 = 0$$

Finding Correspondence

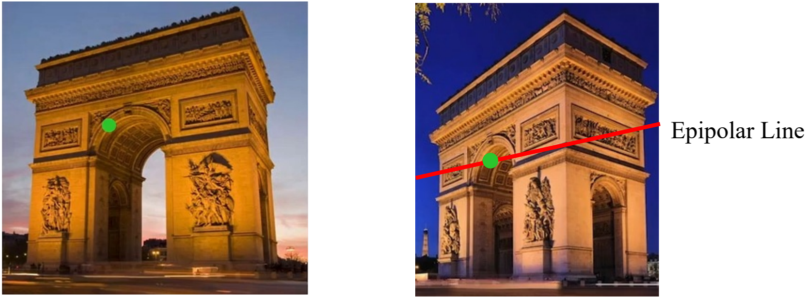


Left Image

Right Image

Corresponding scene points lie on the epipolar lines.
Finding correspondence is a **1D search**.

Finding Correspondence

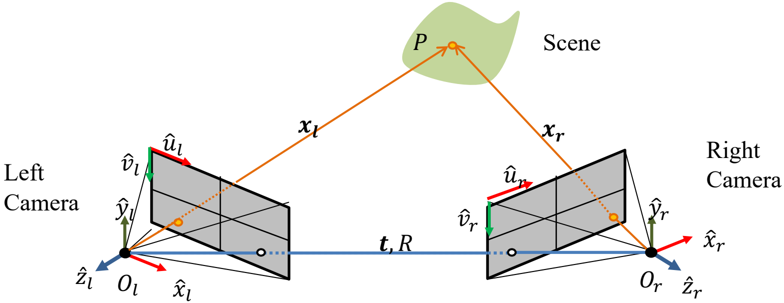


Left Image

Right Image

Corresponding scene points lie on the epipolar lines.
Finding correspondence is a **1D search**.

Uncalibrated Stereo



✓ 1. Assume Camera Matrix K is known for each camera

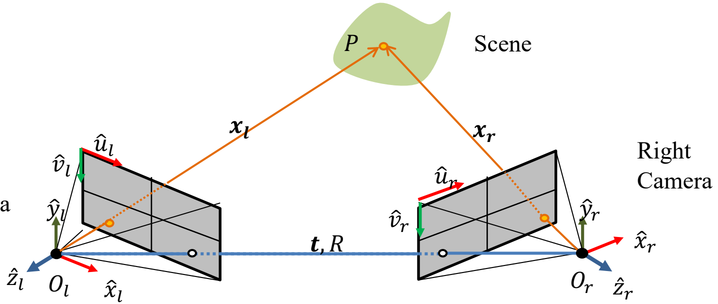
✓ 2. Find a few Reliable Corresponding Points

✓ 3. Find Relative Camera Position t and Orientation R

✓ 4. Find Dense Correspondence

5. Compute Depth using Triangulation

Computing Depth



Given the intrinsic parameters, the projections of scene point on the two image sensors are:

$$\begin{bmatrix}u_l\\v_l\\1\end{bmatrix} \equiv \begin{bmatrix} f_x^{(l)} & 0 & o_x^{(l)} & 0\\ 0 & f_y^{(l)} & o_y^{(l)} & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x_l\\y_l\\z_l\\1\end{bmatrix} \qquad \begin{bmatrix}u_r\\v_r\\1\end{bmatrix} \equiv \begin{bmatrix} f_x^{(r)} & 0 & o_x^{(r)} & 0\\ 0 & f_y^{(r)} & o_y^{(r)} & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x_r\\y_r\\z_r\\1\end{bmatrix}$$

Computing Depth

Left Camera Imaging Equation

$$\begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f_x^{(l)} & 0 & o_x^{(l)} & 0 \\ 0 & f_y^{(l)} & o_y^{(l)} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ z_l \\ 1 \end{bmatrix}$$

Right Camera Imaging Equation

$$\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f_x^{(r)} & 0 & o_x^{(r)} & 0 \\ 0 & f_y^{(r)} & o_y^{(r)} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}$$

We also know the relative position and orientation between the two cameras.

$$\begin{bmatrix} x_l \\ y_l \\ z_l \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}$$

Computing Depth

Left Camera Imaging Equation:

$$\begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f_x^{(l)} & 0 & o_x^{(l)} & 0 \\ 0 & f_y^{(l)} & o_y^{(l)} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}$$

$$\boxed{\widetilde{\boldsymbol{u}_l} = P_l \widetilde{\boldsymbol{x}_r}}$$

Right Camera Imaging Equation:

$$\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f_x^{(r)} & 0 & o_x^{(r)} & 0 \\ 0 & f_y^{(r)} & o_y^{(r)} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}$$

$$\boxed{\widetilde{\boldsymbol{u}_r} = M_{int_r} \widetilde{\boldsymbol{x}_r}}$$

Computing Depth

The imaging equation:

$$\widetilde{\boldsymbol{u}_r} = M_r \widetilde{\boldsymbol{x}_r} \qquad \widetilde{\boldsymbol{u}_l} = P_l \widetilde{\boldsymbol{x}_r}$$

$$\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} \equiv \underbrace{\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}}_{\text{Known}} \underbrace{\begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}}_{\text{Unknown}} \qquad \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} \equiv \underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_{\text{Known}} \underbrace{\begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}}_{\text{Unknown}}$$

Rearranging the terms:

$$\begin{bmatrix} u_r m_{31} - m_{11} & u_r m_{32} - m_{12} & u_r m_{33} - m_{13} \\ v_r m_{31} - m_{21} & v_r m_{32} - m_{22} & v_r m_{33} - m_{23} \\ u_l p_{31} - p_{11} & u_l p_{32} - p_{12} & u_l p_{33} - p_{13} \\ v_l p_{31} - p_{21} & v_l p_{32} - p_{22} & v_l p_{33} - p_{23} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} m_{14} - m_{34} \\ m_{24} - m_{34} \\ p_{14} - p_{34} \\ p_{24} - p_{34} \end{bmatrix}$$

Computing Depth: Least Squares Solution

$$\underset{\substack{A_{4\times 3}\\ \text{(Known)}}}{\boxed{\begin{bmatrix} u_r m_{31} - m_{11} & u_r m_{32} - m_{12} & u_r m_{33} - m_{13} \\ v_r m_{31} - m_{21} & v_r m_{32} - m_{22} & v_r m_{33} - m_{23} \\ u_l p_{31} - p_{11} & u_l p_{32} - p_{12} & u_l p_{33} - p_{13} \\ v_l p_{31} - p_{21} & v_l p_{32} - p_{22} & v_l p_{33} - p_{23} \end{bmatrix}}} \underset{\substack{\boldsymbol{x_r}\\ \text{(Unknown)}}}{\boxed{\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix}}} = \underset{\substack{\boldsymbol{b}_{4\times 1}\\ \text{(Known)}}}{\boxed{\begin{bmatrix} m_{14} - m_{34} \\ m_{24} - m_{34} \\ p_{14} - p_{34} \\ p_{24} - p_{34} \end{bmatrix}}}$$

Find **least squares solution** using **pseudo-inverse**:

$$A\boldsymbol{x}_r = \boldsymbol{b}$$

$$A^T A\boldsymbol{x}_r = A^T\boldsymbol{b}$$

$$\boxed{\boldsymbol{x}_r = (A^T A)^{-1} A^T \boldsymbol{b}}$$

3D Reconstruction with Internet Images

St. Peter's Basilica (1275 Images)

[Snavely 2006]

3D Reconstruction with Internet Images

St. Peter's Basilica (1275 Images)

[Snavely 2006]

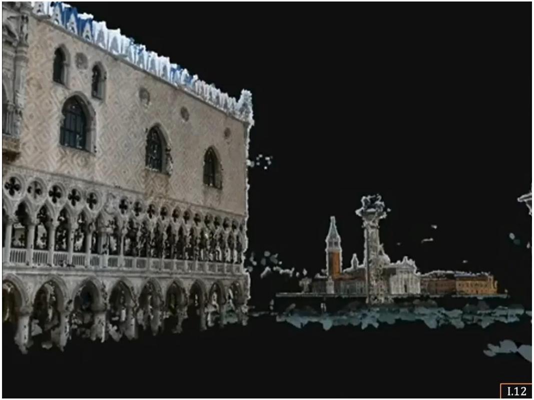
3D Reconstruction with Internet Images

Piazza San Marco (13709 Images)

[Furukawa 2010]

3D Reconstruction with Internet Images

Piazza San Marco (13709 Images)



[Furukawa 2010]

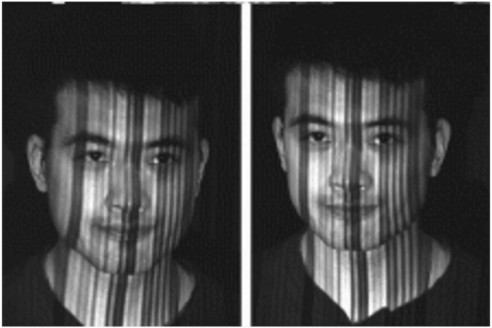
Active Stereo Results

Left Image Right Image

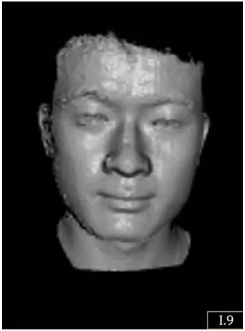
3D Structure

[Zhang 2003]

Active Stereo Results



Left Image Right Image



3D Structure

[Zhang 2003]

中国科学技术大学
University of Science and Technology of China

Optical Flow

张举勇
中国科学技术大学

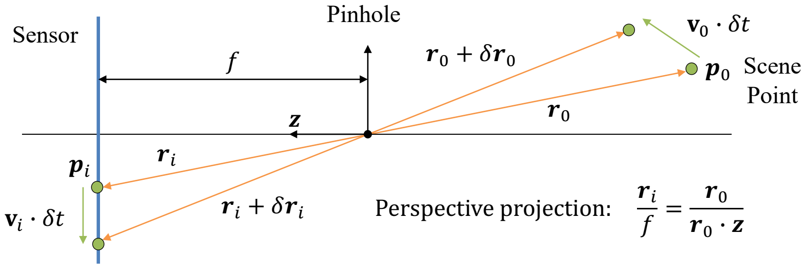
Overview

Method to estimate apparent motion of scene points from a sequence of images

Topics:

(1) Motion Field and Optical Flow
(2) Optical Flow Constraint Equation
(3) Lucas-Kanada Method
(4) Coarse-to-Fine Flow Estimation
(5) Applications of Optical Flow

Motion Field



Perspective projection: $\frac{\mathbf{r}_i}{f} = \frac{\mathbf{r}_0}{\mathbf{r}_0 \cdot \mathbf{z}}$

Image Point Velocity: $\mathbf{v}_i = \frac{d\mathbf{r}_i}{dt} = f\frac{(\mathbf{r}_0 \cdot \mathbf{z})\mathbf{v}_0 - (\mathbf{v}_0 \cdot \mathbf{z})\mathbf{r}_0}{(\mathbf{r}_0 \cdot \mathbf{z})^2}$

(Motion Field)

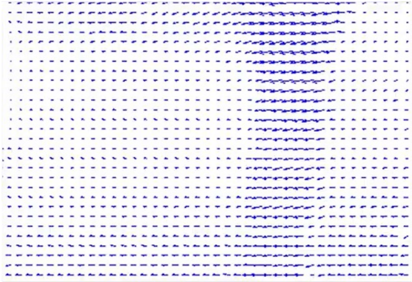
Scene Point Velocity: $\mathbf{v}_0 = \frac{d\mathbf{r}_0}{dt}$

$$\boxed{\mathbf{v}_i = \frac{(\mathbf{r}_0 \times \mathbf{v}_0) \times \mathbf{z}}{(\mathbf{r}_0 \cdot \mathbf{z})^2}}$$

Optical Flow

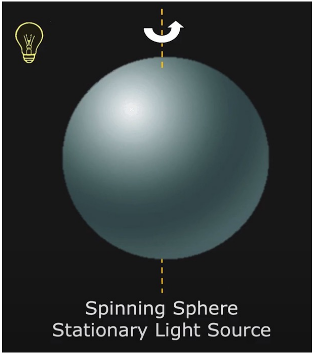
Motion of brightness patterns in the image

Image Sequence
(2 frames)

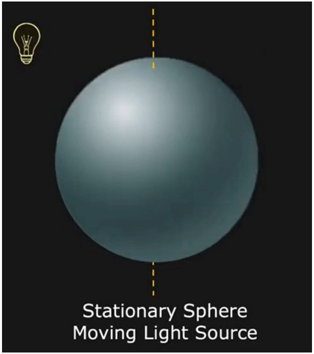


Optical Flow

When is Optical Flow ≠ Motion Field ?



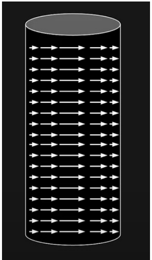
Motion Field exists
But no Optical Flow



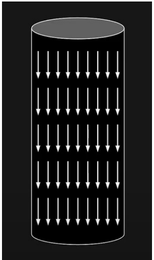
No Motion Field exists
But there is Optical Flow

When is Optical Flow ≠ Motion Field ?

Barber Pole Illusion

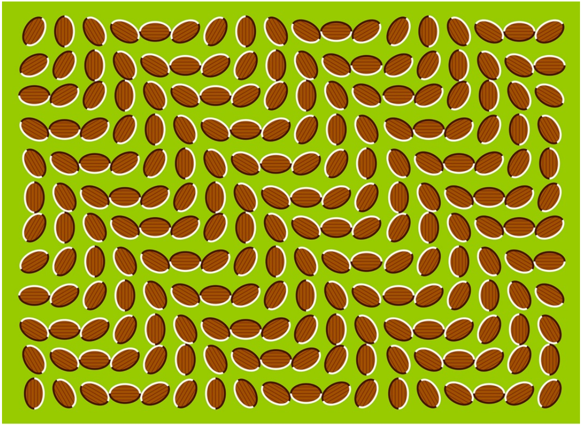


Motion Field



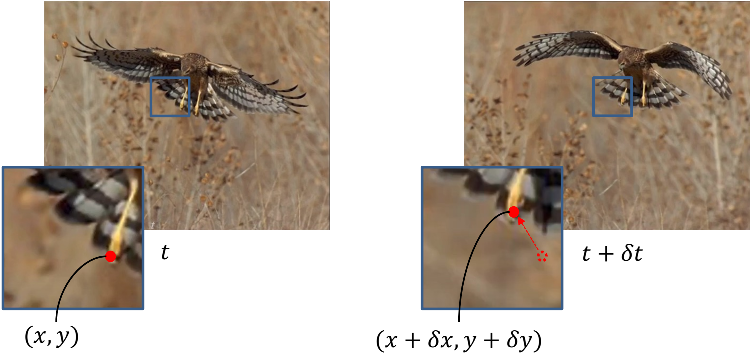
Optical Flow

Motion Illusions



“Dongurakokko” (Donguri wave), produced by Akiyoshi Kitaoka in 2004 as an artwork of waving demonstration of the ‘optimized’ Fraser-Wilcox illusion Type IIa. Fermüller, C., Ji, H., and Kitaoka, A. (2010). Illusory motion due to causal time filtering. Vision Research, 50, 315-329.

Optical Flow



Displacement: $(\delta x, \delta y)$

Optical Flow: $(u, v) = (\frac{\delta x}{\delta t}, \frac{\delta y}{\delta t})$

Optical Flow Constraint Equation

$I(x, y, t)$

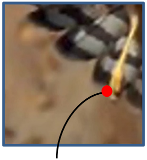
$I(x + \delta x, y + \delta y, t + \delta t)$

Assumption #1：

Brightness of image point remains constant over time

$$\boxed{I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t)}$$

Optical Flow Constraint Equation



$I(x, y, t)$

$I(x + \delta x, y + \delta y, t + \delta t)$

Assumption #2：

Dispacement $(\delta x, \delta y)$ and time step δt are small

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t$$

$$\boxed{I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + I_x\delta x + I_y\delta y + I_t\delta t}$$

Optical Flow Constraint Equation

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) \quad \text{------------- (1)}$$

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + I_x \delta x + I_y \delta y + I_t \delta t \quad \text{------------- (2)}$$

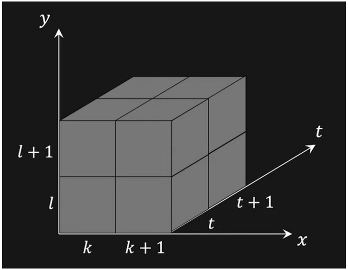
Subtract (1) from (2)： $I_x \delta x + I_y \delta y + I_t \delta t = 0$

Divide by δt and take limit as $\delta t \to 0$ ： $I_x \frac{\partial x}{\partial \mathrm{t}} + I_y \frac{\partial y}{\partial t} + I_t = 0$

Constraint Equation： $\boxed{I_x u + I_y v + I_t = 0}$ (u, v): Optical Flow

(I_x, I_y, I_t) can be easily computed from two frames

Computing Partial Derivatives I_x, I_y, I_t



$I_x(k,l,t)$

$$= \frac{1}{4}[I(k+1,l,t) + I(k+1,l+1,t) + I(k+1,l,t+1) + I(k+1,l+1,t+1)]$$

$$- \frac{1}{4}[I(k,l,t) + I(k,l+1,t) + I(k,l,t+1) + I(k,l+1,t+1)]$$

Similarly find $I_y(k,l,t)$ and $I_t(k,l,t)$

Geometric interpretation

For any point (x, y) in the image, its optical flow (u, v) lies on the line:

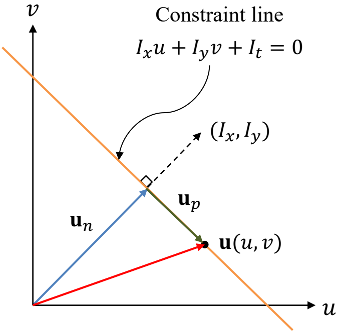
$$I_x u + I_y v + I_t = 0$$

Optical Flow can be split into two components.

$$\boldsymbol{u} = \boldsymbol{u}_n + \boldsymbol{u}_p$$

$\boldsymbol{u}_n$: Normal Flow

$\boldsymbol{u}_p$: Parallel Flow



Normal Flow

Direction of Normal Flow：

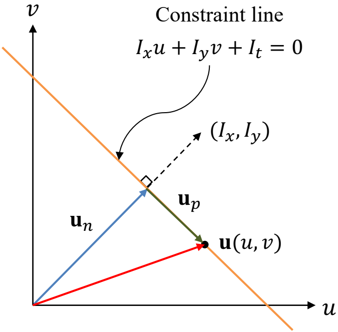
Unit vector perpendicular to the constraint line:

$$\hat{\boldsymbol{u}}_n = \frac{(I_x, I_y)}{\sqrt{I_x^2 + I_y^2}}$$

Magnitude of Normal Flow：

Distance of origin from the constant line:

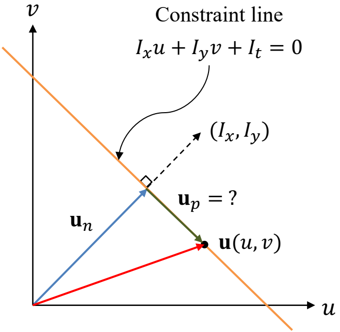
$$|\boldsymbol{u}_n| = \frac{|I_t|}{\sqrt{I_x^2 + I_y^2}}$$



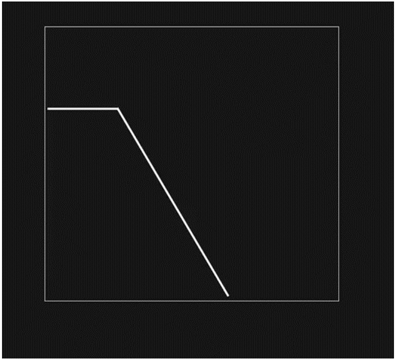
$$\boldsymbol{u}_n = \frac{|I_t|}{(I_x^2 + I_y^2)}(I_x, I_y)$$

Parallel Flow

We can not determine u_p, the optical flow component parallel to the constraint line.



Aperture Problem



Locally, we can only determine Normal Flow!

Optical Flow is Under constrained

Constraint Equation： $\boxed{I_x u + I_y v + I_t = 0}$

2 unknowns, 1 equation.

Lucas-Kanada Solution

Assumption： For each pixel, assume Motion Field, and hence Optical Flow (u, v), is constant within a small neighbourhood W.

That is for all points $(k, l) \in W$：

$$I_x(k,l)u + I_y(k,l)v + I_t(k,l) = 0$$

Lucas-Kanada Solution

For all points $(k,l) \in W$: $I_x(k,l)u + I_y(k,l)v + I_t(k,l) = 0$

Let the size of window W be $n \times n$

In matrix form：

$$\underset{\substack{A\\ \text{(Known)}\\ n^2\times 2}}{\boxed{\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ \vdots & \vdots \\ I_x(k,l) & I_y(k,l) \\ \vdots & \vdots \\ I_x(n,n) & I_y(n,n) \end{bmatrix}}} \underset{\substack{\boldsymbol{u}\\ \text{(Unknown)}\\ 2\times 1}}{\boxed{\begin{bmatrix} \mathrm{u} \\ v \end{bmatrix}}} = \underset{\substack{B\\ \text{(Known)}\\ n^2\times 1}}{\boxed{-\begin{bmatrix} I_t(1,1) \\ \vdots \\ I_t(k,l) \\ \vdots \\ I_t(n,n) \end{bmatrix}}}$$

n^2 Equations, 2 Unknowns： Find Least Squares Solution

When Dose Optical Flow Estimation Work?

$\boxed{Au = B}$ $\boxed{A^T Au = A^T B}$

- $A^T A$ must be invertible. That is $\det(A^T A) \neq 0$
- $A^T A$ must be well-conditioned.

If λ_1 and λ_2 are eigen values of $A^T A$, then

$\lambda_1 > \epsilon$ and $\lambda_2 > \epsilon$

$\lambda_1 \geq \lambda_2$ but not $\lambda_1 \gg \lambda_2$

Smooth Regions (Bad)

$\lambda_1 \sim \lambda_2$

Both are small

Equations for all pixels in window are both more or less the same

Cannot reliably compute flow!

Edges (Bad)

$\lambda_1 \gg \lambda_2$

Badly conditioned. Prominent gradient in one direction.

Cannot reliably compute flow!

Same as Aperture Problem.

Textured Regions (Good)

$\lambda_1 \sim \lambda_2$

Both are Large

Well conditioned. Large and diverse gradient magnitudes.

Can reliably compute optical flow!

What if we have Large Motion?

