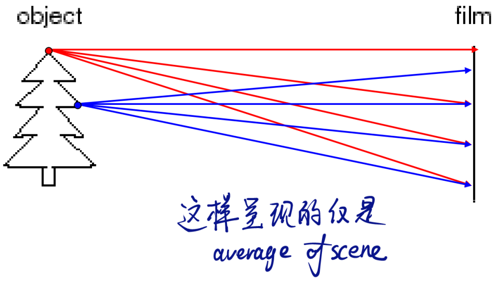
Midterm Review (First half lecture note)

Scope: Beginning → Oct. 9, stereo

Lec 1: Camera

物体不是主动发出光的，而是光照在物体上反射。以下为分析简便，

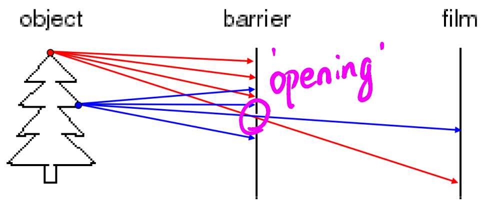


认为是物体发射 rays of light.

How to record (film / take picture) of them?

If we just put a film ......

⇒ Average of scene! Blurry!

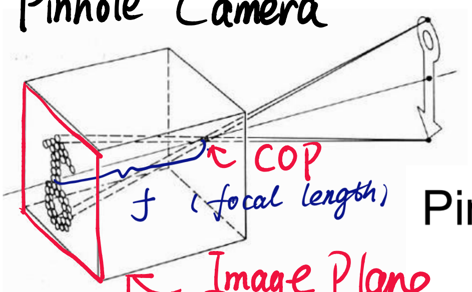


What if we add a barrier?

- Reduce blurring
- Opening is known as aperture (光圈)
- Images are upside-down

With this, we can build pinhole camera!

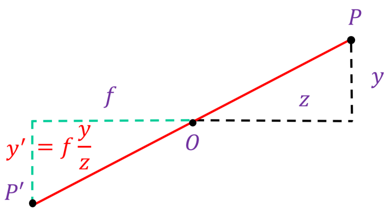
Pinhole Camera



Pinhole model:

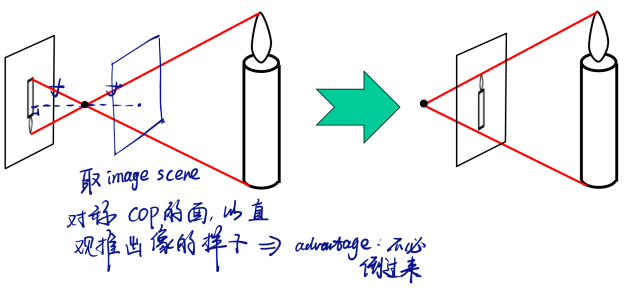
- Captures **pencil of rays** – all rays through a single point
- The point is called **Center of Projection (COP)** 投影中心
- The image is formed on the **Image Plane**
- **Effective focal length *f*** is distance from COP to Image Plane

Δ: focal length: COP → Image Plane



In this model, using similar triangles, we can compute the coordinate mapping:

$$(x, y, z) \Rightarrow \left(f\frac{x}{z}, f\frac{y}{z}\right)$$



Also, we can use the trick shown on the left to avoid inversion:

We can (and most of time we do) pretend that the image plane is in front of camera center.

3D感知来于：双目立体视觉 or 移动+体验.

Why did evolution opt for such strange solution?

- Nice to have a passive, long-range sensor
- Can get 3D with stereo or by moving around, plus experience

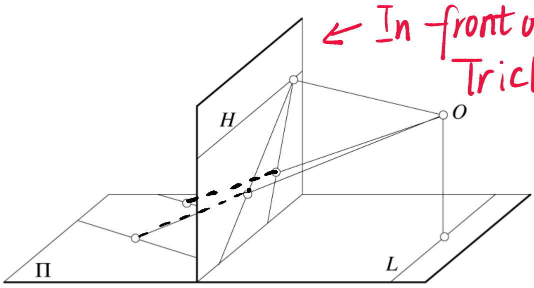
But our eyes, who generate images of 2D for us, can help us perceive 3D world. Why is that so?

Passive: We capture lights, instead of 'using sonar to detect'

Long-range: Light can be far away

Stereo: 两只眼从略微不同角度看世界，会产生两个略微不同的2D，大脑通过比较这个差异，可精确计算物体远近

But from 3D → 2D, their are perspective projection:



← In front of camera

Trick!