Taylor Series approximation of

$I(x + \delta x, y + \delta y, t + \delta t)$ is not valid

Our simple linear constraint equation not valid

$$I_x u + I_y v + I_t \neq 0$$

Large Motion: Coarse-to-Fine Estimation

Resolution
$N \times N$

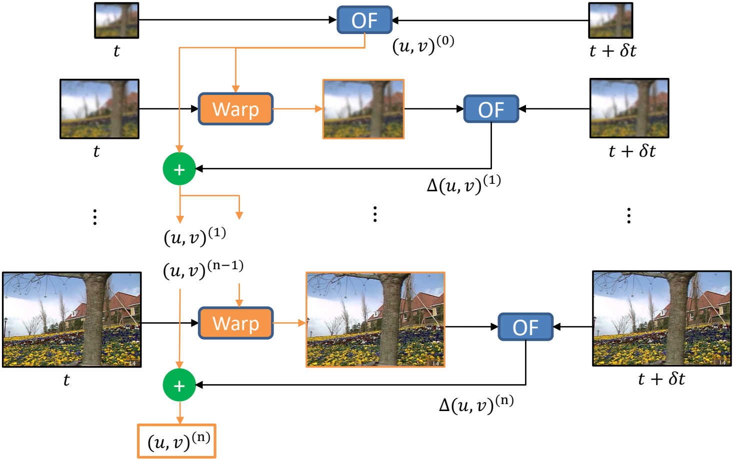
Resolution
$\frac{N}{2} \times \frac{N}{2}$

Resolution
$\frac{N}{4} \times \frac{N}{4}$

Resolution
$\frac{N}{8} \times \frac{N}{8}$

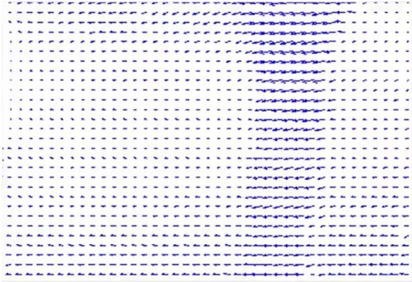
At lowest resolution, motion ≤ 1 pixel

Coarse-to-Fine Estimation Algorithm



Results: Tree Sequence

Image Sequence



Optical Flow

Results: Rotating Ball

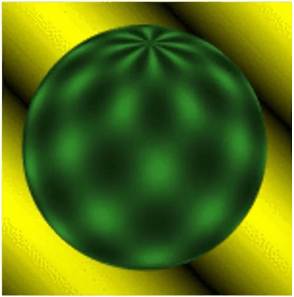
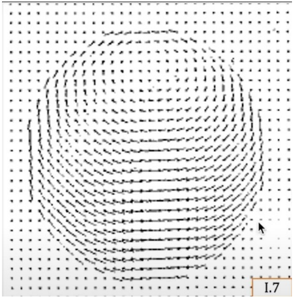


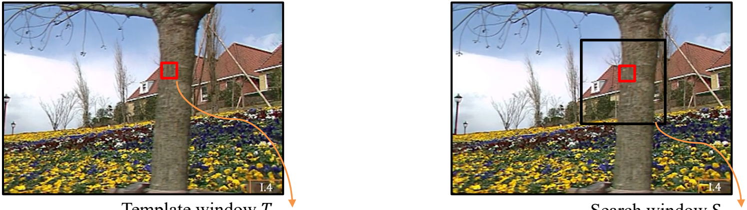
Image Sequence



Optical Flow

Alternative Approach: Template Matching

Determine Flow using Template Matching



Template window T

Search window S

Image I_1 at time t

Image I_2 at time $t + \delta t$

For each template window T in image I_1,
find the corresponding match in image I_2

Alternative Approach: Template Matching

Determine Flow using Template Matching

Template window T

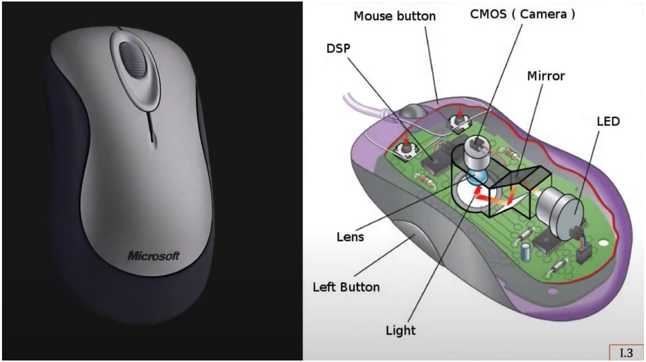
Image I_1 at time t

Search window S

Image I_2 at time $t + \delta t$

1. Template Matching is slow when search window S is large.
2. Also mismatches are possible

Applications: Optical Mouse

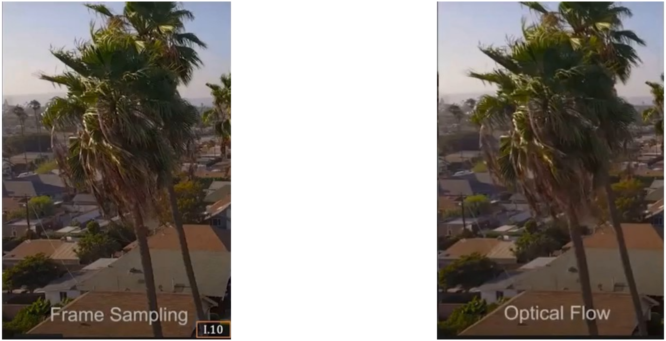


Estimating Mouse Movements

Applications: Traffic Monitoring

Finding Velocities of Vehicles

Applications: Video Retiming

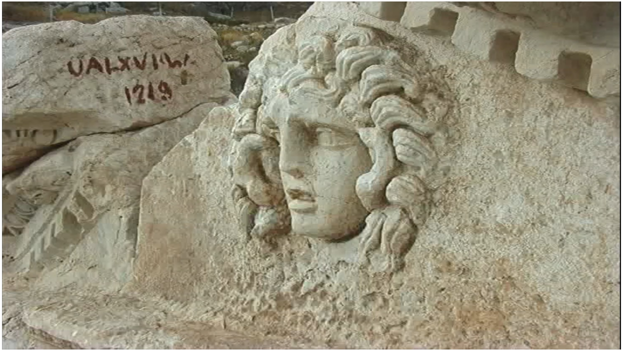


Optical Flow is used to determine the intermediate frames to produce slow-motion effect.

Struction from Motion

张举勇
中国科学技术大学

Uncontrolled (Casual) Video



Overview

Compute 3D scene structure and camera motion from a sequence of frames.

Topics:

(1) Structure from Motion Problem

(2) SFM Observation Matrix

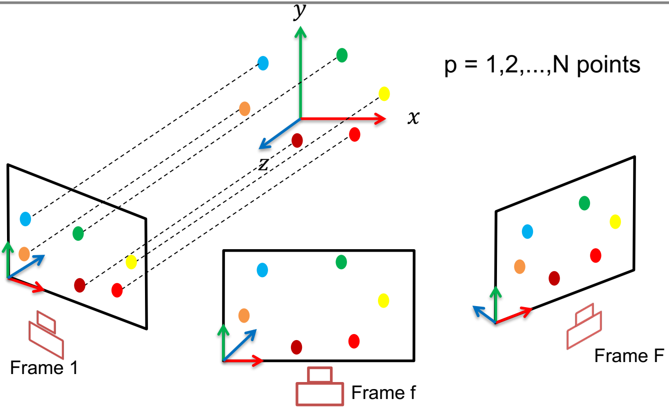
(3) Rank of Observation Matrix

(4) Tomasi-Kanade Factorization

Feature Detection and Tracking

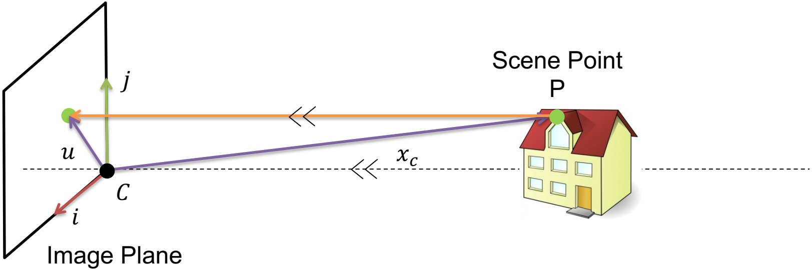
- Detect feature points: Corners , SIFT points , ...
- Track feature points: Template Matching, Optical Flow...

Orthographic Structure from Motion



Given sets of corresponding image points (2D): $(u_{f,p}, v_{f,p})$
Find scene points (3D) P_p, assuming orthographic camera.

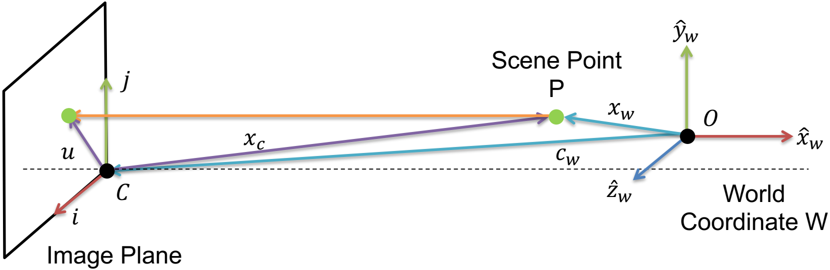
From 3D to 2D: Orthographic Projection



$$u = i \cdot x_c = i^T x_c$$
$$v = j \cdot x_c = j^T x_c$$

Perspective cameras exhibit orthographic projection when distance of scene from camera is large compared to depth variation within scene (magnification is nearly constant).

From 3D to 2D: Orthographic Projection

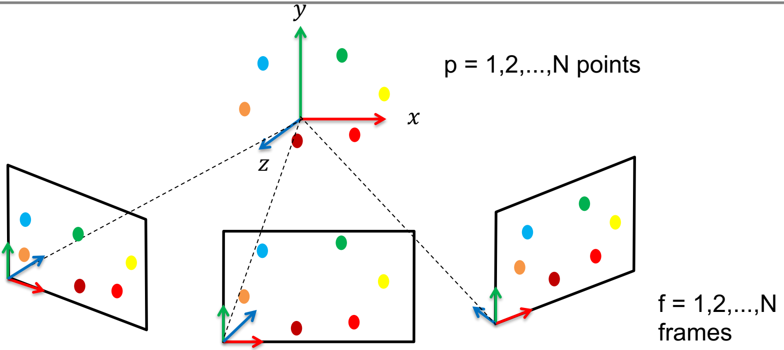


$$u = i^T x_c = i^T(x_w - c_w) = i^T(P - C)$$

$$v = j^T x_c = j^T(x_w - c_w) = j^T(P - C)$$

$$\boxed{\begin{aligned} u &= i^T(P - C) \\ v &= j^T(P - C) \end{aligned}}$$

Orthographic SFM



Given corresponding image points (2D) $(u_{f,p}, v_{f,p})$

Find scene points $\{P_p\}$.

Camera Positions $\{C_f\}$, camera orientations $\{(i_f, j_f)\}$ are unknown.

Orthographic SFM

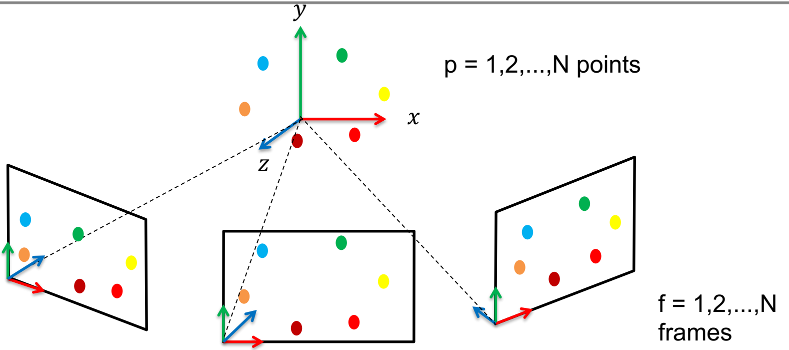


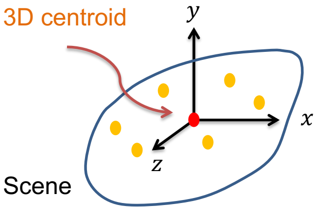
Image of point P in camera frame f :

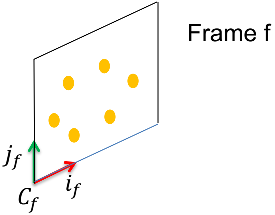
$$\underline{u_{f,p}} = \underline{i_f^T\,(P_p - C_f)}$$
$$\underline{v_{f,p}} = \underline{j_f^T\,(P_p - C_f)}$$

Known Unknown

We can remove C from equations to simply SFM problem.

Centering Trick



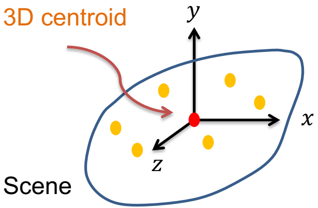


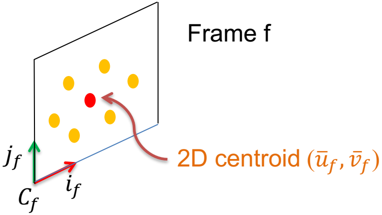
Assume origin of world at centroid of scene points:

$$\frac{1}{N}\sum_{p=1}^{N} P_p = \bar{P} = 0$$

We will compute scene points w.r.t their centroid!

Centering Trick





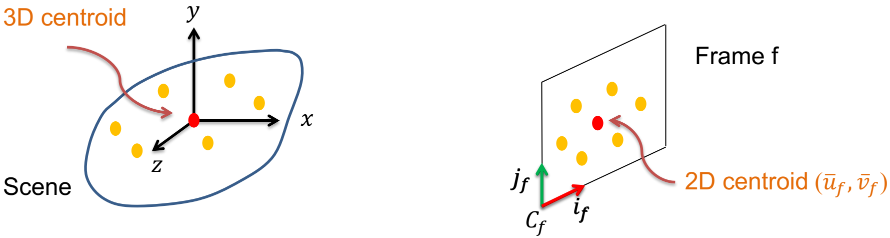
Centroid $(\bar{u}_f, \bar{v}_f)$ of the image points in frame f :

$$\bar{u}_f = \frac{1}{N} \sum_{p=1}^{N} u_{f,p} = \frac{1}{N} \sum_{p=1}^{N} i_f^T \, (P_p - C_f) \qquad \bar{v}_f = \frac{1}{N} \sum_{p=1}^{N} v_{f,p} = \frac{1}{N} \sum_{p=1}^{N} j_f^T \, (P_p - C_f)$$

$$\bar{u}_f = \cancel{\frac{1}{N} i_f^T \sum_{p=1}^{N} P_p} - \frac{1}{N} \sum_{p=1}^{N} i_f^T \, C_f \qquad \bar{v}_f = \cancel{\frac{1}{N} j_f^T \sum_{p=1}^{N} P_p} - \frac{1}{N} \sum_{p=1}^{N} j_f^T \, C_f$$

$$\boxed{\bar{u}_f = -i_f^T \, C_f} \qquad \boxed{\bar{v}_f = -j_f^T \, C_f}$$

Centering Trick



Shift camera origin to the centroid $(\bar{u}_f, \bar{v}_f)$.

Image points w.r.t. $(\bar{u}_f, \bar{v}_f)$:

$$\tilde{u}_{f,p} = u_{f,p} - \bar{u}_f = i_f^T \left(P_p - C_f\right) + i_f^T\, C_f \qquad \tilde{v}_{f,p} = v_{f,p} - \bar{v}_f = j_f^T \left(P_p - C_f\right) + j_f^T\, C_f$$

$$\boxed{\tilde{u}_{f,p} = i_f^T\, P_p} \qquad \boxed{\tilde{v}_{f,p} = j_f^T\, P_p}$$

Camera locations C_f now removed from equations.

Observation Matrix W

$$\begin{aligned} \tilde{u}_{f,p} &= i_f^T \, P_p \\ \tilde{v}_{f,p} &= j_f^T \, P_p \end{aligned} \quad \Longrightarrow \quad \begin{bmatrix} \tilde{u}_{f,p} \\ \tilde{u}_{f,p} \end{bmatrix} = \begin{bmatrix} i_f^T \\ j_f^T \end{bmatrix} P_p$$

$$\left.\begin{array}{l|cccc|}
 & \text{Point 1} & \text{Point 2} & & \text{Point N} \\
\text{Image 1} & \tilde{u}_{1,1} & \tilde{u}_{1,2} & \cdots & \tilde{u}_{1,N} \\
\text{Image 2} & \tilde{u}_{2,1} & \tilde{u}_{2,2} & \cdots & \tilde{u}_{2,N} \\
 & \vdots & \vdots & \vdots & \vdots \\
\text{Image F} & \tilde{u}_{F,1} & \tilde{u}_{F,2} & \cdots & \tilde{u}_{F,N} \\
\hdashline
\text{Image 1} & \tilde{v}_{1,1} & \tilde{v}_{1,2} & \cdots & \tilde{v}_{1,N} \\
\text{Image 2} & \tilde{v}_{2,1} & \tilde{v}_{2,2} & \cdots & \tilde{v}_{2,N} \\
 & \vdots & \vdots & \vdots & \vdots \\
\text{Image F} & \tilde{v}_{F,1} & \tilde{v}_{F,2} & \cdots & \tilde{v}_{F,N}
\end{array}\right.
=
\begin{bmatrix} i_1^T \\ i_2^T \\ \vdots \\ i_N^T \\ \hdashline j_1^T \\ j_2^T \\ \vdots \\ j_N^T \end{bmatrix}
\begin{array}{cccc}
\text{Point 1} & \text{Point 2} & & \text{Point N} \\
\left[P_1 \right. & P_2 & \cdots & \left. P_N \right]
\end{array}$$

$W_{2F\times N}$

Centroid-Subtracted Feature Points (Known)

$M_{2F\times 3}$

Camera Motion (Unknown)

$S_{3\times N}$

Scene Struction (Unknown)

Observation Matrix W

$$\left[\begin{array}{c|cccc} & \text{Point 1} & \text{Point 2} & & \text{Point N} \\ \hline \text{Image 1} & \tilde{u}_{1,1} & \tilde{u}_{1,2} & \cdots & \tilde{u}_{1,N} \\ \text{Image 2} & \tilde{u}_{2,1} & \tilde{u}_{2,2} & \cdots & \tilde{u}_{2,N} \\ & \vdots & \vdots & \vdots & \vdots \\ \text{Image F} & \tilde{u}_{F,1} & \tilde{u}_{F,2} & \cdots & \tilde{u}_{F,N} \\ \hdashline \text{Image 1} & \tilde{v}_{1,1} & \tilde{v}_{1,2} & \cdots & \tilde{v}_{1,N} \\ \text{Image 2} & \tilde{v}_{2,1} & \tilde{v}_{2,2} & \cdots & \tilde{v}_{2,N} \\ & \vdots & \vdots & \vdots & \vdots \\ \text{Image F} & \tilde{v}_{F,1} & \tilde{v}_{F,2} & \cdots & \tilde{v}_{F,N} \end{array}\right] = \left[\begin{array}{c} i_1^T \\ i_2^T \\ \vdots \\ i_N^T \\ \hdashline j_1^T \\ j_2^T \\ \vdots \\ j_N^T \end{array}\right] \begin{array}{cccc} \text{Point 1} & \text{Point 2} & & \text{Point N} \\ \left[P_1\right. & P_2 & \cdots & \left.P_N\right] \end{array}$$

$S_{3\times N}$

Scene Struction (Unknown)

$W_{2F\times N}$

Centroid-Subtracted Feature Points (Known)

$M_{2F\times 3}$

Camera Motion (Unknown)

Can we find M and s from W ?

Linear Independence of Vectors

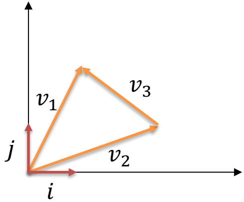
A set of vectors $\{v_1, v_2, \dots\ v_n\}$ is said to be linearly independent if no vector can be represented as a weighted linear sum of the others.

$\{i, j\}$ is linearly independent.

$\{i, j, v_1\}$ is linearly dependent.

$\{i, j, v_3\}$ is linearly dependent.

$\{v_1, v_2, v_3\}$ is linearly dependent.



Rank of a Matrix

- Column Rank: The number of linearly independent columns of the matrix.
- Row Rank: The number of linearly independent rows of the matrix.

$$m\underset{n}{\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix}} = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} = \begin{bmatrix} r_1^T \\ r_2^T \\ \vdots \\ r_n^T \end{bmatrix}$$

$ColumnRank(\mathrm{A}) \leq \mathrm{n}$ $\qquad$ $ColumnRank(\mathrm{A}) \leq \mathrm{m}$

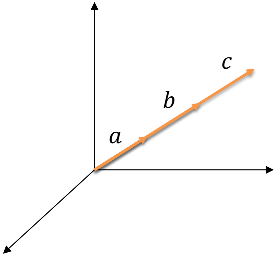
$$\boxed{\begin{gathered} ColumnRank(\mathrm{A}) = RowRank(\mathrm{A}) = Rank(\mathrm{A}) \\ Rank(\mathrm{A}) \leq \min(\mathrm{m},\mathrm{n}) \end{gathered}}$$

Geometric Meaning of Matrix Rank

Rank is the dimensionality of the space spanned by column or row vectors of the matrix.

$$A = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} = [\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]$$

$Rank(\mathrm{A}) = 1$ ⟺

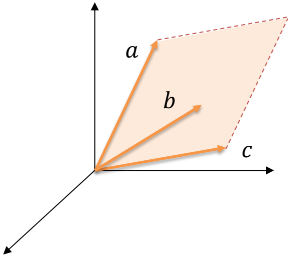


Geometric Meaning of Matrix Rank

Rank is the dimensionality of the space spanned by column or row vectors of the matrix.

$$A = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} = [\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]$$

$Rank(\mathrm{A}) = 2$ ⟺

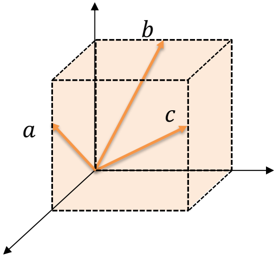


Geometric Meaning of Matrix Rank

Rank is the dimensionality of the space spanned by column or row vectors of the matrix.

$$A = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} = [\boldsymbol{a} \quad \boldsymbol{b} \quad \boldsymbol{c}]$$

$Rank$(A) =3 ⟺



Important Properties of Matrix Rank

- $Rank\ (A^T) = Rank(A)$

- $Rank(A_{m\times n}\ B_{n\times p}\ ) \ = \min\ (Rank(A_{m\times n}), Rank(B_{n\times p}))$
  $\leq \min(\mathrm{m, n, p})$

- $Rank(A\ A^T) = Rank(A^T A) = Rank(A) = Rank(A^T)$

- $A_{m\times m}$ is invertible iff $Rank(A_{m\times m}\ ) = \mathrm{m}$

...Back to Observation Matrix W

$$
\begin{array}{r|cccc|}
 & \text{Point 1} & \text{Point 2} & & \text{Point N} \\
\text{Image 1} & \tilde{u}_{1,1} & \tilde{u}_{1,2} & \cdots & \tilde{u}_{1,N} \\
\text{Image 2} & \tilde{u}_{2,1} & \tilde{u}_{2,2} & \cdots & \tilde{u}_{2,N} \\
 & \vdots & \vdots & \vdots & \vdots \\
\text{Image F} & \tilde{u}_{F,1} & \tilde{u}_{F,2} & \cdots & \tilde{u}_{F,N} \\
\hdashline
\text{Image 1} & \tilde{v}_{1,1} & \tilde{v}_{1,2} & \cdots & \tilde{v}_{1,N} \\
\text{Image 2} & \tilde{v}_{2,1} & \tilde{v}_{2,2} & \cdots & \tilde{v}_{2,N} \\
 & \vdots & \vdots & \vdots & \vdots \\
\text{Image F} & \tilde{v}_{F,1} & \tilde{v}_{F,2} & \cdots & \tilde{v}_{F,N}
\end{array}
=
\begin{bmatrix} i_1^T \\ i_2^T \\ \vdots \\ i_N^T \\ \hdashline j_1^T \\ j_2^T \\ \vdots \\ j_N^T \end{bmatrix}
\begin{array}{cccc}
\text{Point 1} & \text{Point 2} & & \text{Point N} \\
\left[P_1 \right. & P_2 & \cdots & \left. P_N \right]
\end{array}
$$

$W_{2F\times N}$

Centroid-Subtracted Feature Points (Known)

$M_{2F\times 3}$

Camera Motion (Unknown)

$S_{3\times N}$

Scene Struction (Unknown)

Rank of Observation Matrix

$$\underset{2F\times N}{W} = \underset{2F\times 3}{M} \times \underset{3\times N}{S}$$

We know:

$$Rank(MS) \leq Rank(M) \qquad Rank(MS) \leq Rank(S)$$

⇨ $$Rank(MS) \leq min(3{,}2F) \qquad Rank(MS) \leq min(3, N)$$

⇨ $$Rank(W) = Rank(MS) \leq min(3, N, 2F)$$

Rank throem : $Rank(W) \leq 3$

We can “factorize” W into M and S!

Singular Value Decomposition (SVD)

For any matrix A there exists a factorization:

$$A_{M\times N} = U_{M\times M} \cdot \Sigma_{M\times N} \cdot V_{N\times N}^{T}$$

Where $U_{M\times M}$and $V_{N\times N}^{T}$ are orthonormal and $\Sigma_{M\times N}$ is diagonal.

Mathlab : [U,S,V] = svd(A)

$$\Sigma_{M\times N} = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sigma_4 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & \sigma_N \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{bmatrix}$$

$\sigma_1, \ldots, \sigma_N$: Singular Values

If *Rank*(A) = r then A has r non-zero singular values.

Enforcing Rank Constraint

Using SVD:

$$W = U \Sigma V^T$$

$$= \left[\begin{matrix} \\ \\ & U & \\ \\ \\ \end{matrix} \right]_{2F\times 2F} \left[\begin{matrix} \sigma_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sigma_4 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & \sigma_N \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{matrix} \right]_{2F\times N} \left[\begin{matrix} \\ \\ & V^T & \\ \\ \\ \end{matrix} \right]_{N\times N}$$

Where: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_N$ are the singular values of Σ.

Enforcing Rank Constraint

Using SVD:

$$W = U\Sigma V^{T}$$

$$= \begin{bmatrix} & & \\ & U & \\ & & \end{bmatrix}_{2F\times 2F} \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{bmatrix}_{2F\times N} \begin{bmatrix} & & \\ & V^{T} & \\ & & \end{bmatrix}_{N\times N}$$

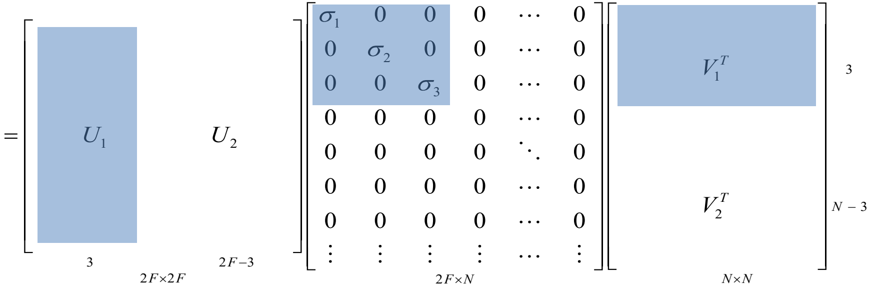
Since $Rank(W) \leq 3$, $Rank(\Sigma) \leq 3$.

All expect first 3 diagonal elements of Σ must be 0.

Enforcing Rank Constraint

Using SVD:

$$W = U\Sigma V^T$$



$$= \underset{2F\times 2F}{\begin{array}{cc} \left[\begin{matrix} U_1 & U_2 \end{matrix}\right] \\ \begin{matrix} 3 & 2F-3 \end{matrix} \end{array}} \underset{2F\times N}{\begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{bmatrix}} \underset{N\times N}{\begin{array}{cc} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} & \begin{matrix} 3 \\ N-3 \end{matrix} \end{array}}$$

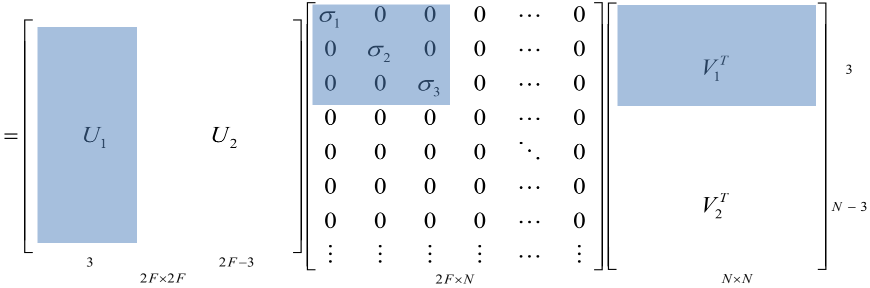
Since $Rank(W) \leq 3$, $Rank(\Sigma) \leq 3$.

Submatrices U_2 and V_2^T do not contribute to W.

Enforcing Rank Constraint

Using SVD:

$$W = U \Sigma V^T$$



$$= \underbrace{\begin{array}{cc} \left[\begin{array}{cc} U_1 & U_2 \end{array}\right] \\ 3 \quad 2F-3 \end{array}}_{2F\times 2F} \underbrace{\left[\begin{array}{cccccc} \sigma_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{array}\right]}_{2F\times N} \underbrace{\begin{array}{cc} \left[\begin{array}{c} V_1^T \\ V_2^T \end{array}\right] & \begin{array}{c} 3 \\ N-3 \end{array} \end{array}}_{N\times N}$$

$$\boxed{W = U_1 \; \Sigma_1 \; V_1^T}$$

$$(2F\times 3)(3\times 3)(3\times P)$$

Factorization (Finding M , S)

$$W = \underbrace{U_1 (\Sigma_1)^{\frac{1}{2}}}_{(2F\times 3)} \underbrace{(\Sigma_1)^{\frac{1}{2}} V_1^T}_{(3\times N)}$$

$= M$?　　$= S$?

Not so fast. Decomposition not unique!

For any 3X3 non-singular matrix Q :

$$W = \underbrace{U_1 (\Sigma_1)^{\frac{1}{2}} Q}_{(2F\times 3)} \underbrace{Q^{-1} (\Sigma_1)^{\frac{1}{2}} V_1^T}_{(3\times N)}$$ is also valid.

$= M$　　$= S$ for some Q

How to find the matrix Q ?

Orthonormality of M

The Motion Matrix M:

$$M = \begin{bmatrix} i_1^T \\ \vdots \\ i_F^T \\ j_1^T \\ \vdots \\ j_F^T \end{bmatrix} = U_1(\Sigma_1)^{1/2}Q = \begin{bmatrix} \hat{i}_1^T \\ \vdots \\ \hat{i}_F^T \\ \hat{j}_1^T \\ \vdots \\ \hat{j}_F^T \end{bmatrix} Q = \begin{bmatrix} \hat{i}_1^T Q \\ \vdots \\ \hat{i}_F^T Q \\ \hat{j}_1^T Q \\ \vdots \\ \hat{j}_F^T Q \end{bmatrix}$$

Orthonormality Constraints:

$$\boxed{\begin{aligned} i_f \cdot i_f &= i_f^T i_f = 1 \\ j_f \cdot j_f &= j_f^T j_f = 1 \\ i_f \cdot j_f &= i_f^T j_f = 0 \end{aligned}} \Rightarrow \boxed{\begin{aligned} \hat{i}_f^T Q Q^T \hat{i}_f &= 1 \\ \hat{j}_f^T Q Q^T \hat{j}_f &= 1 \\ \hat{i}_f^T Q Q^T \hat{j}_f &= 0 \end{aligned}}$$

Orthonormality of M

- We have computed $(\hat{\imath}_f^T, \hat{\jmath}_f^T)$ for f = 1,…,F.

$$\left.\begin{aligned} \hat{\imath}_f^T Q Q^T \hat{\imath}_f &= 1 \\ \hat{\jmath}_f^T Q Q^T \hat{\jmath}_f &= 1 \\ \hat{\imath}_f^T Q Q^T \hat{\jmath}_f &= 0 \end{aligned}\right\} \qquad \text{Q is unknown.}$$

- Q is 3x3 matrix, 9 variables, 3F quadratic equations.
- Q can be solved with 3 or more images (F≥ 3) using Newton's method.

Final Solution: $\boxed{M = U_1\,(\Sigma_1)^{\frac{1}{2}}\,Q}$ $\boxed{S = Q^{-1}\,(\Sigma_1)^{\frac{1}{2}}\,V_1^T}$

Camera Motion

Scene struction

Summary: Orthographic SFM

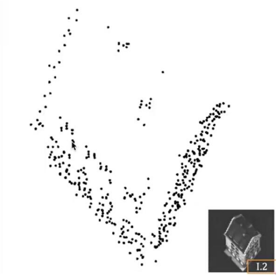
1. Detect and track feature points.
2. Create the centroid subtracted matrix w of corresponding feature points.
3. Compute SVD of W and enforce rank constraint.

$$W = U\,\Sigma\,V^T = \underset{(2F\times3)}{U_1}\;\underset{(3\times3)}{\Sigma_1}\;\underset{(3\times P)}{V_1^T}$$

4. Set $M = U_1\,(\Sigma_1)^{\frac{1}{2}}\,Q$ and $S = Q^{-1}\,(\Sigma_1)^{\frac{1}{2}}\,V_1^T$.
5. Find Q by enforcing the orthonormality constraint.

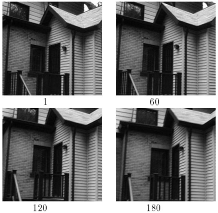
Result

Input image sequence



Estimated 3D points

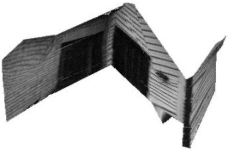
Result



Input image sequence

Tracked features

3D reconstruction



3D reconstruction

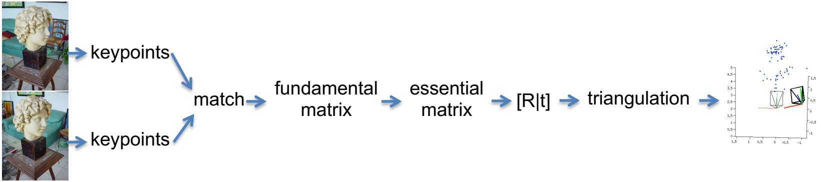
Structure from Motion Result

中国科学技术大学
University of Science and Technology of China

Bundle Adjustment

张举勇
中国科学技术大学

Two-view Reconstruction

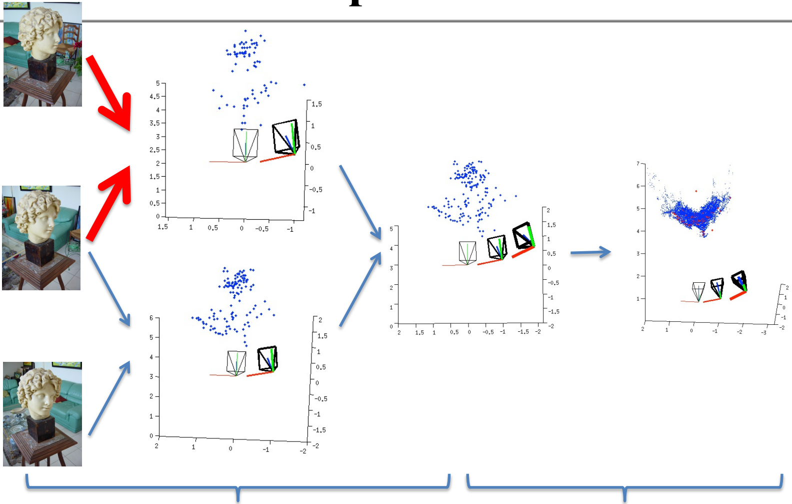


keypoints

keypoints

match → fundamental matrix → essential matrix → [R|t] → triangulation →

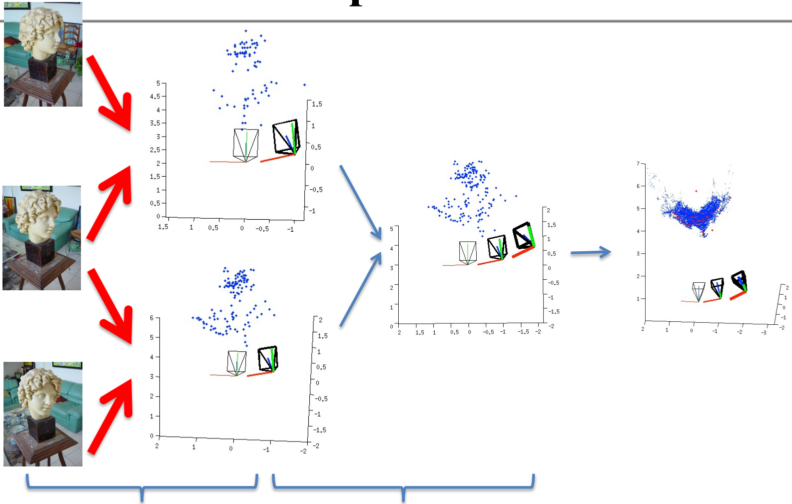
Pipeline



Structure from Motion (SFM)

Multi-view Stereo (MVS)

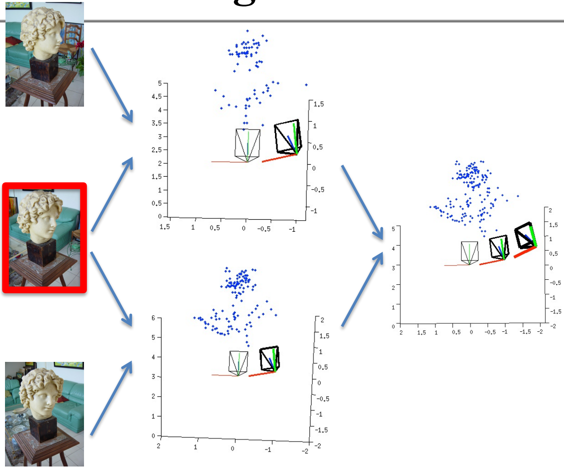
Pipeline



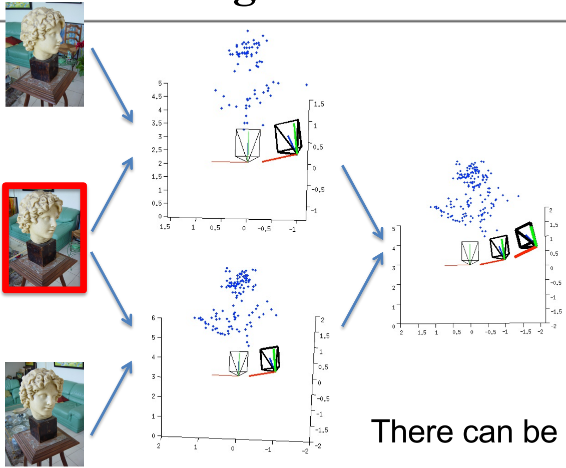
Taught

Next

Merge Two Point Cloud



Merge Two Point Cloud



There can be only one $\left[\mathbf{R}_2 | \mathbf{t}_2\right]$

Merge Two Point Cloud

- From the 1st and 2nd images, we have

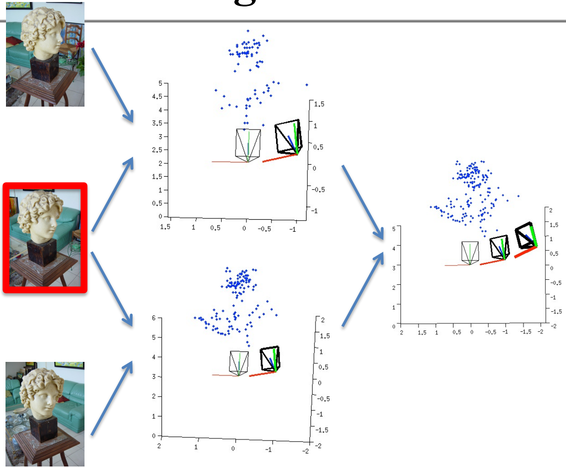
  $$[\mathbf{R}_1|\mathbf{t}_1] \text{ and } [\mathbf{R}_2|\mathbf{t}_2]$$

- From the 2nd and 3rd images, we have

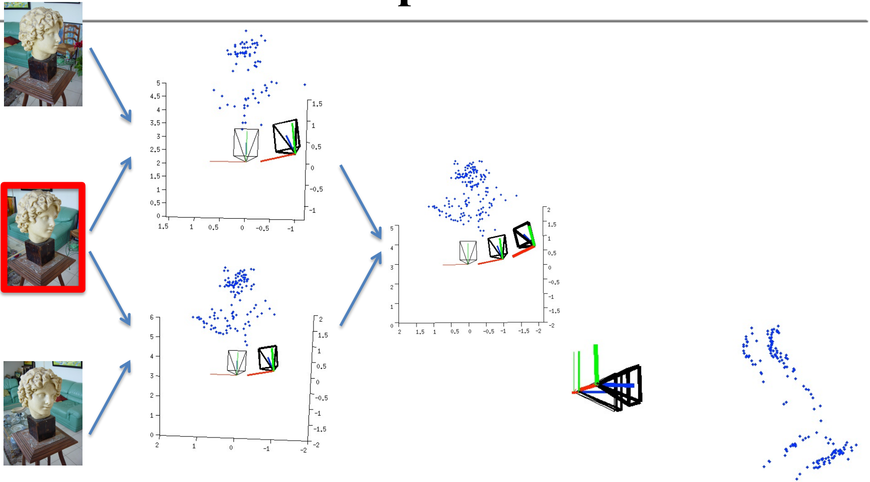
  $$[\mathbf{R}_2|\mathbf{t}_2] \text{ and } [\mathbf{R}_3|\mathbf{t}_3]$$

- How to transform the coordinate system of the second point cloud to align with the first point cloud so that there is only one $[\mathbf{R}_2|\mathbf{t}_2]$?

Merge Two Point Cloud

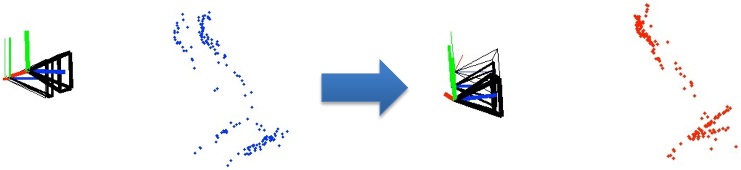


Oops



See From a Different Angle

Bundle Adjustment



Rethinking the SFM problem

- Input: Observed 2D image position

$$\begin{array}{ccc} \tilde{\mathbf{x}}_1^1 & \tilde{\mathbf{x}}_1^2 & \\ \tilde{\mathbf{x}}_2^1 & \tilde{\mathbf{x}}_2^2 & \tilde{\mathbf{x}}_2^3 \\ \tilde{\mathbf{x}}_3^1 & & \tilde{\mathbf{x}}_3^3 \end{array}$$

- Output:

  Unknown Camera Parameters (with some guess)

$$\left[\mathbf{R}_1|\mathbf{t}_1\right],\left[\mathbf{R}_2|\mathbf{t}_2\right],\left[\mathbf{R}_3|\mathbf{t}_3\right]$$

  Unknown Point 3D coordinate (with some guess)

$$\mathbf{X}^1,\mathbf{X}^2,\mathbf{X}^3,\cdots$$

Bundle Adjustment

A valid solution $[\mathbf{R}_1|\mathbf{t}_1],[\mathbf{R}_2|\mathbf{t}_2],[\mathbf{R}_3|\mathbf{t}_3]$ and $\mathbf{X}^1,\mathbf{X}^2,\mathbf{X}^3,\cdots$ must let

$$\text{Re-projection}\left\{\begin{array}{lll} \mathbf{x}_1^1=\mathbf{K}[\mathbf{R}_1|\mathbf{t}_1]\mathbf{X}^1 & \mathbf{x}_1^2=\mathbf{K}[\mathbf{R}_1|\mathbf{t}_1]\mathbf{X}^2 & \\ \mathbf{x}_2^1=\mathbf{K}[\mathbf{R}_2|\mathbf{t}_2]\mathbf{X}^1 & \mathbf{x}_2^2=\mathbf{K}[\mathbf{R}_2|\mathbf{t}_2]\mathbf{X}^2 & \mathbf{x}_2^3=\mathbf{K}[\mathbf{R}_2|\mathbf{t}_2]\mathbf{X}^3 \\ \mathbf{x}_3^1=\mathbf{K}[\mathbf{R}_3|\mathbf{t}_3]\mathbf{X}^1 & & \mathbf{x}_3^3=\mathbf{K}[\mathbf{R}_3|\mathbf{t}_3]\mathbf{X}^3 \end{array}\right.$$

$$=$$

$$\text{Observation}\left\{\begin{array}{lll} \tilde{\mathbf{x}}_1^1 & \tilde{\mathbf{x}}_1^2 & \\ \tilde{\mathbf{x}}_2^1 & \tilde{\mathbf{x}}_2^2 & \tilde{\mathbf{x}}_2^3 \\ \tilde{\mathbf{x}}_3^1 & & \tilde{\mathbf{x}}_3^3 \end{array}\right.$$

Bundle Adjustment

A valid solution $\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right], \left[\mathbf{R}_2 \middle| \mathbf{t}_2\right], \left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]$ and $\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \cdots$ must let the Re-projection close to the Observation, i.e. to minimize the reprojection error

$$\min \sum_i \sum_j \left(\tilde{\mathbf{x}}_i^j - \mathbf{K}\left[\mathbf{R}_i \middle| \mathbf{t}_i\right]\mathbf{X}^j \right)^2$$

Solving This Optimization Problem

- Theory:

  The Levenberg–Marquardt algorithm

  http://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

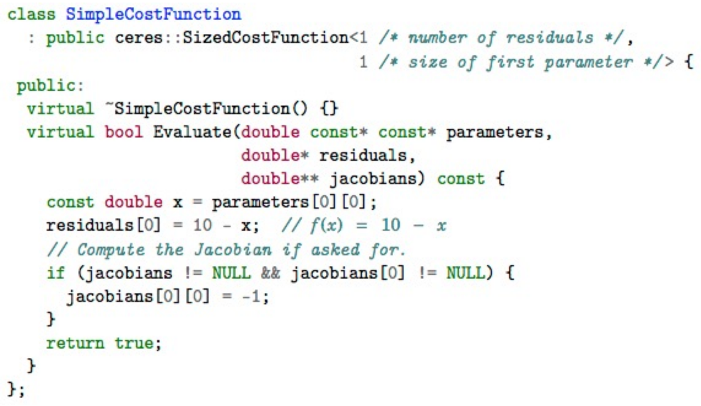
- Practice:

  The Ceres-Solver from Google

  http://code.google.com/p/ceres-solver/

Ceres-solver: A Nonlinear Least Squares Minimizer

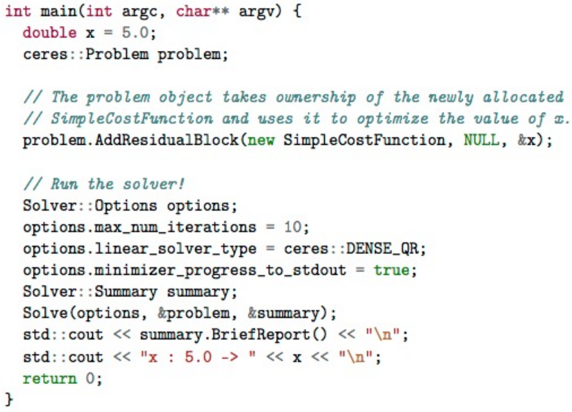
Toy problem to solve $\min\left(10 - x\right)^2$



```
class SimpleCostFunction
  : public ceres::SizedCostFunction<1 /* number of residuals */,
                                    1 /* size of first parameter */> {
 public:
  virtual ~SimpleCostFunction() {}
  virtual bool Evaluate(double const* const* parameters,
                        double* residuals,
                        double** jacobians) const {
    const double x = parameters[0][0];
    residuals[0] = 10 - x;  // f(x) = 10 - x
    // Compute the Jacobian if asked for.
    if (jacobians != NULL && jacobians[0] != NULL) {
      jacobians[0][0] = -1;
    }
    return true;
  }
};
```

Ceres-solver: A Nonlinear Least Squares Minimizer

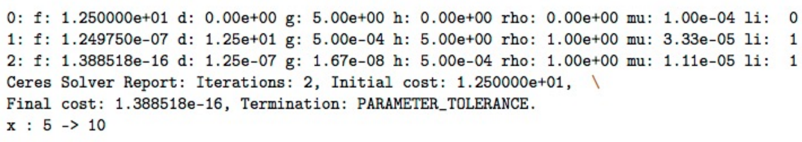
Toy problem to solve $\min(10-x)^2$



```
int main(int argc, char** argv) {
  double x = 5.0;
  ceres::Problem problem;

  // The problem object takes ownership of the newly allocated
  // SimpleCostFunction and uses it to optimize the value of x.
  problem.AddResidualBlock(new SimpleCostFunction, NULL, &x);

  // Run the solver!
  Solver::Options options;
  options.max_num_iterations = 10;
  options.linear_solver_type = ceres::DENSE_QR;
  options.minimizer_progress_to_stdout = true;
  Solver::Summary summary;
  Solve(options, &problem, &summary);
  std::cout << summary.BriefReport() << "\n";
  std::cout << "x : 5.0 -> " << x << "\n";
  return 0;
}
```

Ceres-solver: A Nonlinear Least Squares Minimizer

Toy problem to solve $\min(10-x)^2$



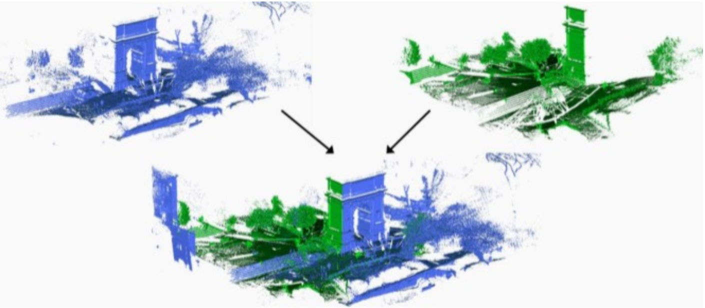
```
0: f: 1.250000e+01 d: 0.00e+00 g: 5.00e+00 h: 0.00e+00 rho: 0.00e+00 mu: 1.00e-04 li:  0
1: f: 1.249750e-07 d: 1.25e+01 g: 5.00e-04 h: 5.00e+00 rho: 1.00e+00 mu: 3.33e-05 li:  1
2: f: 1.388518e-16 d: 1.25e-07 g: 1.67e-08 h: 5.00e-04 rho: 1.00e+00 mu: 1.11e-05 li:  1
Ceres Solver Report: Iterations: 2, Initial cost: 1.250000e+01,  \
Final cost: 1.388518e-16, Termination: PARAMETER_TOLERANCE.
x : 5 -> 10
```

中国科学技术大学
University of Science and Technology of China

Geometry Registration

张举勇
中国科学技术大学

什么是注册？



- 计算最佳空间变换，以使得多个几何曲面之间进行对齐。
  - 将传感器采集的多个局部测量数据拼接成一个完整的几何模型
  - 将新测量数据对齐到已知模型以估计其姿态

变换类型

- Same object in a different position: size and shape preserving
  - Rigid-body transformation (rotation and translation)
  - Six degrees of freedom
    - ✦ translation $\mathbf{t} = (t_x, t_y, t_z)^T$
    - ✦ rotation (α, β, γ)

$$\mathbf{T}_{\text{rigid}}(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{t}$$

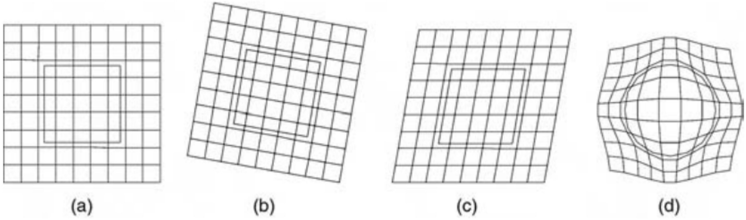
$$\mathbf{T}_{\text{rigid}}(\mathbf{x}) = \begin{bmatrix} \cos\beta\cos\gamma & \cos\alpha\sin\gamma + \sin\alpha\sin\beta\cos\gamma & \sin\alpha\sin\gamma - \cos\alpha\sin\beta\cos\gamma & t_x \\ -\cos\beta\sin\gamma & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & \sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma & t_y \\ \sin\beta & -\sin\alpha\cos\beta & \cos\alpha\cos\beta & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

变换类型

- Affine or Linear Transformation
  - Rigid-body transformation (rotation and translation)
  - Scaling and Shearing
  - Twelve degrees of freedom

$$\mathbf{T}(\mathbf{x}) = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

变换类型



Example of different types of transformations of a square

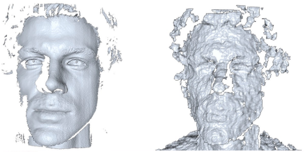
(a) identity transformation

(c) affine transformation

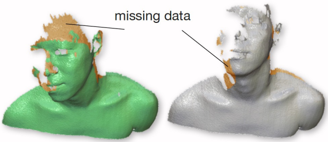
(b) rigid transformation

(d) nonrigid transformation

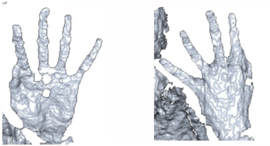
配准问题中的一些挑战



Noise



Partial matching



Ambiguity

Illumination changes

配准问题建模

- 将配准问题表达为能量最小化问题：

$$\underset{T}{\mathrm{argmax}} E_{reg}(T,P,Q)$$

$$E_{reg}(T,P,Q) = E_{match}(T,P,Q) + E_{prior}(T)$$

配准误差

如何衡量配准结果的质量？

变换误差

变换的类型与表示方式？

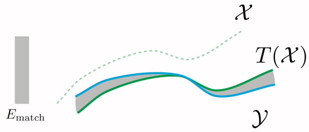
配准问题建模

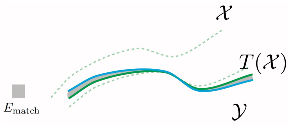
- 配准误差

$$E_{reg}(T,P,Q) = E_{match}(T,P,Q) + E_{prior}(T)$$

$$E_{match}(T,P,Q) = \int_X \phi(T(p),Q)dx$$

距离度量函数



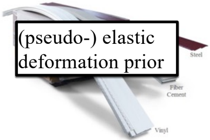


配准问题建模

- 变换误差

$$E_{reg}(T,P,Q) = E_{match}(T,P,Q) + E_{prior}(T)$$

Rigid



Elastic

Articulated

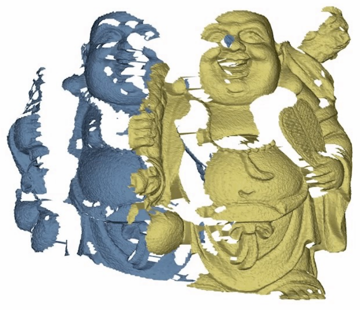
Composite

几何数据融合与跟踪-刚性注册

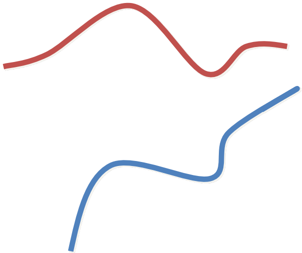
- 刚体几何建模：将不同视角点云进行刚性拼接，以获得完整几何模型

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in K} \| \mathbf{p} - \mathbf{T}\mathbf{q} \|^2$$

T 是一个包含旋转与平移的刚性变换



Aligning 3D Data

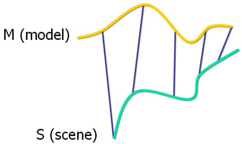


Corresponding Point Set Alignment

- Let M be a model point set.
- Let S be a scene point set.