Parallel lines in real 3D world don't go parallel in 2D images

Orthographic or parallel projection

正交 平行投影

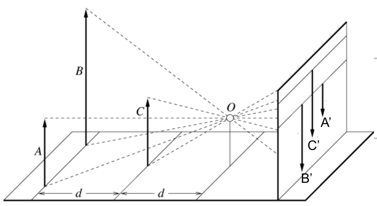
- What happens if we walk infinitely far away and zoom infinitely far in?

Left: Intersect

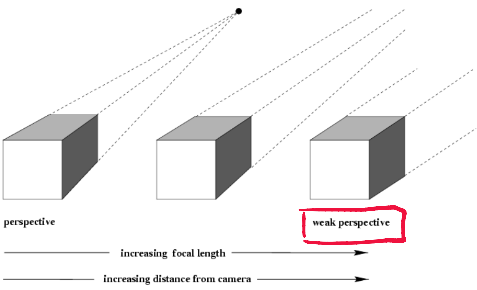
Right: Almost parallel ⇒

平行投影：所有投影线平行

正交：不仅平行，而且垂直于投影平面.



同时，depth也开始不可信……



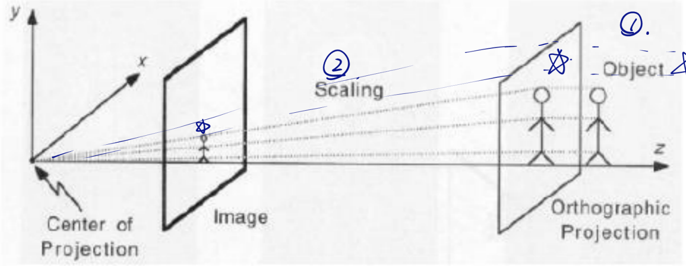
For perspective, if we let COP far away from obj and let focal length also rather far?

i.e., : What happens if we walk infinitely far away and zoom infinitely far in?

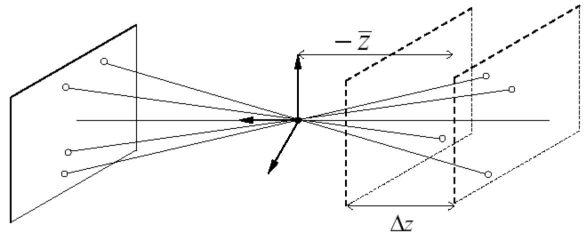
⇒ We are getting close to parallel perspective.

Or we can use another approximate modeling:

Scaled Orthographic or "Weak Perspective"



First orthgraphic one, then normal scaling



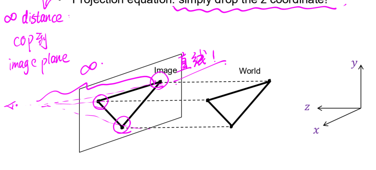
If $\Delta z << -\bar{z}$: $x' \approx -mx$ $\quad m = -\frac{f}{\bar{z}}$
$y' \approx -my$

Justified if scene depth is small relative to average distance from camera

If $\Delta z << -\bar{z}$, ortho first ⇐ then scale.

Orthographic or parallel projection

- Special case of perspective projection 透视投影
  - Distance from center of projection to image plane is infinite
  - Projection equation: simply drop the z coordinate!



$\boxed{(x, y, z) \rightarrow (x, y)}$

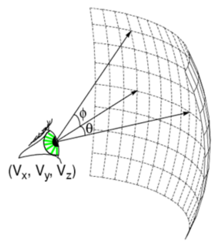
Wrap up.

Another Projection: Spherical One

(Actually more similar to how our eyes see!) $(\theta, \phi, z) \rightarrow (\theta, \phi)$

☆: Doesn't depend on focal length

Spherical Projection



What if PP is spherical with center at COP?

In spherical coordinates, projection is trivial:

$(\theta, \phi, f) \rightarrow (\theta, \phi)$

Note: doesn't depend on focal length f! ☆

Now if wanting to build a camera. a few things to calibrate ······ Like Aperture.

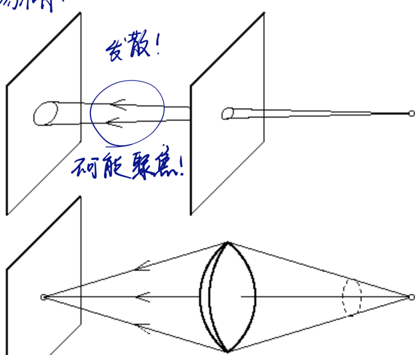
⇒ How aperture affects image!