We assume:

1. $N_M = N_S$.
2. Each point S_i correspond to



Corresponding Point Set Alignment

The MSE objective function :

$$f(R,T)=\frac{1}{N_S}\sum_{i=1}^{N_S}\left\|m_i-Rot(s_i)-Trans\right\|^2$$

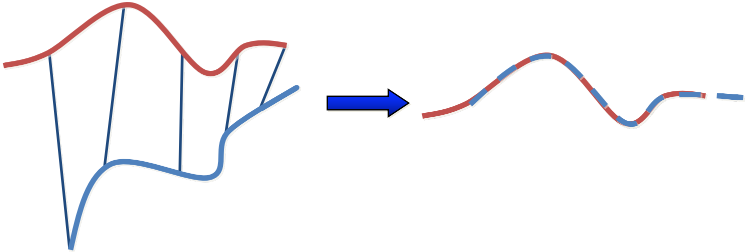
$$f(q)=\frac{1}{N_S}\sum_{i=1}^{N_S}\left\|m_i-R(q_R)s_i-q_T\right\|^2$$

The alignment is:

$$(rot,trans,d_{mse})=\Phi(M,S)$$

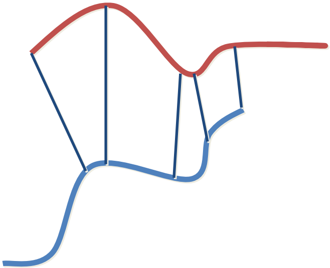
Aligning 3D Data

- If correct correspondences are known, can find correct relative rotation/translation



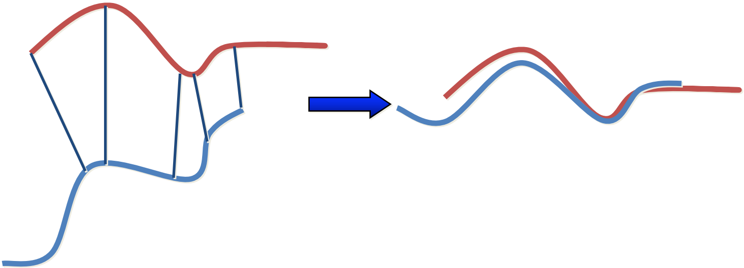
Aligning 3D Data

- How to find correspondences: User input? Feature detection? Signatures?
- Alternative: assume closest points correspond



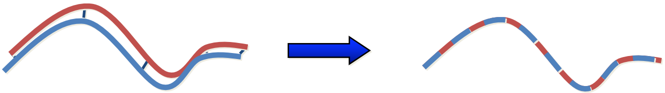
Aligning 3D Data

- How to find correspondences: User input? Feature detection? Signatures?
- Alternative: assume closest points correspond



Aligning 3D Data

- Converges if starting position “close enough“



Closest Point

- Given 2 points r_1 and r_2 , the Euclidean distance is:

$$d(r_1, r_2) = \|r_1 - r_2\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

- Given a point r_1 and set of points A , the Euclidean distance is:

$$d(r_1, A) = \min_{i \in 1..n} d(r_1, a_i)$$

Finding Matches

- The scene shape S is aligned to be in the best alignment with the model shape M.
- The distance of each point s of the scene from the model is :

$$d(s, M) = \min_{m \in M} d\|m - s\|$$

Finding Matches

$$d(s,M)=\min_{m\in M} d\left\|m-s\right\|=d(s,y)$$

$$y\in M$$

$$Y=C(S,M)$$

$$Y\subseteq M$$

C – the closest point operator

Y – the set of closest points to S

Finding Matches

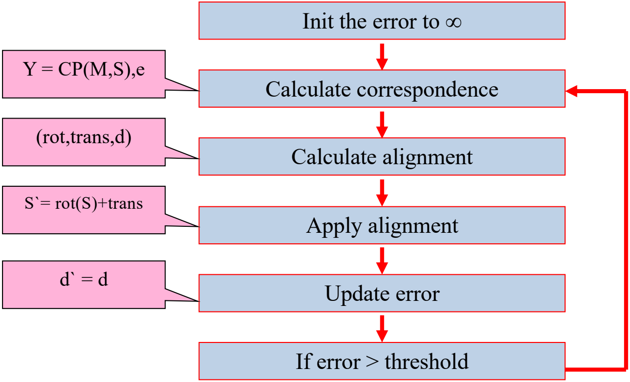
- Finding each match is performed in O(N_M) worst case.
- Given Y we can calculate alignment

$$(rot, trans, d) = \Phi(S, Y)$$

- S is updated to be :

$$S_{new} = rot(S) + trans$$

The Algorithm



Init the error to ∞

Calculate correspondence

Calculate alignment

Apply alignment

Update error

If error > threshold

Y = CP(M,S),e

(rot,trans,d)

S`= rot(S)+trans

d` = d

Convergence Theorem

- The ICP algorithm always converges monotonically to a local minimum with respect to the MSE distance objective function.

Convergence Theorem

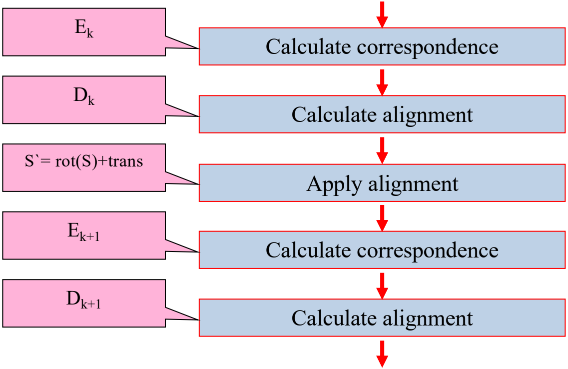
- Correspondence error :

$$e_k = \frac{1}{N_S}\sum_{i=1}^{N_S}\left\| y_{ik} - s_{ik} \right\|^2$$

- Alignment error:

$$d_k = \frac{1}{N_S}\sum_{i=1}^{N_S}\left\| y_{ik} - Rot_k(s_{io}) - Trans_k \right\|^2$$

Convergence Theorem



E_k → Calculate correspondence

D_k → Calculate alignment

S`= rot(S)+trans → Apply alignment

E_{k+1} → Calculate correspondence

D_{k+1} → Calculate alignment

Convergence Theorem

- Proof :

$$S_k = Rot_k(S_0) + Trans_k$$

$$Y_k = C(M, s_k)$$

$$e_k = \frac{1}{N_S} \sum_{i=1}^{N_S} \left\| y_{ik} - s_{ik} \right\|^2$$

$$d_k = \frac{1}{N_S} \sum_{i=1}^{N_S} \left\| y_{ik} - Rot_k(s_{io}) - Trans_k \right\|^2$$

Convergence Theorem

- Proof : $d_k \leq e_k$

If not - the identity transform would yield a smaller MSE than the least square alignment.

Apply the alignment q_k on S_0 → S_{k+1} .

Assuming the correspondences are maintained : the MSE is still d_k.

$$d_k = \frac{1}{N_M} \sum_{i=1}^{N_M} \left\| y_{ik} - S_{ik} \right\|^2$$

Convergence Theorem

- Proof :

After the last alignment, the closest point operator is applied : $Y_{k+1} = C(M, S_{k+1})$

It is clear that:

$$\left\| y_{i,k+1} - S_{i,k+1} \right\| \le \left\| y_{ik} - S_{i,k+1} \right\|$$

$$e_{k+1} \le d_k$$

Thus : $$0 \le d_{k+1} \le e_{k+1} \le d_k \le e_k$$

Time analysis

Each iteration includes 3 main steps

A. Finding the closest points :

$O(N_M)$ per each point

$O(N_M*N_S)$ total.

B. Calculating the alignment: $O(N_S)$

C. Updating the scene: $O(N_S)$

Optimizing the Algorithm

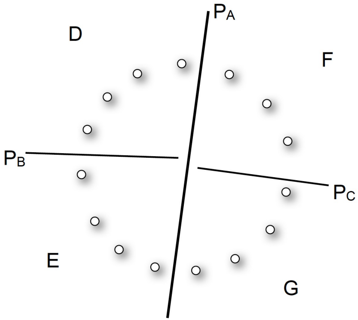
The best match/nearest neighbor problem :

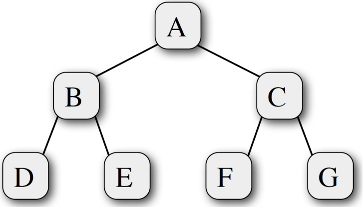
Given a record, and a dissimilarity measure **D**, find the closest record from a set to the query record.

Closest Point Search

- Find closest point of a query point
  - Brute force: O(n) complexity
- Use hierarchical BSP tree
  - Binary space partitioning tree (also kD-tree)
  - Recursively partition 3D space by planes
  - Tree should be balanced, put plane at median
  - log(n) tree levels, complexity O(log n)

BSP Closest Point

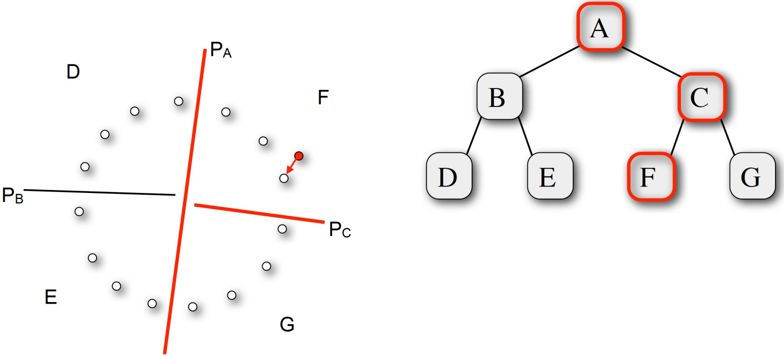




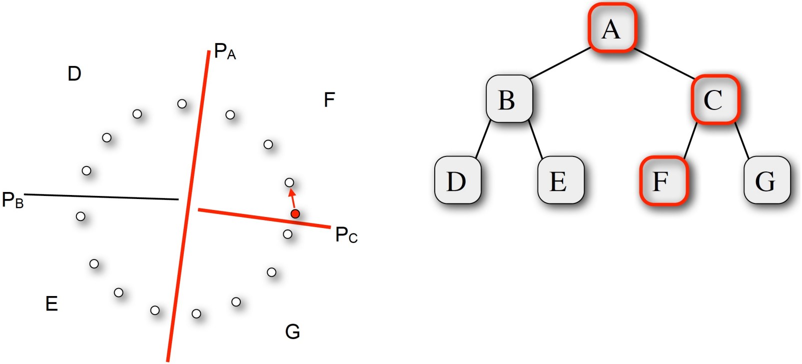
BSP Closest Point

```
BSPNode::dist(Point x, Scalar& dmin)
{
  if (leaf_node())
    for each sample point p[i]
      dmin = min(dmin, dist(x, p[i]));

  else
  {
    d = dist_to_plane(x);
    if (d < 0)
    {
      left_child->dist(x, dmin);
      if (|d| < dmin) right_child->dist(x, dmin);
    }
    else
    {
      right_child->dist(x, dmin);
      if (|d| < dmin) left_child->dist(x, dmin);
    }
  }
}
```

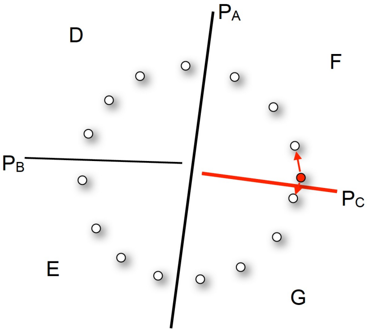
BSP Closest Point

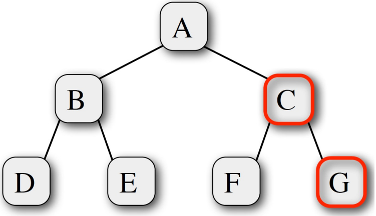


BSP Closest Point

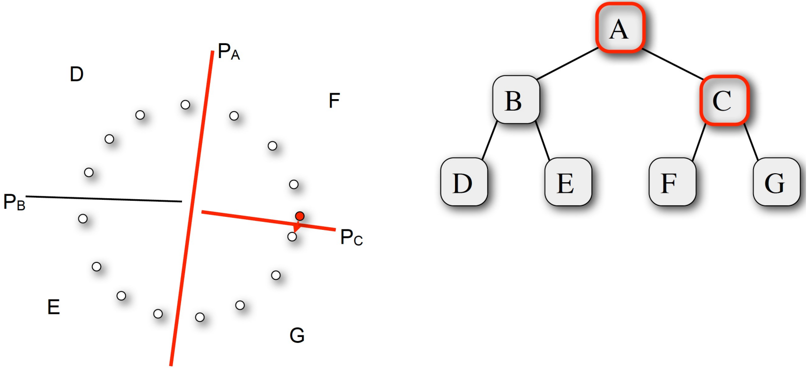


BSP Closest Point





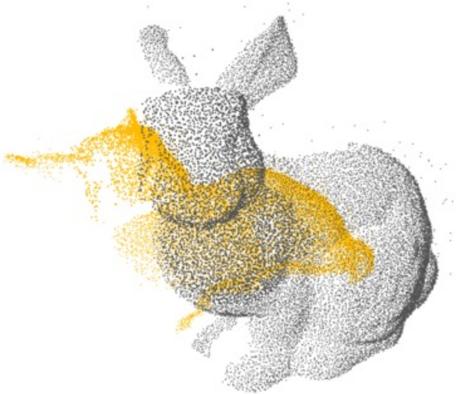
BSP Closest Point



ICP Variants

- Variants on the following stages of ICP have been proposed:
  - Selecting sample points (from one or both meshes)
  - Matching to points in the other mesh
  - Weighting the correspondences
  - Rejecting certain (outlier) point pairs
  - Assigning an error metric to the current transform
  - Minimizing the error metric w.r.t. transformation

Real Time ICP



中国科学技术大学
University of Science and Technology of China

Object Tracking

张举勇
中国科学技术大学

Overview

Track the location of target objects in each frame of a video sequence

Topics:

(1) Change Detection

(2) Gaussian Mixture Model

(3) Object Tracking using Templates

(4) Tracking by Feature Detection

Change Detection

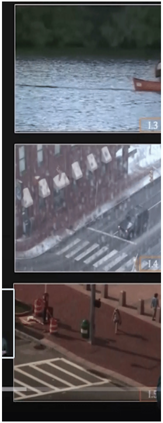
Given: Static cameras observing scene (room, street, etc.)
Find: Meaningful changes (moving objects, people, etc.)

Robust and real-time classification of each pixel as "foreground" (motion/change) or "background" (static).

Change Detection: Challenges

Ignore uninteresting changes:

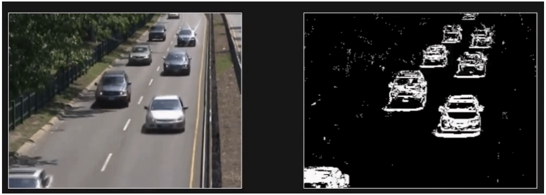
- Background fluctuations
- Image noise
- Rain, snow, turbulence
- Illumination changes & shadows
- Camera shake



Simple Frame Difference

Label significant difference between current and previous frames as background.

$F_t = |I_t - I_{t-1}| > T$ T: threshold



Input video sequence Frame difference

Not Robust!

Background Modeling: Average

Build simple model of background before classification.

Background B
median{I_1, I_2,…, I_K}
(First K frames)

Input Frame I_t

Foreground F_t
$F_t = |I_t - B| > T$

Cannot handle change in lighting, background, etc.

Background Modeling: Median

Build simple model of background before classification.

Background $\mathrm{B_t}$
median{I_{t-1}, I_{t-2},…,
I_{t-K}}
(Last K frames)

Input Frame I_t

Foreground F_t
$F_t = |I_t - \mathrm{B}| > T$

Cannot handle change in lighting, background, etc.

Background Modeling: Moving Median

Build simple adaptive model of background over time.

Background B_t
median$\{I_{t-1}, I_{t-2}, \ldots, I_{t-K}\}$
(Last K frames)

Input Frame I_t

Foreground F_t
$F_t = |I_t - B| > T$

Requires keeping the last K frames in memory.
Finding median for each pixel is expensive.

Background Modeling: Moving Median

Build simple adaptive model of background over time.

Background $\mathrm{B_t}$

median{I_{t-1}, I_{t-2},…, I_{t-K}}

(Last K frames)

Input Frame I_t

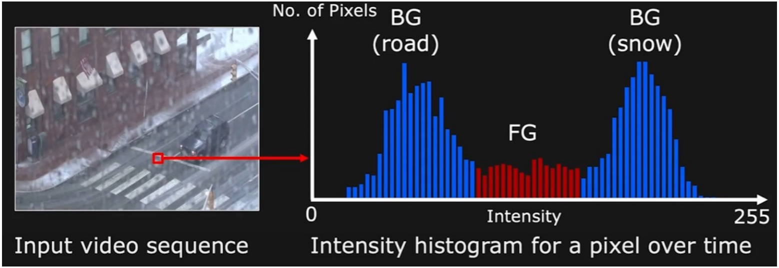
Foreground F_t

$F_t = |I_t - \mathrm{B}| > T$

Cannot handle significant pixel fluctuations (weather shadow, shake, etc.)

Mixture Model

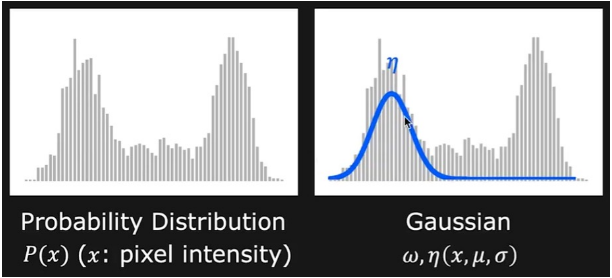
Intensity distribution at each pixel over time:



Intensity variations due to static scene (road), noise (snow), and occasional moving objects(vehicles)

Intuition: Pixels are background most of time.

Gaussian Model



Probability Distribution

$P(x)$ (x: pixel intensity)

Gaussian

$\omega, \eta(x, \mu, \sigma)$

1-Dimensional Gaussian:

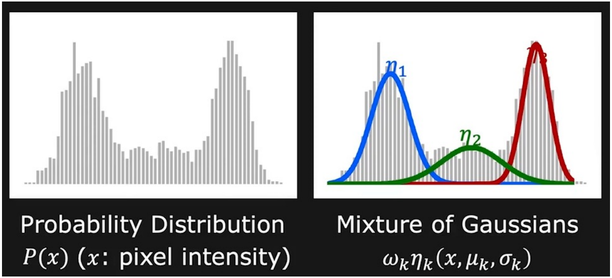
$$\boxed{\omega\, \eta(x, \mu, \sigma) = \omega \frac{1}{\sqrt{2\pi}\sigma}\, e^{-\frac{1}{2\sigma^2}(x-\mu)^2}}$$

μ: Mean

σ: Std. Deviation

ω: Scale

Mixture of Gaussians

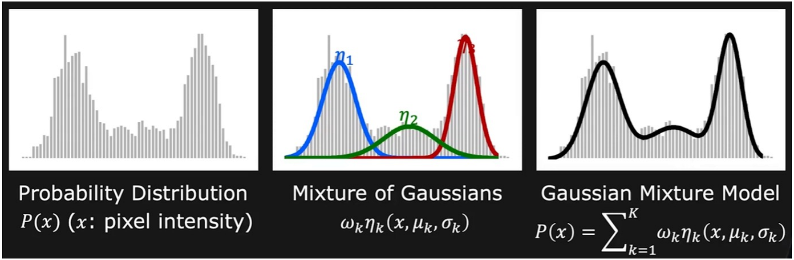


Probability Distribution $P(x)$ (x: pixel intensity)

Mixture of Gaussians $\omega_k \eta_k(x, \mu_k, \sigma_k)$

Assume P(x) is made of K different Gaussians.

Gaussian Mixture Model (GMM)



Probability Distribution
$P(x)$ (x: pixel intensity)

Mixture of Gaussians
$\omega_k \eta_k(x, \mu_k, \sigma_k)$

Gaussian Mixture Model
$P(x) = \sum_{k=1}^{K} \omega_k \eta_k(x, \mu_k, \sigma_k)$

GMM Distribution: Weighted sum of K Gaussians

$$\boxed{P(x) \cong \sum_{k=1}^{K} \omega_k \eta_k(x, \mu_k, \sigma_k)} \quad \text{such that} \quad \sum_{k=1}^{K} \omega_k = 1$$

High Dimensional Model

Let $P(\mathbf{X})$ be a probability distribution of a D-dimensional random variable $\mathbf{X} \in \mathcal{R}^D$.For example: $\mathbf{X} = [r, g, b]^T$

GMM of $P(\mathbf{X})$: Sum of K D-dimensional Gaussians

$$P(\mathbf{X}) \cong \sum_{k=1}^{K} \omega_k \eta_k(\mathbf{X}, \boldsymbol{\mu}_k, \Sigma_k) \text{ such that } \sum_{k=1}^{K} \omega_k = 1$$

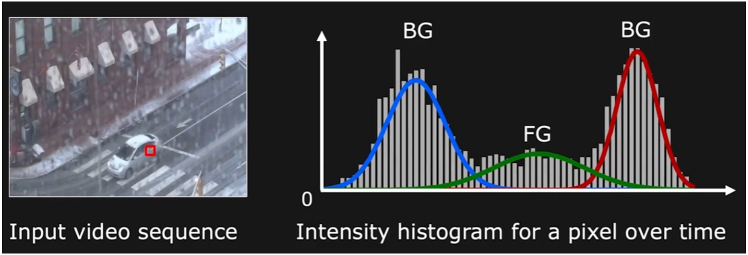
where: $\eta(\mathbf{X}, \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{X}-\boldsymbol{\mu})^T(\Sigma)^{-1}(\mathbf{X}-\boldsymbol{\mu})}$

Mean $\boldsymbol{\mu} = \begin{bmatrix} \mu_r \\ \mu_g \\ \mu_b \end{bmatrix}$ Covariance matrix $\Sigma = \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}$ (can be a full matrix)

GMM can be estimated from $P(\mathbf{X})$. (MATLAB: gmdistribution.fit)

Background Modeling with GMM

Given: A GMM for intensity/color variation at a pixel over time
Classify: Individual Gaussians as foreground/background



Input video sequence

Intensity histogram for a pixel over time

Intuition: Pixels are background most of time.That is,Gaussians with large supporting evidence ω and small σ.

Large $\frac{\omega}{\sigma}$: Background

Small $\frac{\omega}{\sigma}$: Foreground

Change Detection using GMM

For each pixel:

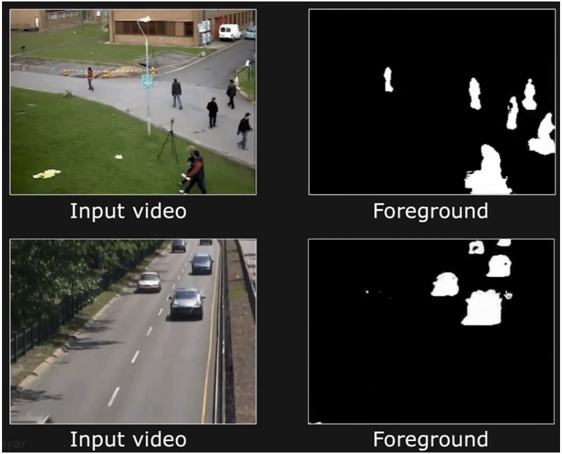
1. Compute pixel color histogram H using first N frames.
2. Normalize histogram: $\widehat{H} \leftarrow H/\| H \|$.
3. Model $\widehat{H}$ as mixture of K (3 to 5) Gaussians.
4. For each subsequent frame:
   a. The pixel value $\mathbf{X}$ belongs to Gaussian k in GMM for which $\|\mathbf{X} - \boldsymbol{\mu}_k\|$ is minimum and $\|\mathbf{X} - \boldsymbol{\mu}_k\| < 2.5\sigma_k$

   b. If ${}^{\omega_k}/\sigma_k$ is large then classify pixel as background. Else classify as foreground.

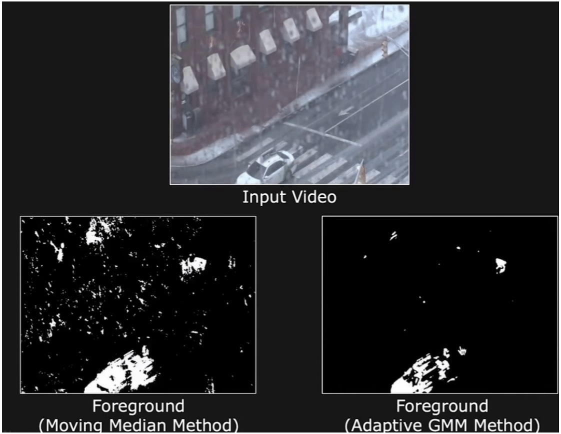
   c. Update histogram H using new pixel intensity.

   d. If $\widehat{H}$ and $H/\| H \|$ differ a lot $\big(\|\widehat{H} - {}^{H}/\|H\|\|$ is large), $\widehat{H} \leftarrow H/\| H \|$ and refit GMM.

Adaptive GMM based change detection



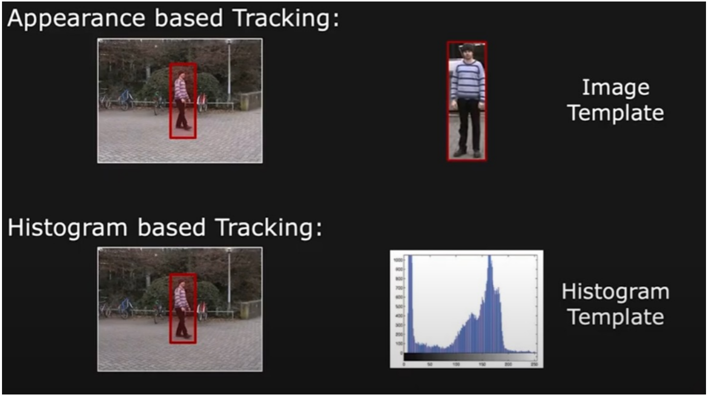
Adaptive GMM based change detection



Object Tracking

Given: Location of target in initial or previous frame.
Find: Location of target in current frame.

Target templates for Tracking



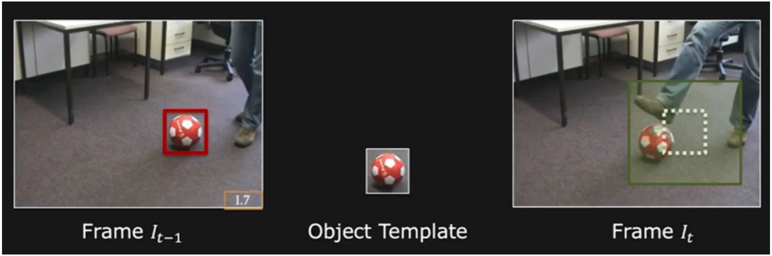
Appearance based Tracking:

Image Template

Histogram based Tracking:

Histogram Template

Tracking using Appearance Matching



Given template window S in frame I_{t-1}, search neighborhood to find match in image I_t.

Simple implementation.Not robust to change in scale,viewpoint, Occlusion,etc.

Similarity Metrics for Template Matching

Find pixel $(k, l) \in S$ with Minimum Sum of Absolute Differences:

$$SAD(k, l) = \sum_{(i,j)\in T} |I_1(i, j) - I_2(i, +kj + l)|$$

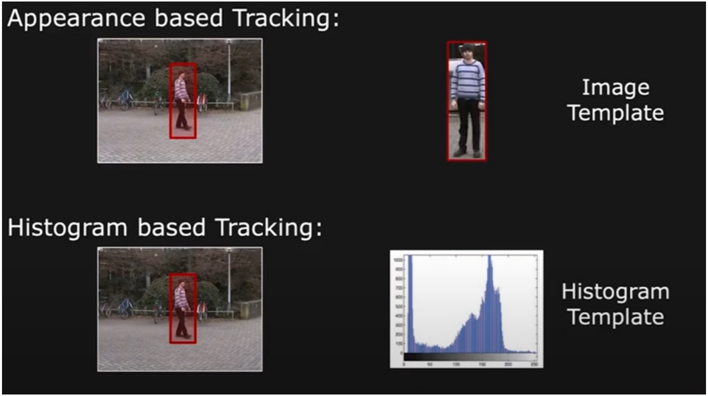
Find pixel $(k, l) \in S$ with Minimum Sum of Squared Differences:

$$\mathrm{SSD}(k, l) = \sum_{(i,j)\in T} |I_1(i, j) - I_2(i + k, j + l)|^2$$

Find pixel $(k, l) \in S$ with Minimum Normalized Cross-Correlation:

$$\mathrm{NCC}(k, l) = \frac{\sum_{(i,j)\in T} I_1(i, j) I_2(i + k, j + l)}{\sqrt{\sum_{(i,j)\in T} I_1(i, j)^2 \sum_{(i,j)\in T} I_2(i + k, j + l)^2}}$$

Target templates for Tracking



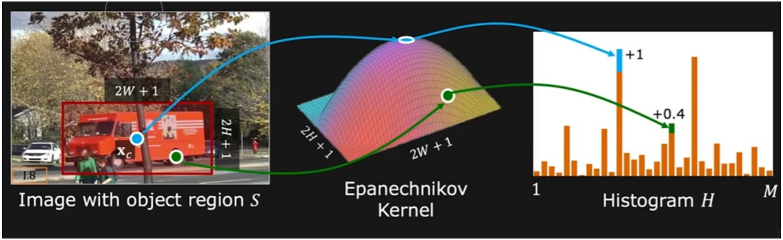
Appearance based Tracking:

Image Template

Histogram based Tracking:

Histogram Template

Computing Weighted Histogram



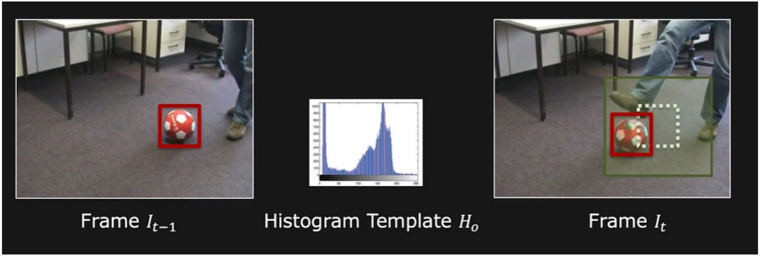
Weighted histogram gives more importance to pixels at center.

Epanechnikov Kernel:

$$k(\tilde{\mathbf{x}}) = \begin{cases} 1 - \| \tilde{\mathbf{x}} \|^2, & \| \tilde{\mathbf{X}} \| < 1 \\ 0, & \text{otherwise} \end{cases} \quad \tilde{\mathbf{x}} = \begin{bmatrix} (x - x_c)/W \\ (y - y_c)/H \end{bmatrix}$$

Comparing Histograms: Correlation, Intersection, etc.

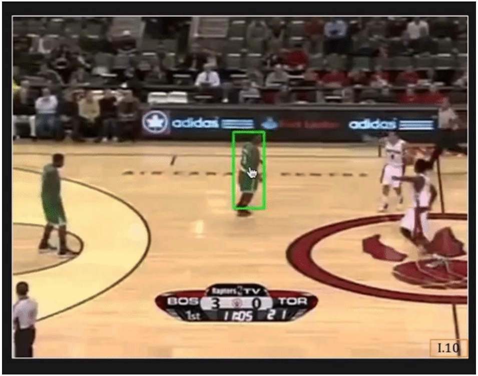
Tracking using Histogram Matching



Given a histogram template H_0 and location x_{t-1} in I_{t-1}, search neighborhood in I_t to find window in matching histogram.

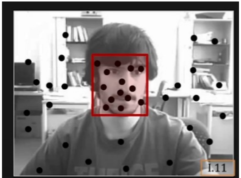
More resilient to changes in object pose and/or scale

Histogram Based Tracking: Results



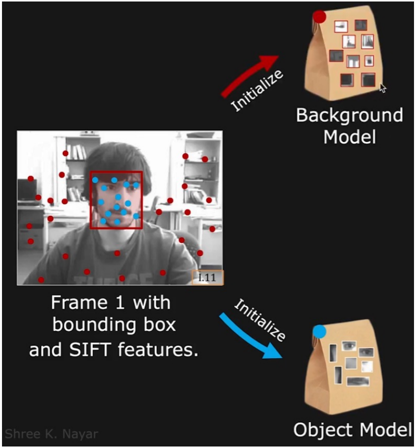
Robust when object appearance is unique in the environment and its size remains more or less the same.

Tracking by Feature Detection

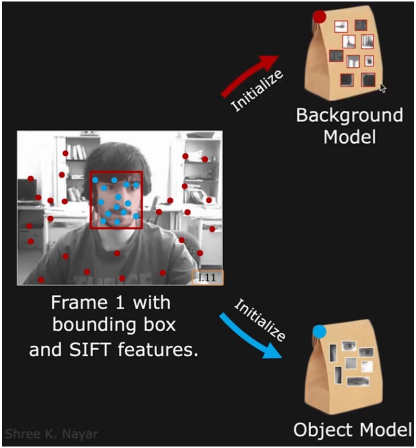


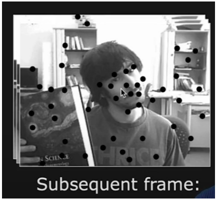
Frame 1 with
bounding box
and SIFT features.

Tracking by Feature Detection

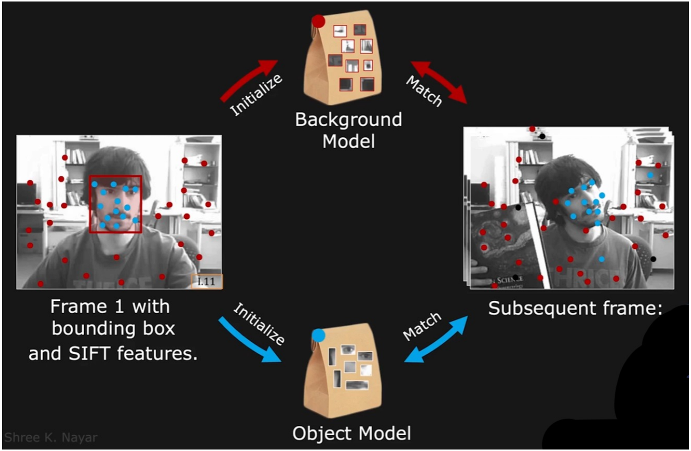


Tracking by Feature Detection



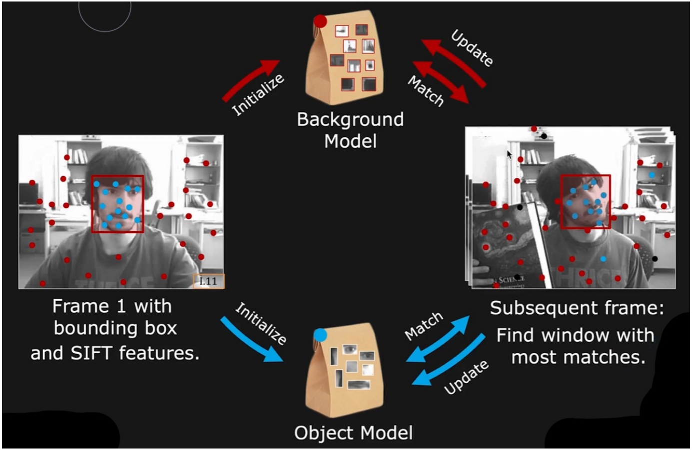


Tracking by Feature Detection



[redacted]

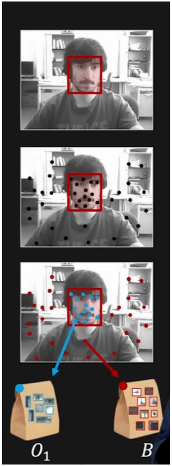
Tracking by Feature Detection



Tracking Intialization

At frame 1:

1. User selects a bounding box W_1 as object/target.
2. Compute SIFT (or similar) features for the frame.
3. Classify features within the box as object and assign them to set O_1.
4. Classify remaining features as background and assign them to set B.



Object Tracking

At frame t :

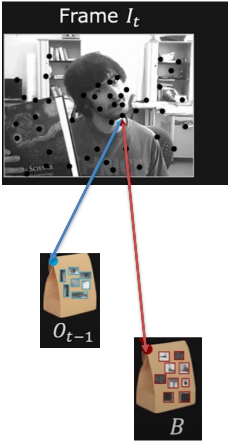
1. Compute SIFT features and SIFT descriptors $\{\mathbf{v}_1, \ldots, \mathbf{v}_K\}$ for frame I_t.
2. For each feature and corresponding descriptor $\mathbf{v}_i$:

   a. Compute distance d_o between $\mathbf{v}_i$ and the best match in object set O_{t-1}

   b. Compute distance d_B between $\mathbf{v}_i$ and the best match in background set B.

   c. $C(\mathbf{v}_i) =$

$$\begin{cases} +1 & \text{if } d_O/d_B < 0.5 (\mathbf{v}_i \text{ may belong to object }) \\ -1 & \text{otherwise } (\mathbf{v}_i \text{ does not belong to object }) \end{cases}$$

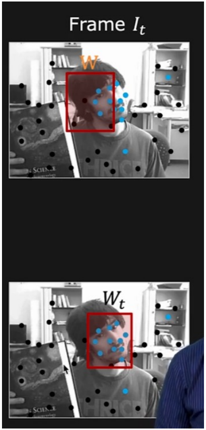


Object Tracking

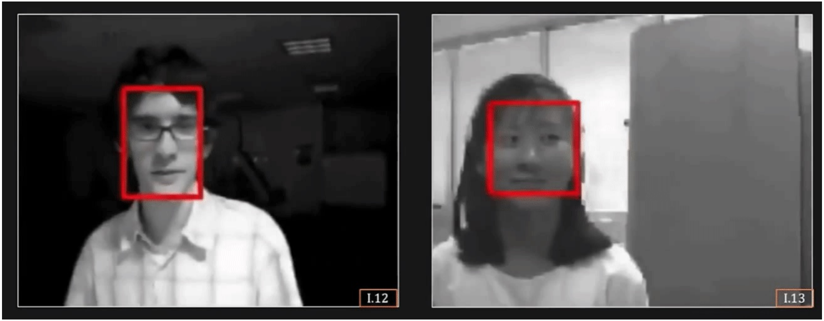
3. For each Search Window W :
   a. Compute $\varphi(W) = \sum C(\mathbf{v}_i)$ for all features $\mathbf{v}_i$ inside W.
   b. Compute a heuristic $\tau(W, W_{t-1})$ that penalizes large deviations from previous location, size and shape W_{t-1}.
   C. Compute Match Score
   $$\mu(W) = \varphi(W) - \tau(W, W_{t-1})$$

4. Select window W_t with the best match score as new object location.

5. Update object appearance model: $O_t = O_{t-1} \cup \{\mathbf{v}_i\} \forall \mathbf{v}_i$ inside W_t such that $C(\mathbf{v}_i) = +1$.



Tracking Results: Scale and Orientation

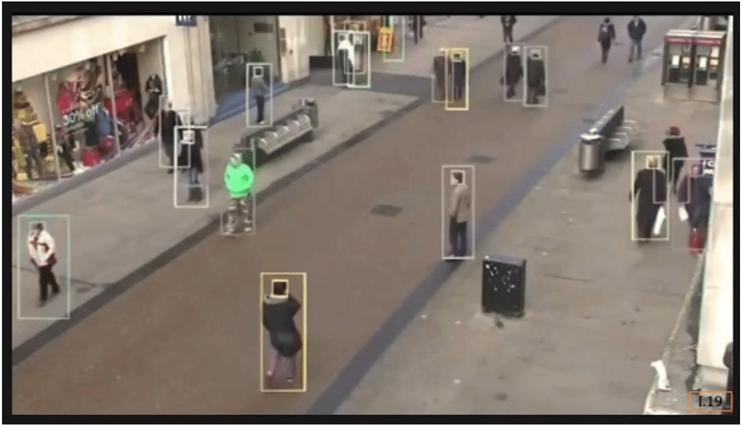


Resilient to changes in scale and orientation.

Tracking Results: Occlusion

Resilient to occlusion.

Tracking Applications



Tracking people in the wild.

Tracking Applications

Tracking people in the wild.

Tracking Applications

Traffic Monitoring.

Tracking Applications

Customer Behavior for In-Store Analytics.

中国科学技术大学
University of Science and Technology of China

3D Face Reconstruction

张举勇
中国科学技术大学

背景：数字世界

真实世界

时空约束限制了工作、生活、娱乐等方面的需求

数字世界

无限拓展想象力与创造力

三维数字内容建模与生成

- 对物理世界进行高效高保真数字化是支撑VR、AR、元宇宙等上层应用的核心基础

微软-Fusion4D系统

Meta-Horizon Workrooms

背景—人脸重建应用


Source Actor
Real-time Reenactment
Reenactment Result
Target Actor

背景—人脸重建应用

背景—人脸重建应用

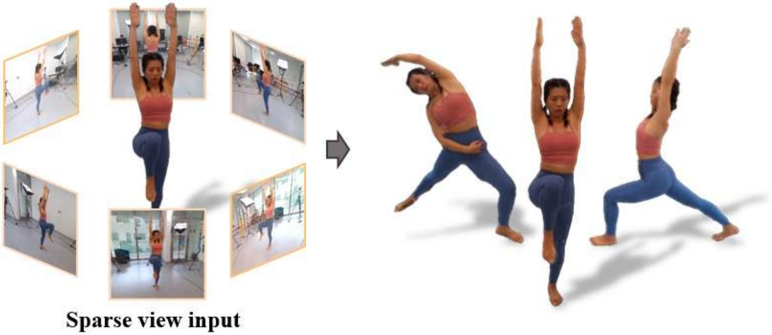
背景—数字交流

基于相机阵列的数字人建模

- 😀 高精度建模效果
- 😀 可恢复材质、光照等

- 😞 受控的采集环境、昂贵的价格
- 😞 复杂的制作流程

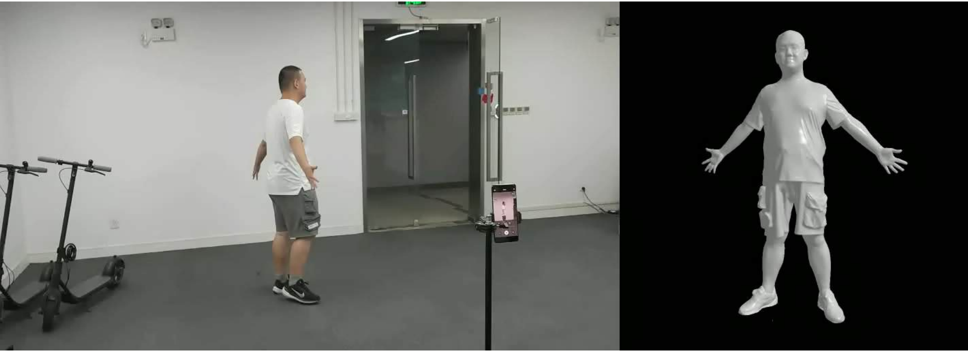
基于稀疏视角的数字人建模



😀 成本与便捷性得到极大提高

😞 对于普通大众仍遥不可及

愿景：基于单目相机的数字人建模与驱动



基于深度学习的实时单目三维人脸重建

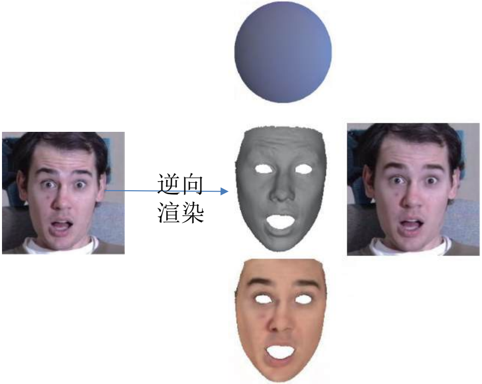
研究问题

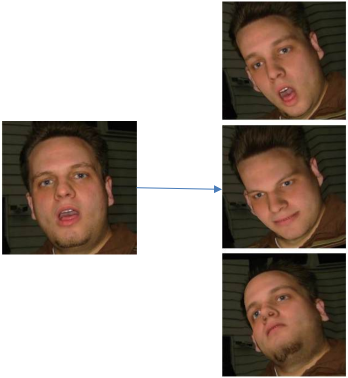
Input

Output

主要想法

基于逆向渲染的逼真人脸图片合成

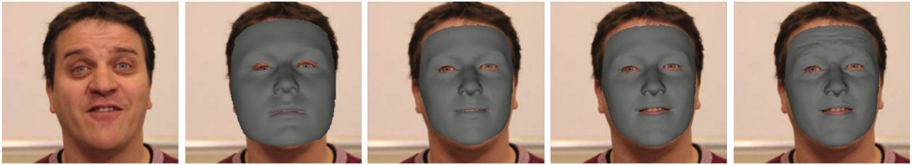




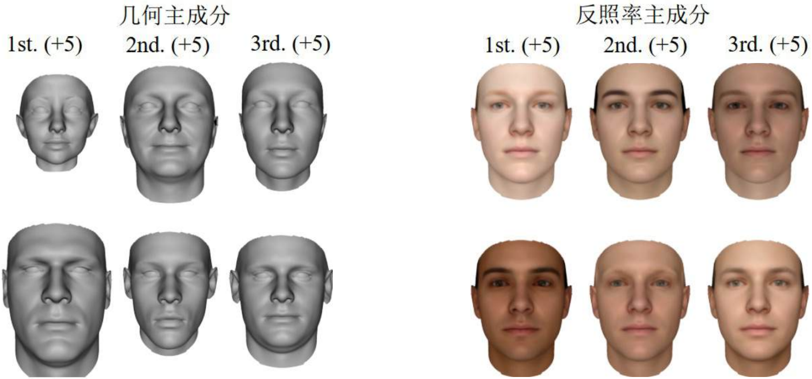
三维人脸表示

Mean Face Identity Expression Displacement

$$F = (\bar{F} + A_{\mathrm{id}}\alpha_{\mathrm{id}} + A_{\mathrm{exp}}\alpha_{\mathrm{exp}}) + F_{\mathrm{disp}}$$



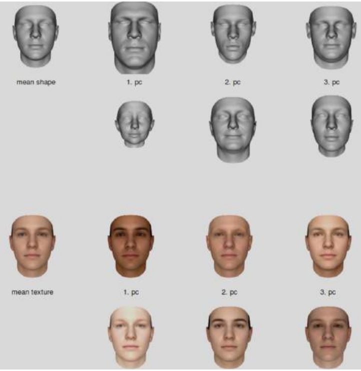
三维人脸参数化表示



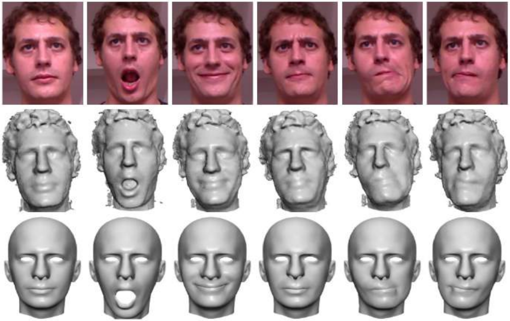
$$\mathbf{p} = \bar{\mathbf{p}} + \mathbf{A}_{\text{id}}\boldsymbol{\alpha}_{\text{id}} + \mathbf{A}_{\text{exp}}\boldsymbol{\alpha}_{\text{exp}}$$

$$\mathbf{b} = \bar{\mathbf{b}} + \mathbf{A}_{\text{alb}}\boldsymbol{\alpha}_{\text{alb}}$$

一些三维人脸参数化模型

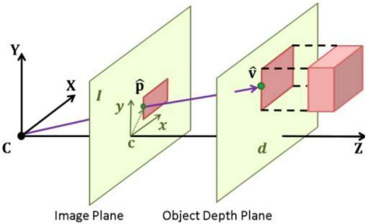


3DMM



FaceWarehouse

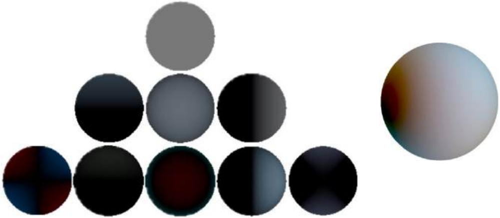
逆向渲染-渲染过程



$$\mathbf{q}_i = s\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} R\mathbf{p}_i + \mathbf{t}$$