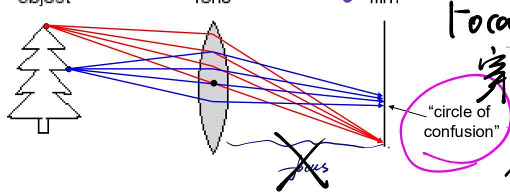
Too large & Too small ⇒ blurry

↳ Easy to understand ↓ (衍射)

That's why we need a len: ① less light to get through ② Diffraction

可见 pin hole size (aperture) 很影响呈像大小于质量! 很容易糊!



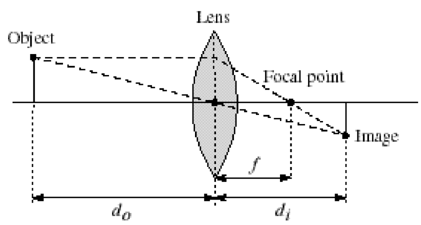


Focal Point: 平行光穿过len后汇聚的点。

A lens focuses light onto the film

- There is a specific distance at which objects are "in focus"
  - other points project to a "circle of confusion" in the image
- Changing the shape of the lens changes this distance

$$\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f}$$

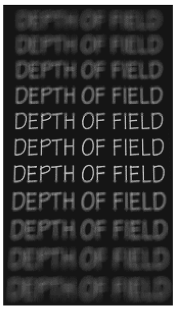


Thin lens equation: $\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f}$

d_o: 物 → len $\quad d_i$: "in focus" → len

Depth of Field 景深

看起来清晰可接受的范围或区域



光圈控制景深!!

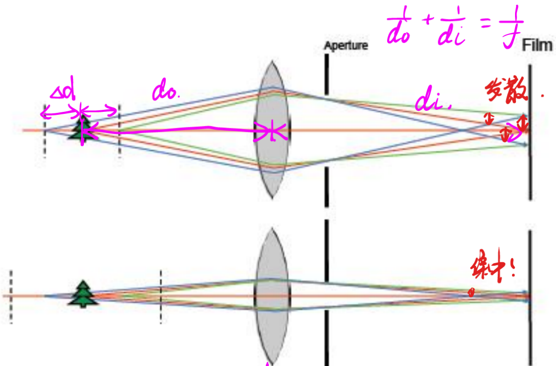
Aperture ↑, DOF ↓

If Aperture ↓, DOF ↑, and need more exposure

Another topic is Depth of Field (DOF)

Settled d_i & f ⇒ settled d_o ??

Aperture controls Depth of Field



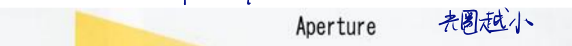
光圈越大，景深越小. 因为DOF深/浅物体进来的ray少且集中 点object来的ray.

Changing the aperture size affects depth of field

- A smaller aperture increases the range in which the object is approximately in focus
- But small aperture reduces amount of light – need to increase exposure 但同时, ray少, 因此曝光要多些

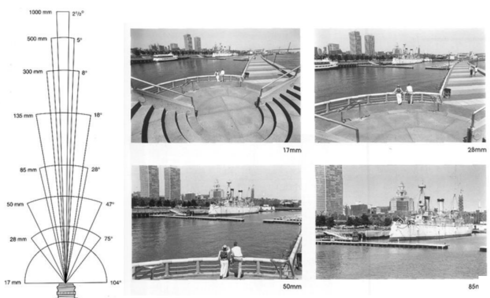
F-number: focal length / aperture diameter

$$F = \frac{\text{focal length } (f)}{\text{aperture diameter}}$$, F越大, 在f不变下, 光圈越小



In camera community: $\text{F-number} = \frac{\text{focal length}}{\text{aperture diameter}} \propto \text{DOF}$

Field of View (Zoom)



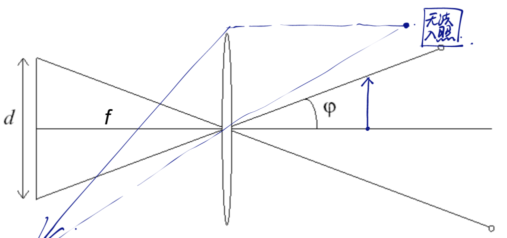
From London and Upton

Another topic: FOV (Field of view).