相机投影

$$Y_l^m(\theta,\phi) = \begin{cases} \sqrt{2}K_l^m \cdot \cos(m\cdot\phi)\cdot P_l^m(\cos(\theta)) & \text{if } m>0 \\ K_l^m\cdot P_l^m(\cos(\theta)) & \text{if } m=0 \\ \sqrt{2}K_l^m\cdot\sin(-m\cdot\phi)\cdot P_l^{-m}(\cos(\theta)) & \text{if } m<0 \end{cases}$$



$$\mathcal{L}(\mathbf{n}_i, b_i \mid \gamma) = b_i \cdot \sum_{k=1}^{B^2} \gamma_k \phi_k(\mathbf{n}_i)$$

光照模型

逆向渲染-优化过程

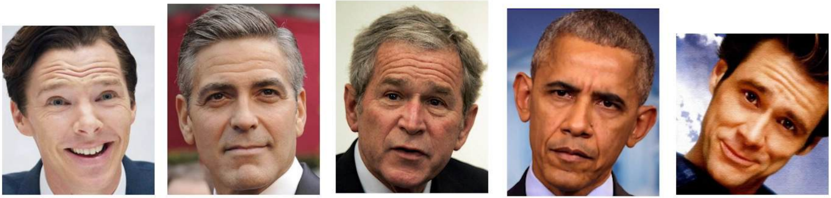
$$E(\chi) = E_{\text{con}} + w_l E_{\text{lan}} + w_r E_{\text{reg}}$$

$$E_{\text{con}}(\chi) = \frac{1}{|\mathcal{F}|} \| I_{\text{ren}} - I_{\text{in}} \|^2$$

$$E_{\text{lan}}(\chi) = \frac{1}{|\mathcal{L}|} \sum_{i \in \mathcal{L}} \| \mathbf{q}_i - (\Pi R \mathbf{p}_i + \mathbf{t}) \|^2$$

$$\chi = \{\boldsymbol{\alpha}_{\text{id}}, \boldsymbol{\alpha}_{\text{exp}}, \boldsymbol{\alpha}_{\text{alb}}, s, pitch, yaw, roll, \mathbf{t}, \mathbf{r}\}$$

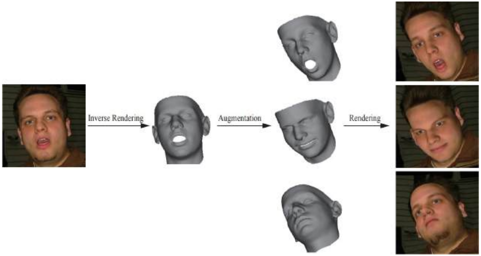
逆向渲染-几何细节



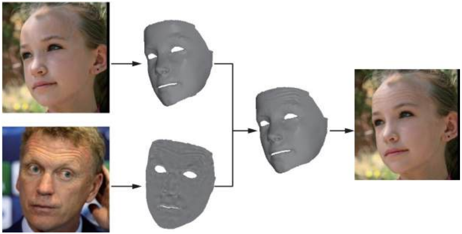
$$E(\mathbf{d}) = E_{\text{con}} + \mu_1 \|\mathbf{d}\|_2^2 + \mu_2 \| \triangle \mathbf{d}\|_1$$

构造训练数据

CoarseData



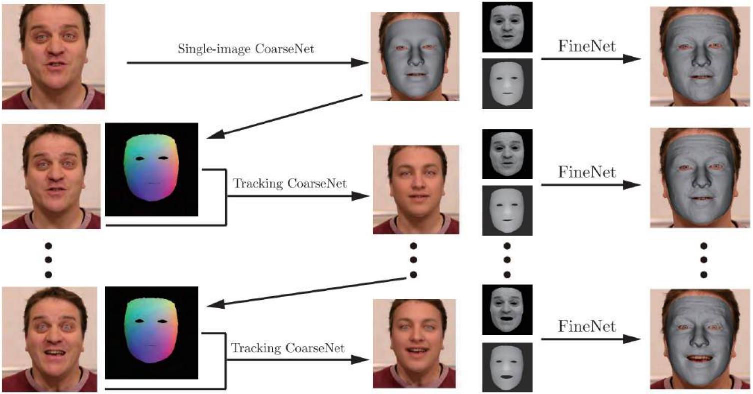
FineData



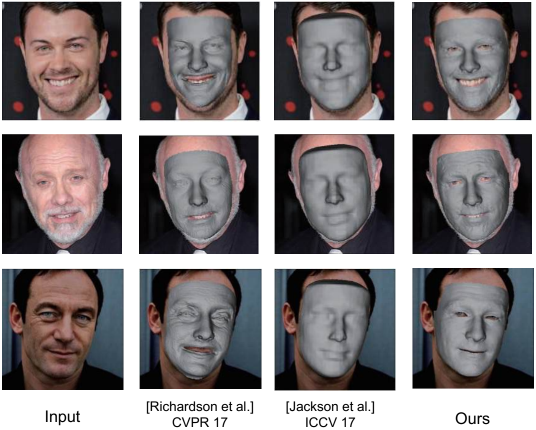
$$\min_{\tilde{\mathbf{d}}_t} \sum_{(i,j)\in\Omega} \|\nabla \tilde{\mathbf{d}}_t(i,j) - \mathbf{w}(i,j)\|^2,$$

$$\text{s.t.} \quad \tilde{\mathbf{d}}_t(i,j) = \mathbf{d}_t(i,j) \qquad (i,j) \in \partial\Omega$$

神经网络流程


Single-image CoarseNet
FineNet
Tracking CoarseNet
FineNet
Tracking CoarseNet
FineNet

实验结果



Method	RMSE [mm]	MAE [mm]
[61]	5.946	4.420
[60]	5.367	3.923
Ours	4.915	3.846

Quantitative comparison results with other methods on FRGC dataset

应用-川剧变脸

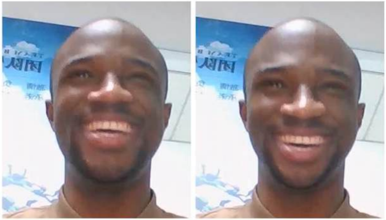
输入视频

输出视频

基于前置摄像头的实时人脸视角矫正

研究问题

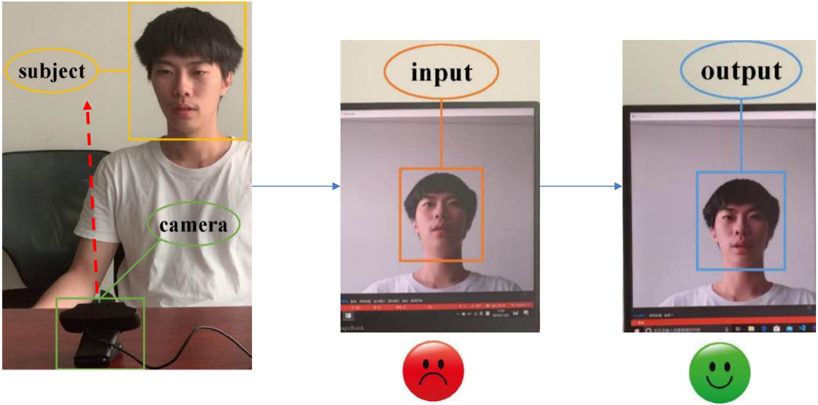
Input Output



Input Output

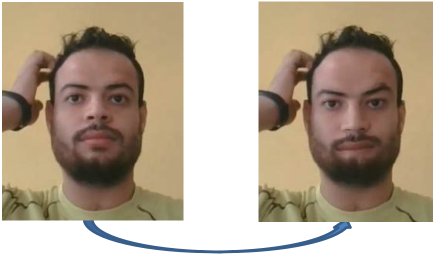
没有正对相机 → 正对相机

研究意义



问题挑战

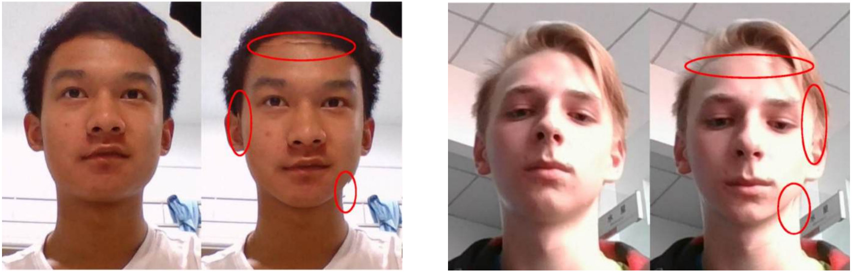
- 映射挑战: 输入输出间的复杂映射
- 融合挑战: 如何融合原始背景和新的前景



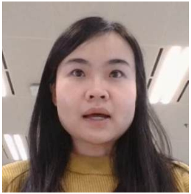
本质上为三维映射，基于二维映射的方法会失败

问题挑战

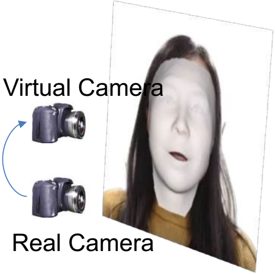
- 映射挑战: 输入输出间的复杂映射
- 融合挑战: 如何融合原始背景和新的前景



映射-3D



Input



Synthesized with virtual camera

映射-3D


Input


Dense Track


Virtual Overlay

融合-割缝优化

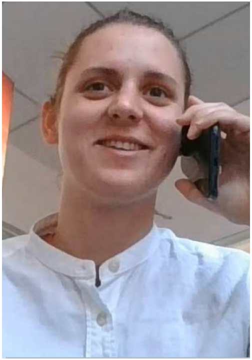
Outside the seam:
content from the original image $\boldsymbol{I}$

Inside the seam:
content from the rendered face $\boldsymbol{J}$

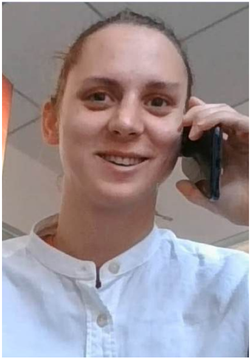
$$E_{\text{seam}} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{P}} \alpha(\boldsymbol{x},\boldsymbol{y}) \cdot (\|\boldsymbol{I}(\boldsymbol{x}) - \boldsymbol{J}(\boldsymbol{x})\|_2 + \|\boldsymbol{I}(\boldsymbol{y}) - \boldsymbol{J}(\boldsymbol{y})\|_2)$$

$\mathcal{P}$: adjacent pixels across the seam

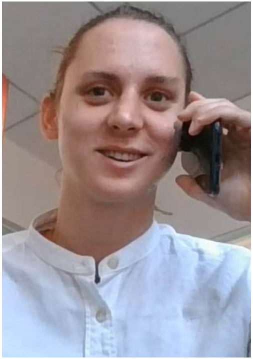
融合-Laplacian融合



Input

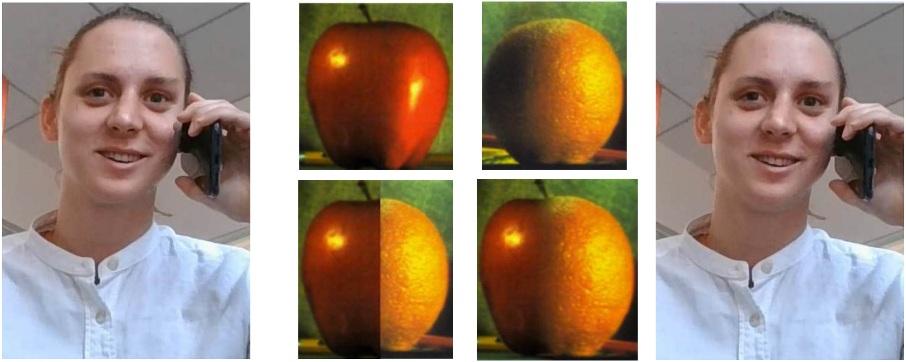


Overlay



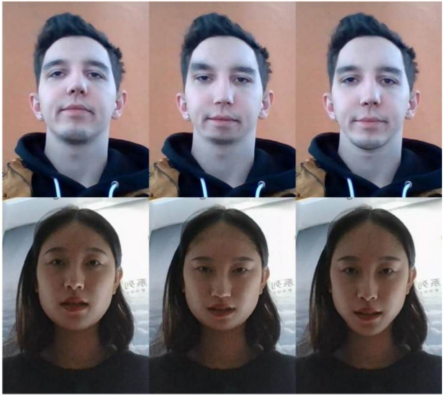
Seam
Optimization

融合-Laplacian融合



Laplacian Blending

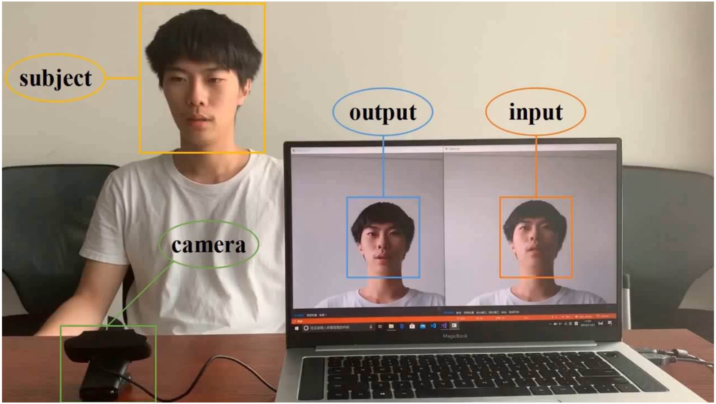
实验结果



(a) Input (b) Giger et al. (2014) (c) Ours

(a) Input (b) Giger et al. (2014) (c) Ours

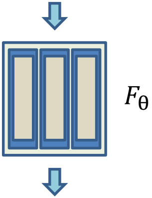
应用-实际效果



背景：神经隐式函数

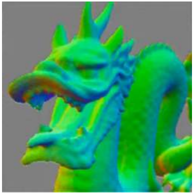
$$F_\theta(\mathbf{x}) = (y_1, y_2, \cdots, y_k)$$

Query coordinate

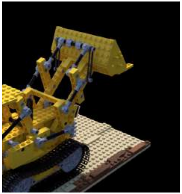


occupancy, SDF, color, …

Image regression
(x,y) → RGB

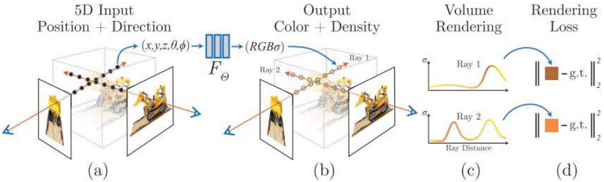


Shape regression
(x,y,z) → occupancy



Neural rendering
(x,y,z) → density,RGB

神经辐射场



- 隐式表示：$F_\Theta: (X, D) \rightarrow (c, \sigma)$

  X:空间点，D:观测X的view direction

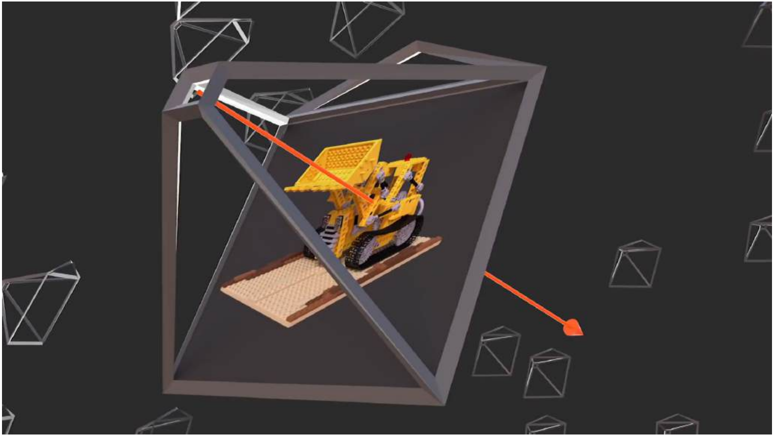
  c: 预测的点X的颜色，σ:预测的点X的density

- 体渲染：

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))\mathbf{c}_i \,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$

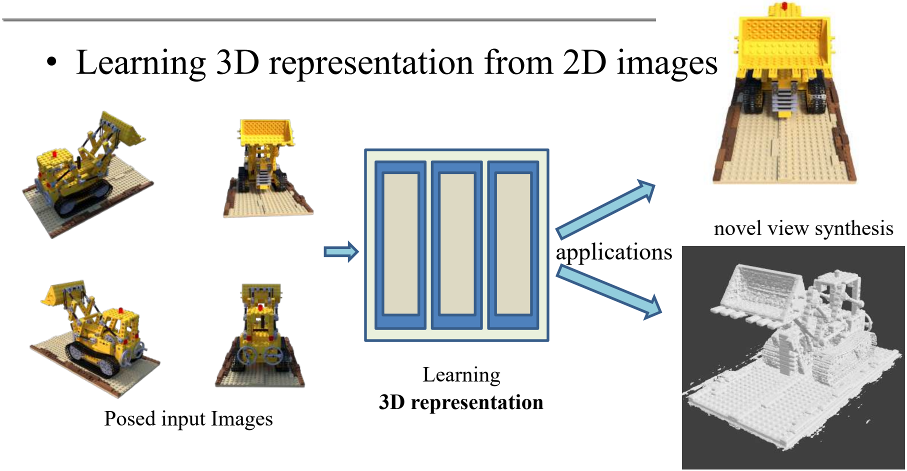
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis
Mildenhall et al. ECCV 2020

神经辐射场



神经隐式表示与逆向渲染

- Learning 3D representation from 2D images



Posed input Images

Learning

3D representation

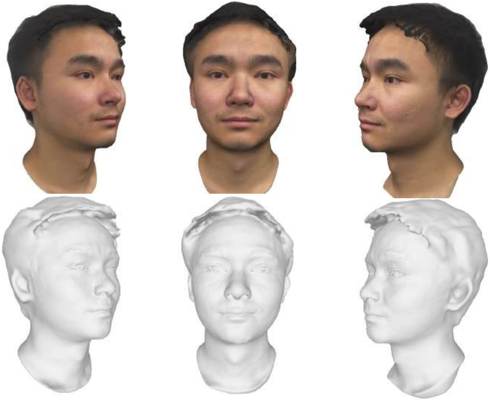
applications

novel view synthesis

3D reconstruction

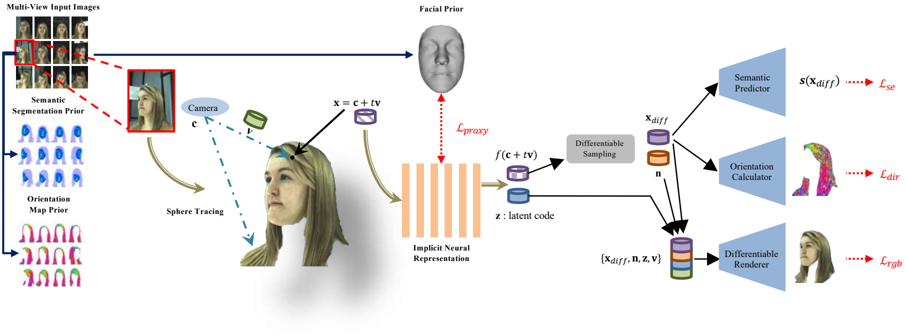
单手机高精度人头重建-静态

单个手机拍摄



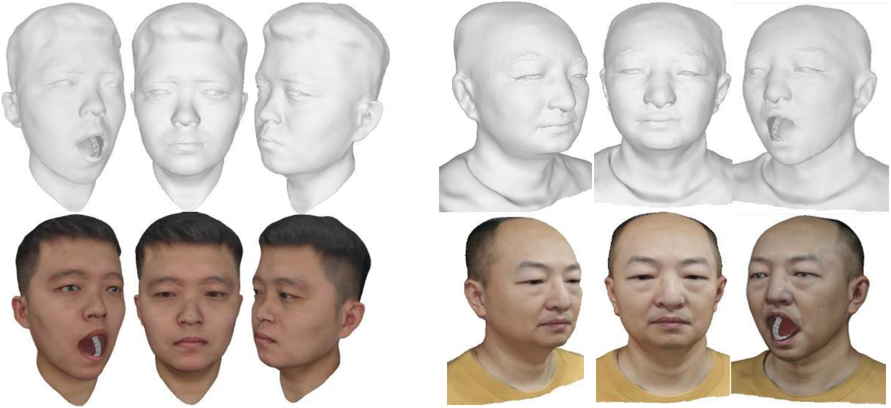
全自动重建高保真头部三维模型

先验引导的神经隐式重建

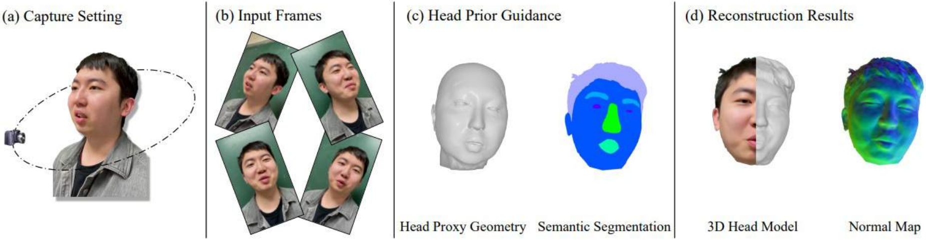


Prior-Guided Multi-View 3D Head Reconstruction
IEEE Transactions on Multimedia (TMM), 2021

更多结果展示



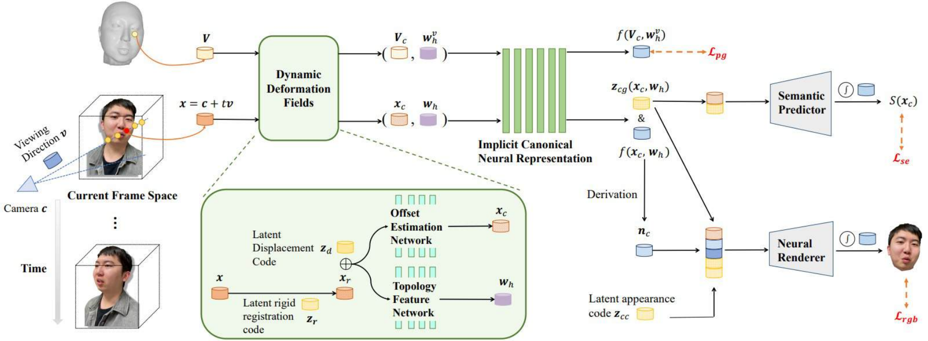
单目可驱动高精度人头重建



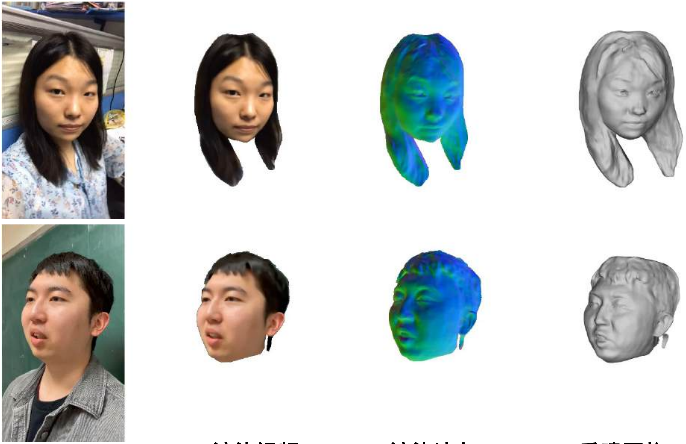
HeadRecon: High-Fidelity 3D Head Reconstruction from Monocular Video

算法流程

- 输入：单目说话视频
- 输出：可驱动的高精度三维人头模型



结果展示

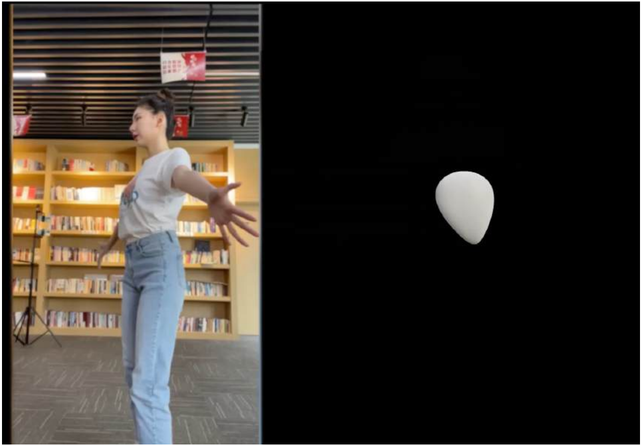


渲染视频 渲染法向 重建网格

单目自转视频的三维人体重建

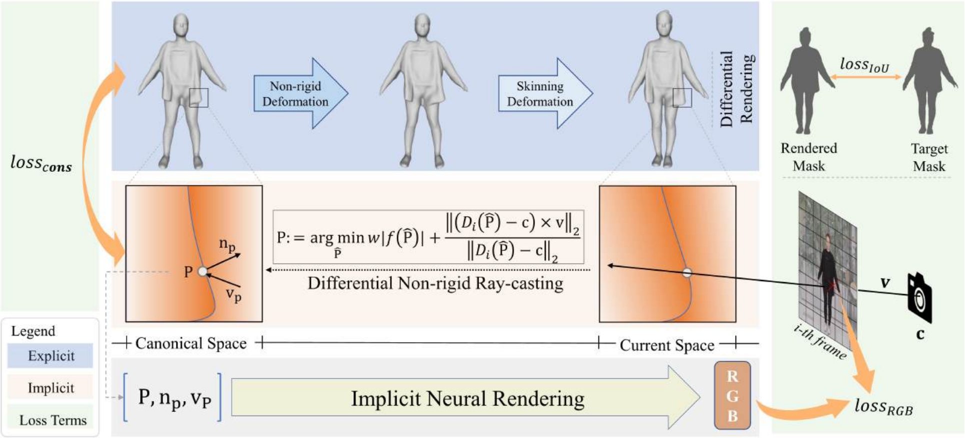
SelfRecon: Self Reconstruction Your Digital Avatar from Monocular Video
CVPR, Oral Presentation, 2022

期望的三维人体重建方式



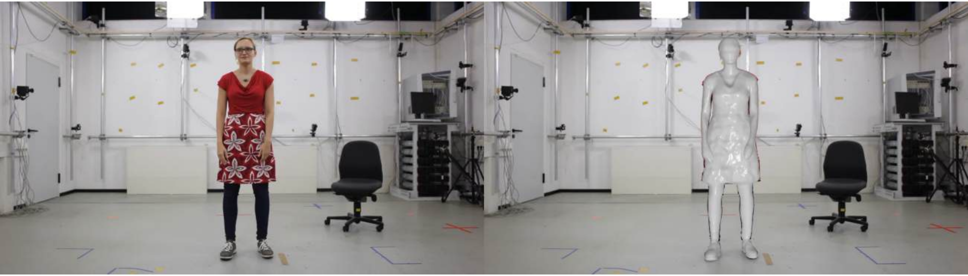
- ✓ Single camera
- ✓ Easy to capture
- ✓ High-fidelity result

算法：整体流程


Non-rigid Deformation
Skinning Deformation
Differential Rendering
$loss_{cons}$
$loss_{IoU}$
Rendered Mask
Target Mask
$\mathrm{n_p}$
P
$\mathrm{v_p}$
$\mathrm{P} := \arg\min_{\hat{\mathrm{P}}} w|f(\hat{\mathrm{P}})| + \frac{\|(D_i(\hat{\mathrm{P}}) - \mathrm{c}) \times \mathrm{v}\|_2}{\|D_i(\hat{\mathrm{P}}) - \mathrm{c}\|_2}$
Differential Non-rigid Ray-casting
v
c
i-th frame
Canonical Space
Current Space
Legend
Explicit
Implicit
Loss Terms
$\mathrm{P}, \mathrm{n_p}, \mathrm{v_P}$
Implicit Neural Rendering
R G B
$loss_{RGB}$

结果展示

应用至任意人体动作序列



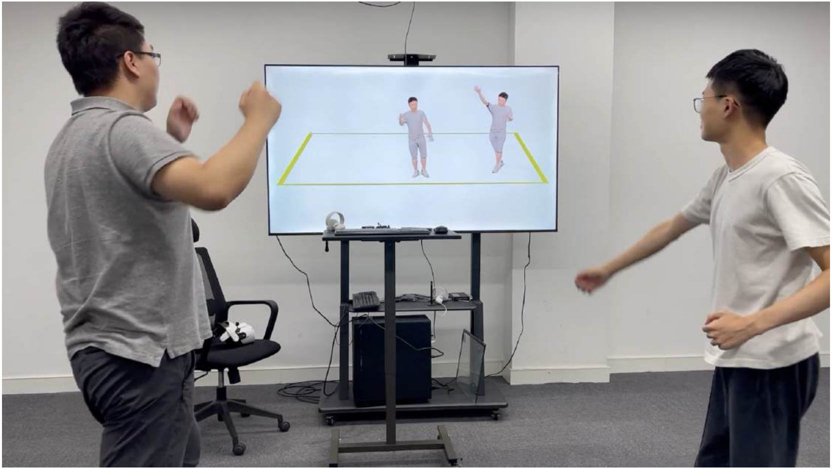
非刚性形变的可视化展示

Input video

Non-rigid deformation

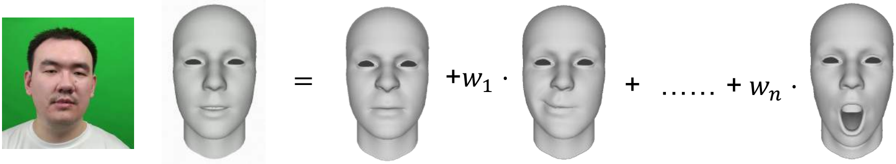
Whole deformations

应用展示



我们真的需要几何模型吗？

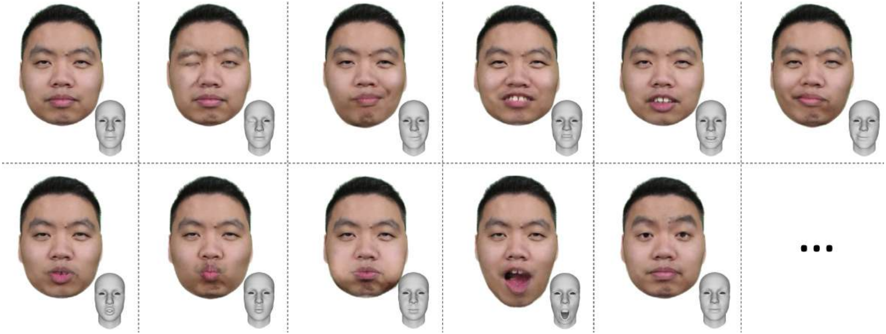
- 现有的数字人系统通常先对头部进行高精几何建模
- 例如，采用Blendshape表示来实现人脸驱动等



Reconstructing Personalized Semantic Facial NeRF Models From Monocular Video
Conditionally Accepted to the Journal Track of SIGGRAPH Asia, 2022

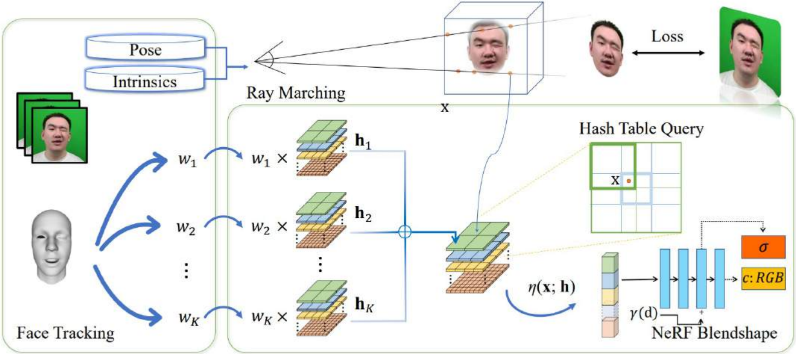
基于NeRF表示的个性化人头参数化

- 不同于3DMM等基于网格的三维几何参数化表示，我们采用基于NeRF的三维神经渲染参数化表示



拍摄输入的单目视频数据

算法流程


Pose
Intrinsics
Ray Marching
x
Loss
Face Tracking
w_1
w_2
w_K
w_1 ×
w_2 ×
w_K ×
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_K$
Hash Table Query
x
$\eta(\mathbf{x}; \mathbf{h})$
$\gamma(\mathrm{d})$
σ
c: RGB
NeRF Blendshape

训练过程展示

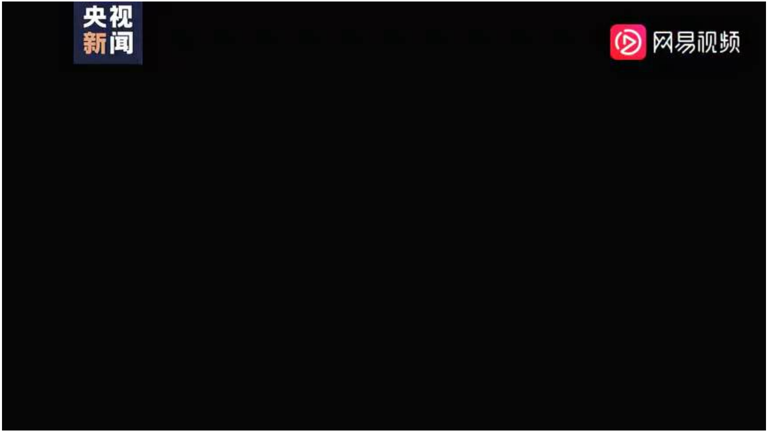
跨身份表情驱动

😀 单目相机　　😀 实时计算　　😀 微表情迁移

研究问题

With audio and pose input only, AD-NeRF could synthesize high-fidelity talking head video

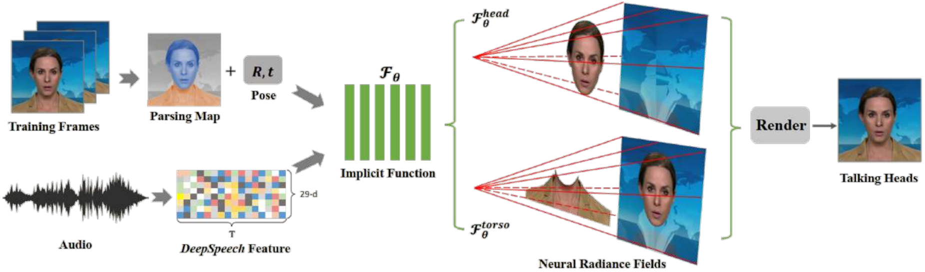
现有的语音驱动数字人



建模复杂、虚拟感强

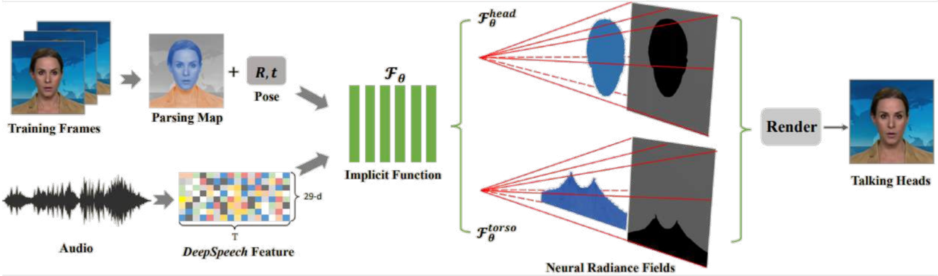
主要想法

利用神经辐射场直接学习语音信号到说话人视频的映射

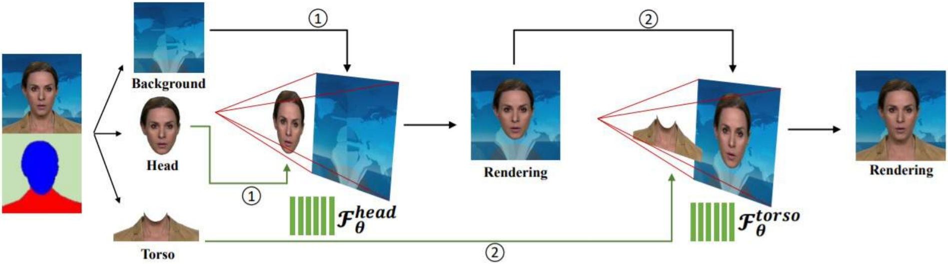


分辨率高、无需中间模态、支持姿态改变

算法流程


Training Frames
Parsing Map
+
R, t
Pose
Audio
29-d
T
DeepSpeech Feature
$\mathcal{F}_\theta$
Implicit Function
$\mathcal{F}_\theta^{head}$
$\mathcal{F}_\theta^{torso}$
Neural Radiance Fields
Render
Talking Heads

分离的NeRF场



头部和躯干运动的不一致性

实验结果-对比实验

Comparisons with Other Methods

实验结果-分离NeRF场

Ground-Truth

w.o. Individual Training

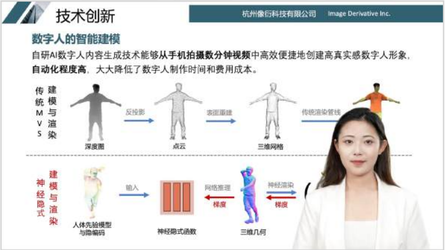
w. Individual Training

语音驱动高保真数字人应用

虚拟主持人/虚拟讲解员

电商虚拟主播

像衍科技
Image Derivative Inc.



技术创新

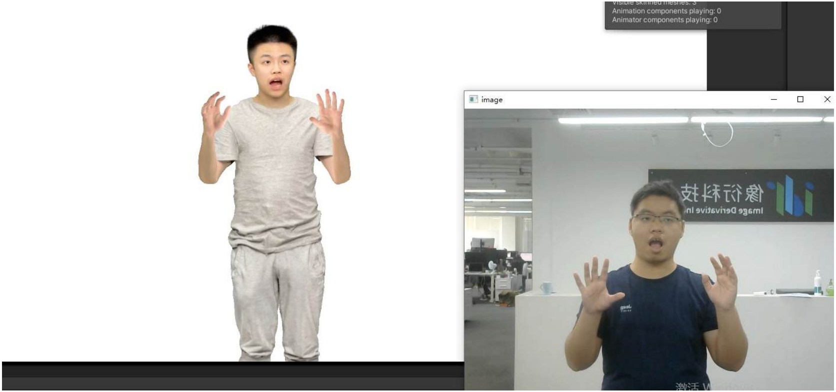
杭州像衍科技有限公司　Image Derivative Inc.

数字人的智能建模

自研AI数字人内容生成技术能够**从手机拍摄数分钟视频**中高效便捷地创建高真实感数字人形象，**自动化程度高**，大大降低了数字人制作时间和费用成本。

虚拟培训分身

完整数字人建模与驱动展示



Animation components playing: 0
Animator components playing: 0

相关应用：沉浸式视频会议、远程教育等

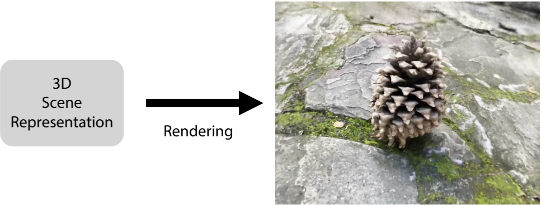
总结与展望

- 基于新的表示方式、端到端可微优化框架，数字人建模变得更便捷、高效、高保真
- 便捷、高效、高保真仍需不断提高
  - 单目设备在采集光照、角度、表情与动作幅度的鲁棒性
  - 移动端上建模
  - 进一步提高时空信息之间的精准融合

Neural Radiance Fields

张举勇
中国科学技术大学

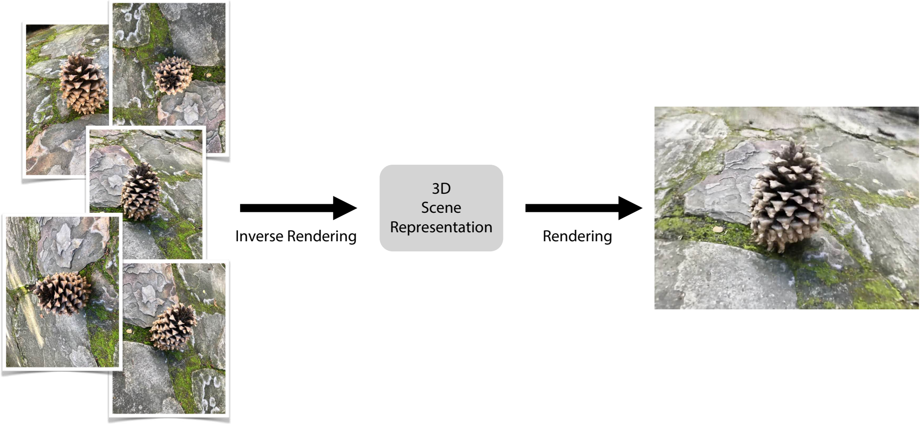
Computer vision as inverse rendering



3D
Scene
Representation

Rendering

Computer vision as inverse rendering

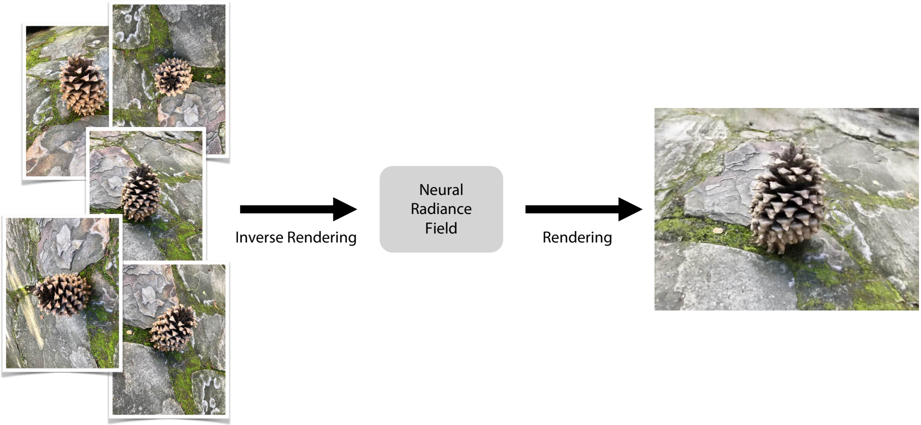


Inverse Rendering

3D
Scene
Representation

Rendering

Neural Radiance Fields (NeRF) as an approach to inverse rendering



Inverse Rendering

Neural Radiance Field

Rendering

Deep learning for 3D reconstruction

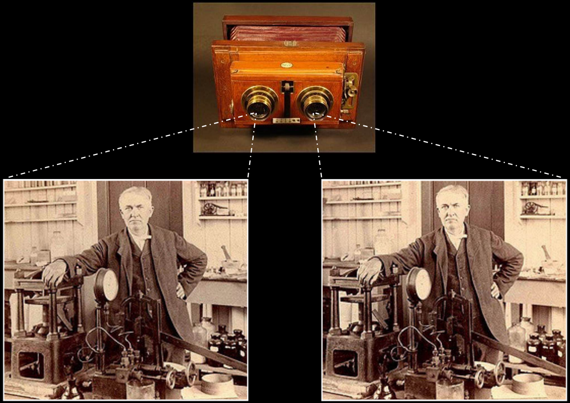
- Previously: we reconstruct geometry by running stereo or multi-view stereo on a set of images
  - “Classical” approach

- How can we leverage powerful tools of deep learning?
  - Deep neural networks
  - GPU-accelerated stochastic gradient descent

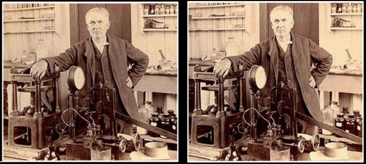
NeRF and related methods – Key ideas

- We need to create a loss function and a scene representation that we can optimize using gradient descent to reconstruct the scene
- *Differentiable rendering*

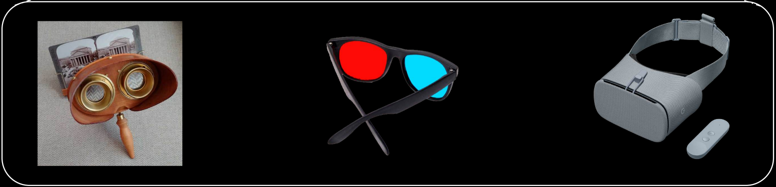
Side Topic: Stereo Photography



Stereo Photography



Viewing Devices

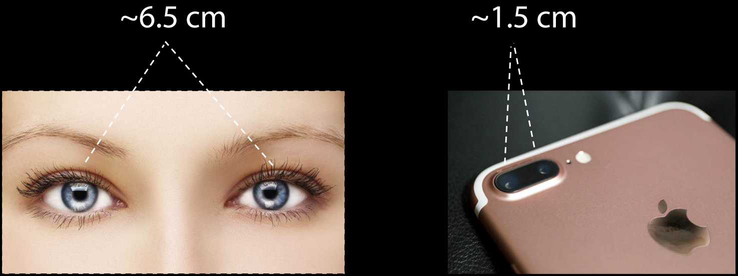


Stereo Photography

Queen Victoria at World Fair, 1851

Stereo Photography

Issue: Narrow Baseline

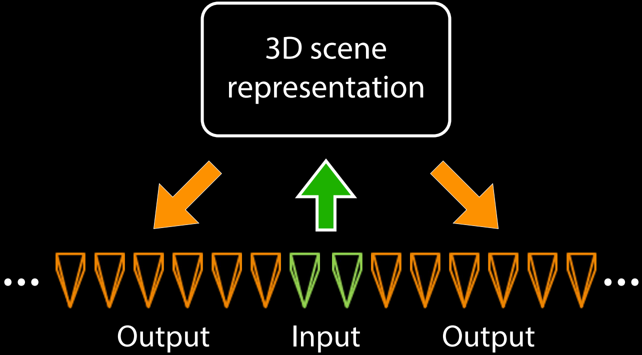



Left
MAZZER


Right
MAZZER


MAZZER

Problem Statement



3D scene representation

…

…

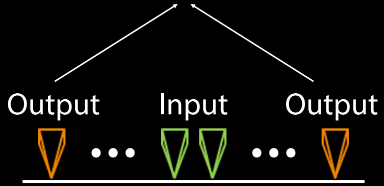
Output

Input

Output

Challenges

Extrapolation



Large disocclusion

Output

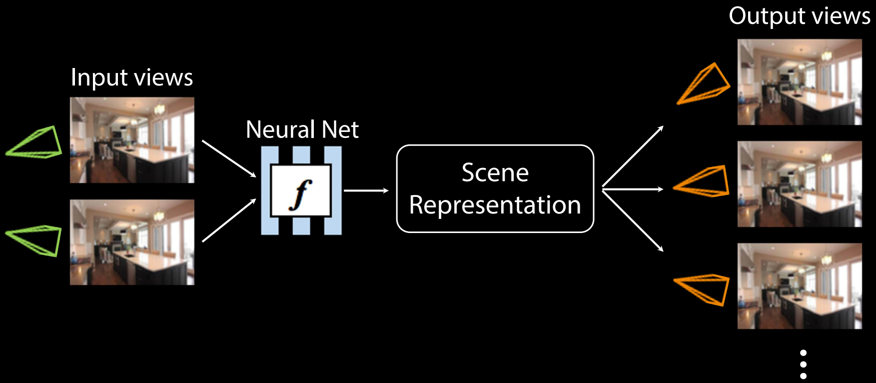
Input

Output

Non-Lambertian Effects

Reflections, transparencies, etc.

Neural prediction of scene representations



Input views

Neural Net

f

Scene Representation

Output views

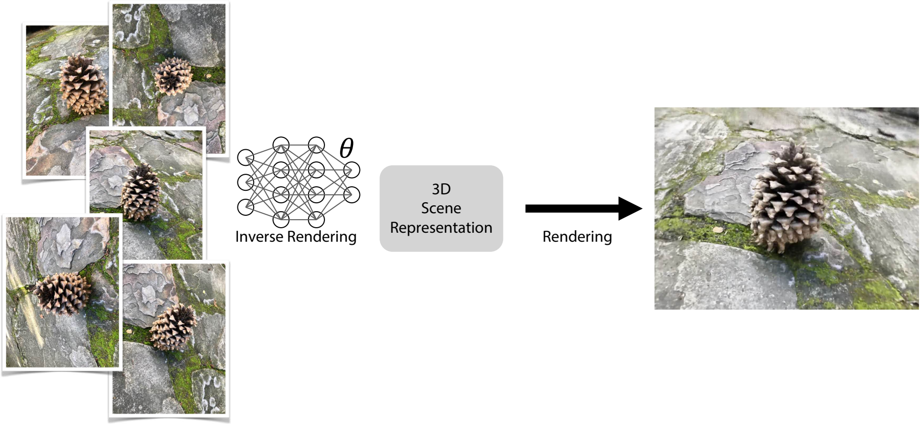
Computer vision as inverse rendering

Inverse Rendering

3D
Scene
Representation

Rendering

Paradigm 1: “Feedforward” inverse rendering

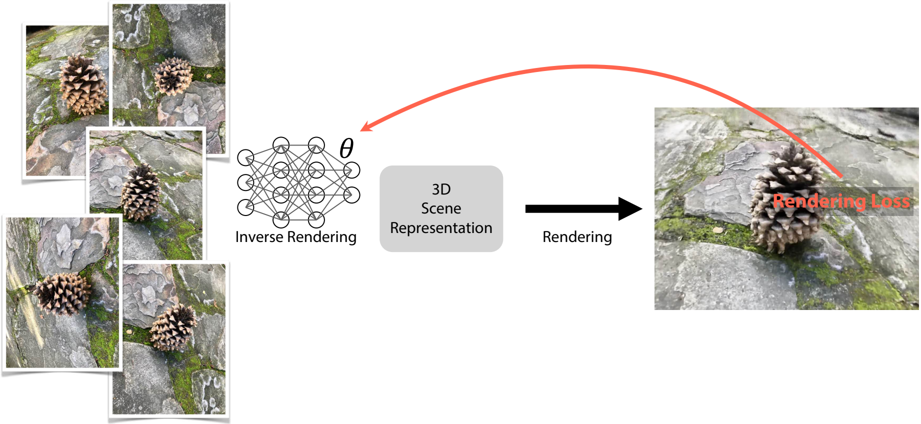


Inverse Rendering

3D
Scene
Representation

Rendering

Paradigm 1: “Feedforward” inverse rendering



θ

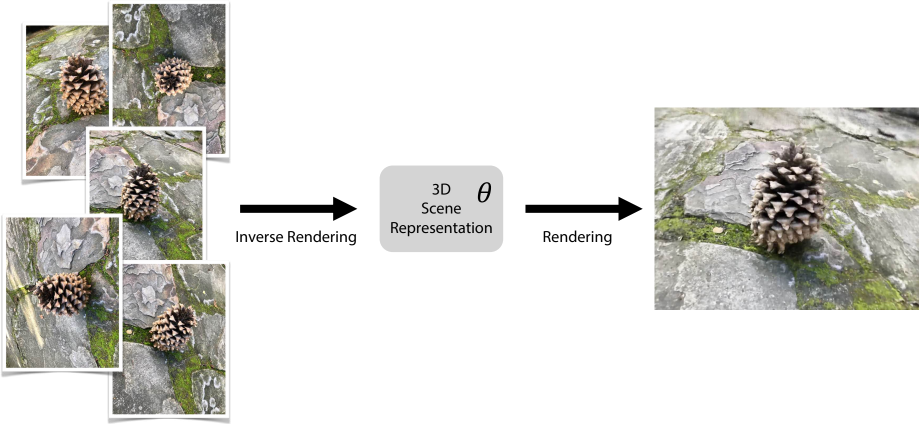
Inverse Rendering

3D
Scene
Representation

Rendering

Rendering Loss

Paradigm 2: “Render-and-compare”

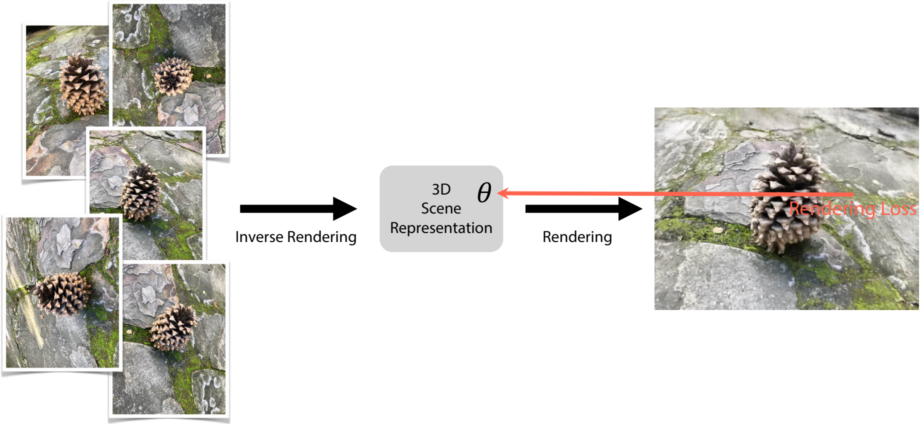


Inverse Rendering

3D θ
Scene
Representation

Rendering

Paradigm 2: “Render-and-compare”



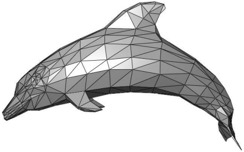
Inverse Rendering

3D Scene Representation θ

Rendering

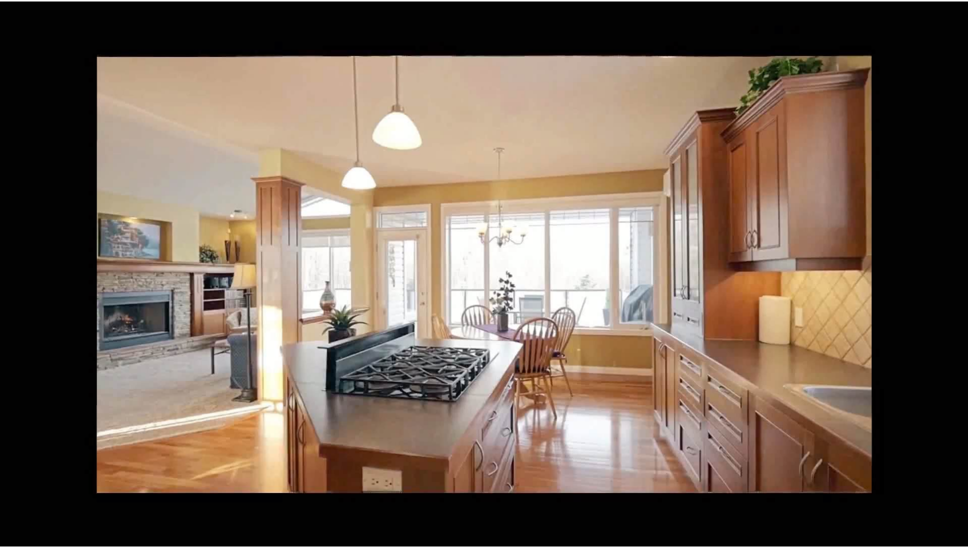
Rendering Loss

What representation to use?



- Could use triangle meshes, but hard to differentiate during rendering

- Multiplane images (MPIs) are easy to differentiate, but only allow for rendering a small range of views



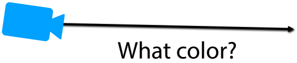
NeRF == Differentiable Rendering with a **Neural Volumetric Representation**

Barron et al 2021, Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields

Neural Volumetric Rendering

Neural Volumetric Rendering

querying the radiance value
along rays through 3D space

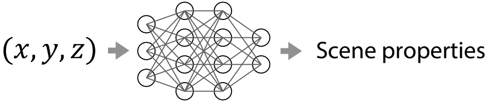


Neural Volumetric Rendering

continuous, differentiable
rendering model without concrete
ray/surface intersections

Neural Volumetric Rendering

using a neural network as a
scene representation, rather
than a voxel grid of data



(x, y, z) → Scene properties

Multi-layer Perceptron
(Neural Network)

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

ECCV 2020

Ben Mildenhall*

UC Berkeley

Pratul Srinivasan*

UC Berkeley

Matt Tancik*

UC Berkeley

Jon Barron

Google Research

Ravi Ramamoorthi

UC San Diego

Ren Ng

UC Berkeley

Given a set of sparse views of an object with known camera poses

Optimize a NeRF model

3D reconstruction viewable from any angle

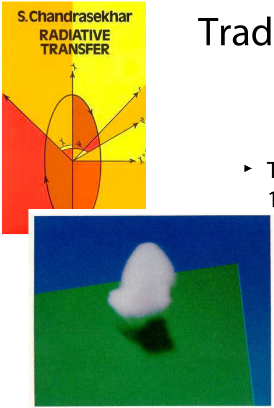
NeRF Overview

- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

NeRF Overview

- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

Traditional volumetric rendering

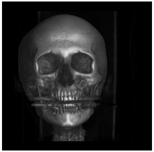


Ray tracing simulated cumulus cloud [Kajiya]

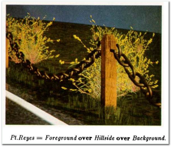
Chandrasekhar 1950, Radiative Transfer
Kajiya 1984, Ray Tracing Volume Densities

- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering

Traditional volumetric rendering



Medical data visualisation [Levoy]



Alpha compositing [Porter and Duff]

- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering
- Adapted for visualising medical data and linked with alpha compositing

Chandrasekhar 1950, Radiative Transfer
Kajiya 1984, Ray Tracing Volume Densities
Levoy 1988, Display of Surfaces from Volume Data
Max 1995, Optical Models for Direct Volume Rendering
Porter and Duff 1984, Compositing Digital Images

Traditional volumetric rendering

Physically-based Monte Carlo rendering [Novak et al]

- Theory of volume rendering co-opted from physics in the 1980s: absorption, emission, out-scattering/in-scattering
- Adapted for visualising medical data and linked with alpha compositing
- Modern path tracers use sophisticated Monte Carlo methods to render volumetric effects

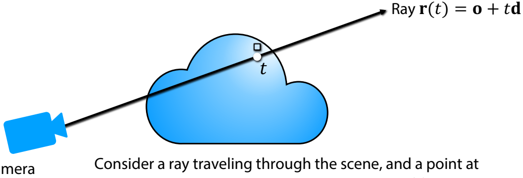
Chandrasekhar 1950, Radiative Transfer
Kajiya 1984, Ray Tracing Volume Densities
Levoy 1988, Display of Surfaces from Volume Data
Max 1995, Optical Models for Direct Volume Rendering
Porter and Duff 1984, Compositing Digital Images
Novak et al 2018, Monte Carlo methods for physically based volume rendering

Volumetric formulation for NeRF

Scene is a cloud of colored fog

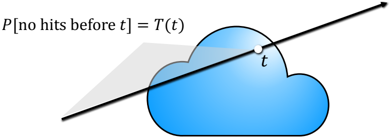
Max and Chen 2010, Local and Global Illumination in the Volume Rendering Integral

Volumetric formulation for NeRF



Consider a ray traveling through the scene, and a point at distance t along this ray. We look up its color $\mathbf{c}(t)$, and its opacity (alpha value) $\alpha(t)$

Volumetric formulation for NeRF



But t may also be blocked by earlier points along the ray.
$T(t)$: probability that the ray didn't hit any particles earlier.
$T(t)$ is called “transmittance”

Volume rendering estimation: integrating color along a ray

Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

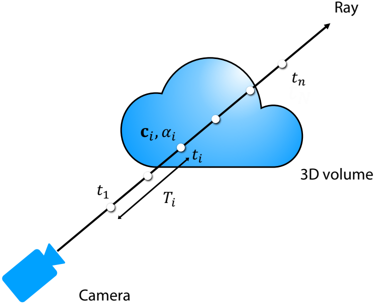
final rendered color along ray

weights

colors

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$



Computing the color for a set of rays through the pixels of an image yields a rendered image

Volume rendering estimation: integrating color along a ray

Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

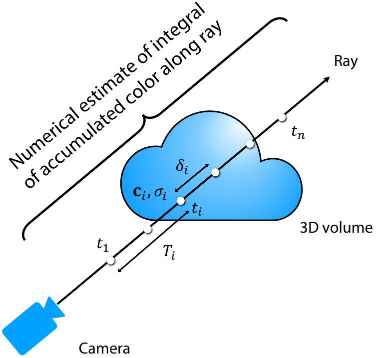
final rendered color along ray

weights

colors

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$



Slight modification: α is not directly stored in the volume, but instead is derived from a stored volume density sigma (σ) that is multiplied by the distance between samples delta (δ):

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$

Volume rendering estimation: integrating color along a ray

Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

final rendered color along ray

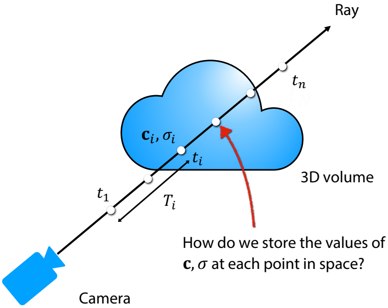
weights

colors

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

Computing the color for a set of rays through the pixels of an image yields a rendered image

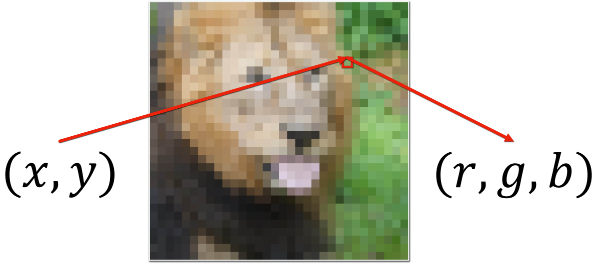


How do we store the values of $\mathbf{c}, \sigma$ at each point in space?

NeRF Overview

- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

Toy problem: storing 2D image data

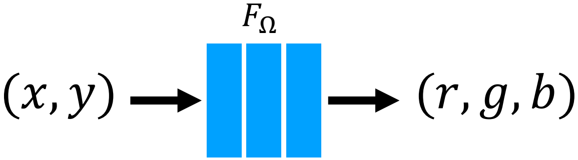


(x, y)

(r, g, b)

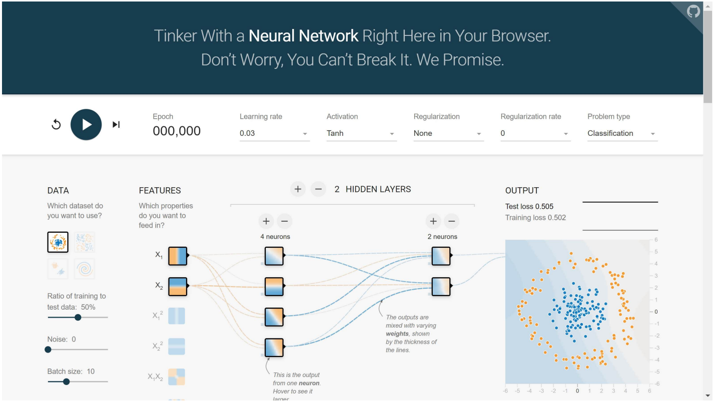
Usually we store an image as a 2D grid of RGB color values

Toy problem: storing 2D image data



What if we train a simple fully-connected network (MLP) to do this instead?

Recall the TensorFlow playground



Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Epoch 000,000

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

2 neurons

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

*This is the output from one **neuron**. Hover to see it larger*

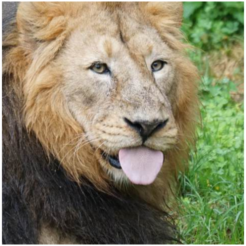
OUTPUT

Test loss 0.505

Training loss 0.502

Same concept as before, except we are computing an image, instead of a classifier!

Naive approach fails!



Ground truth image

Neural network output fit with gradient descent

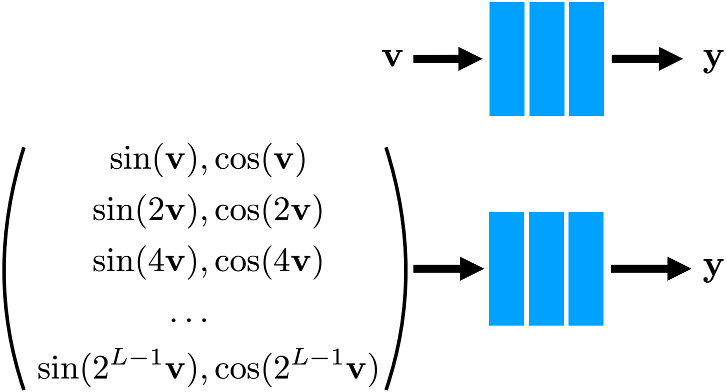
Problem:

“Standard” coordinate-based MLPs cannot represent high frequency functions

Solution:

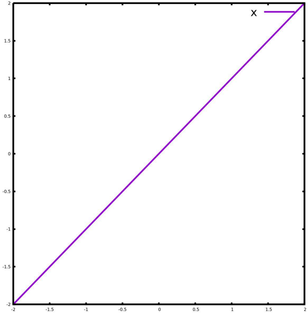
Pass input coordinates through a high frequency mapping first

Example mapping: “positional encoding”

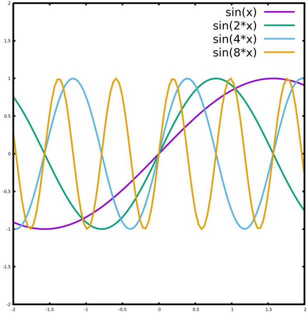


$$\begin{pmatrix} \sin(\mathbf{v}), \cos(\mathbf{v}) \\ \sin(2\mathbf{v}), \cos(2\mathbf{v}) \\ \sin(4\mathbf{v}), \cos(4\mathbf{v}) \\ \dots \\ \sin(2^{L-1}\mathbf{v}), \cos(2^{L-1}\mathbf{v}) \end{pmatrix}$$

Positional encoding

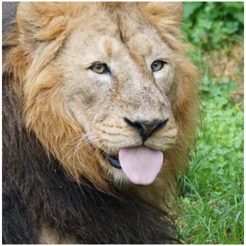


Raw encoding of a number ***x***

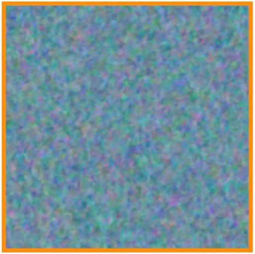


“Positional encoding” of a number ***x***

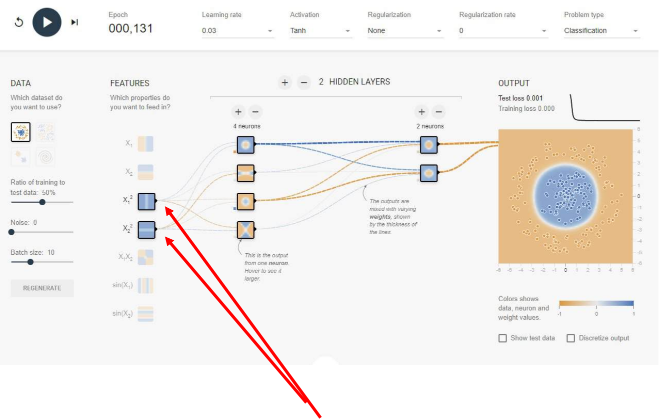
Problem solved!


Ground truth image


Neural network output without high frequency mapping


Neural network output with high frequency mapping

Sometimes a better input encoding is all you need



Epoch 000,131

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA
Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES
Which properties do you want to feed in?

This is the output from one *neuron*. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

OUTPUT

Test loss 0.001
Training loss 0.000

Colors shows data, neuron and weight values.

☐ Show test data ☐ Discretize output

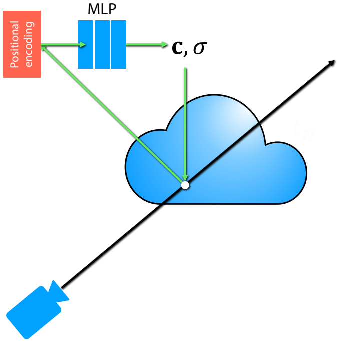
Recall “squared” encoding in TensorFlow Playground

NeRF Overview

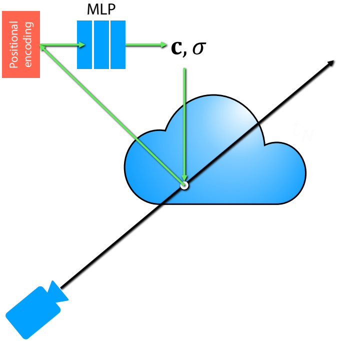
- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

NeRF = volume rendering + coordinate-based network

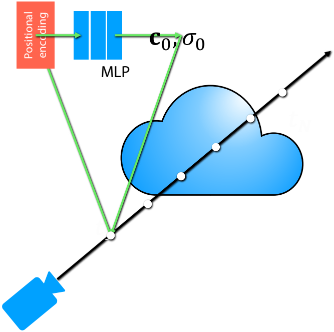
How do we store the values of $\mathbf{c}, \sigma$ at each point in space



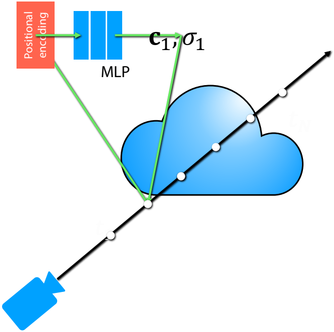
How do we store the values of $\mathbf{c}, \sigma$ at each point in space



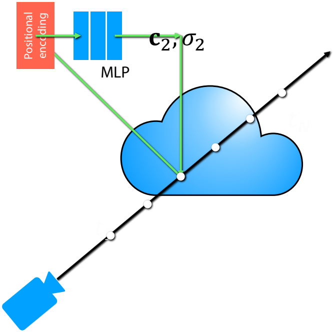
How do we store the values of $\mathbf{c}, \sigma$ at each point in space



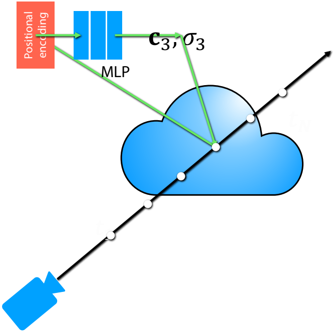
How do we store the values of $\mathbf{c}, \sigma$ at each point in space



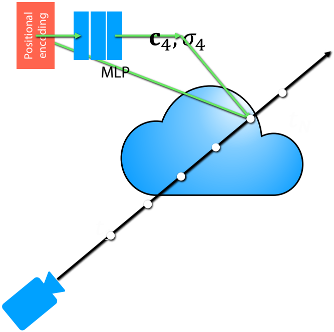
How do we store the values of $\mathbf{c}, \sigma$ at each point in space



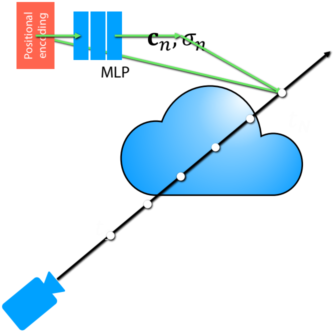
How do we store the values of $\mathbf{c}, \sigma$ at each point in space



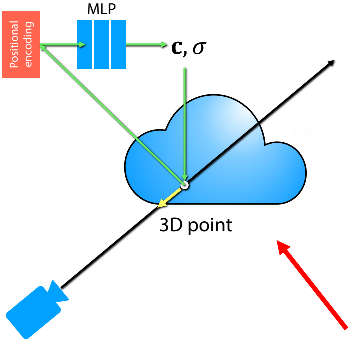
How do we store the values of $\mathbf{c}, \sigma$ at each point in space



How do we store the values of $\mathbf{c}, \sigma$ at each point in space

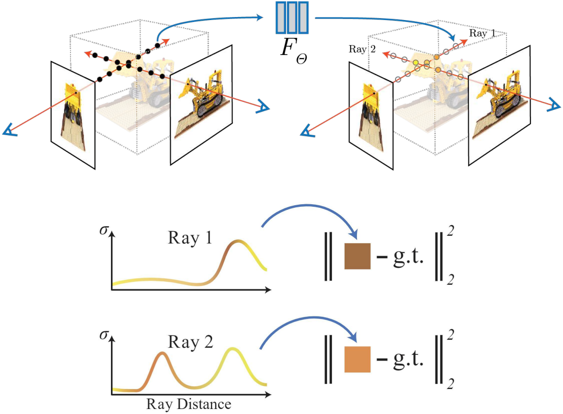


Extension: view-dependent field

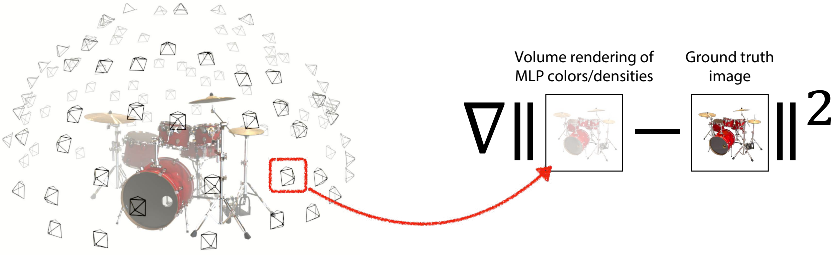


Include the ray direction in the input to the MLP → allows for capturing and rendering view-dependent effects (e.g., shiny surfaces)

Putting it all together



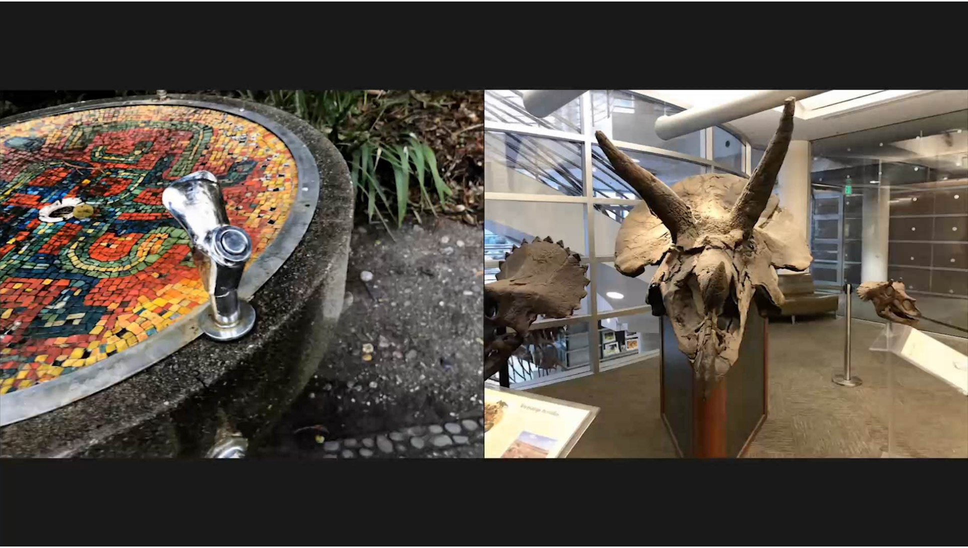
Train network using gradient descent to reproduce all input views of scene



Volume rendering of MLP colors/densities

Ground truth image

Results



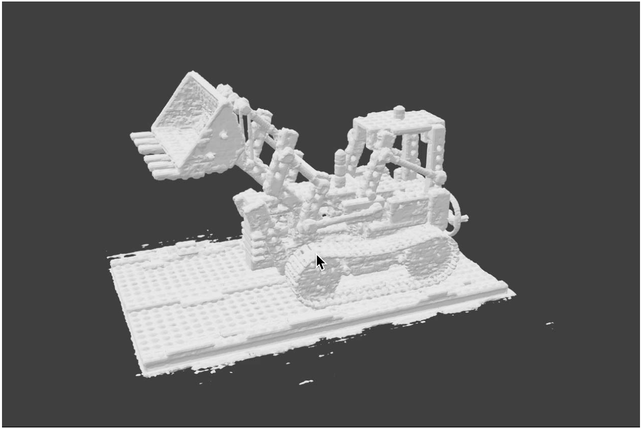
NeRF encodes convincing view-dependent effects using directional dependence

NeRF encodes convincing view-dependent effects using directional dependence

NeRF encodes detailed scene geometry with occlusion effects

NeRF encodes detailed scene geometry with occlusion effects

NeRF encodes detailed scene geometry



Summary

- Represent the scene as volumetric colored “fog”
- Store the fog color and density at each point as an MLP mapping 3D position (x, y, z) to color c and density σ
- Render image by shooting a ray through the fog for each pixel
- Optimize MLP parameters by rendering to a set of known viewpoints and comparing to ground truth images