With fixed image size, f ↑, FOV ↓

$\varphi = \tan^{-1}(d/2f)$

FOV depends of Focal Length

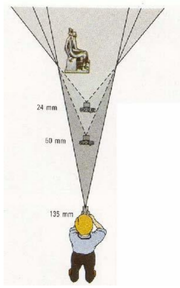


Size of field of view governed by size of the camera retina:

$$\varphi = \tan^{-1}\left(\frac{d}{2f}\right)$$ $\varphi = \arctan\left(\frac{d}{2f}\right)$

Smaller FOV = larger Focal Length

Field of View / Focal Length

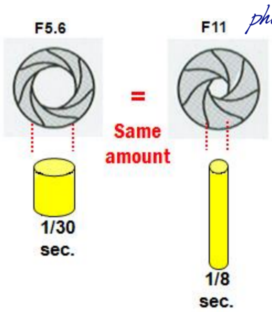


Large FOV, small f
Camera close to car

Small FOV, large f
Camera far from the car

Exposure: shutter speed vs. aperture

进入ray. or should say photon 数量



Finally: exposure: shutter speed ⇕ aperture.

How much photon / rays of light?

But lens aren't flawless:

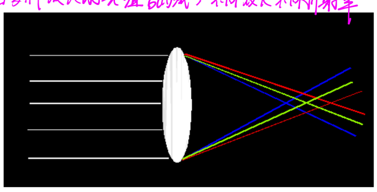
Lens Flaws: Chromatic Aberration 色差.

学瑕疵

色散 Dispersion: wavelength-dependent refractive index
- (enables prism to spread white light beam into rainbow)

Modifies ray-bending and lens focal length: f(λ)

白光实则由多种波长的光组合而成；不同波长不同折射率



color fringes near edges of image

Corrections: add 'doublet' lens of flint glass, etc.

① Rays of different wave length → diff ray-bending → Dispersion → Chromatic Aberration.

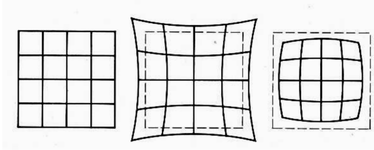
易于 image obj 边缘处产生!

Radial Distortion (*e.g.* 'Barrel' and 'pin-cushion')

径向畸变

straight lines curve around the image center

Radial Distortion



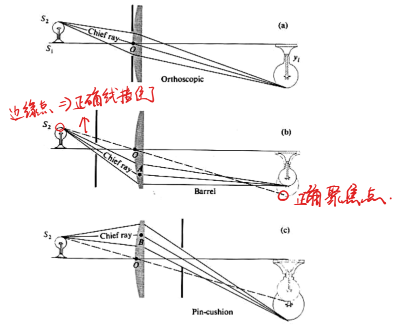
< 中心放大倍率 边缘放大倍率 >.

Radial distortion of the image

- Caused by imperfect lenses 不能聚焦良好
- Deviations are most noticeable for rays that pass through the edge of the lens

边缘最显著.

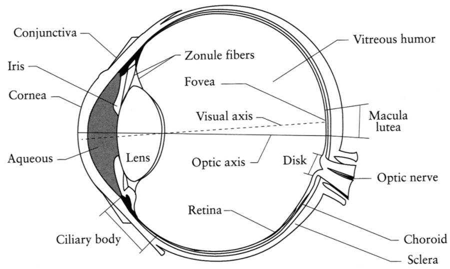
Radial Distortion



② Bending can be different at the center & fringe part of len

Lec2. Capturing Light

The Eye



The human eye is a camera!

- **Iris** - colored annulus with radial muscles 虹膜⇒光圈 控制
- **Pupil** - the hole (aperture) whose size is controlled by the iris 瞳孔⇒光圈
- What's the "film"?
  - photoreceptor cells (rods and cones) in the **retina** 胶片 ⇑ 视网膜

Slide

Our eye:

Iris control Aperture

Pupil → Aperture

Retina → Film

On retina:

cone & rod ← 暗, gray

↑亮, color

Two types of light-sensitive receptors

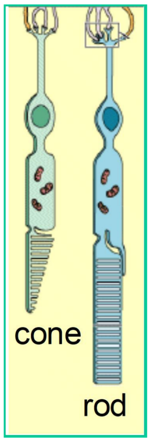
Cones
- cone-shaped
- less sensitive
- operate in high light
- color vision

白天 ⇓ color vision

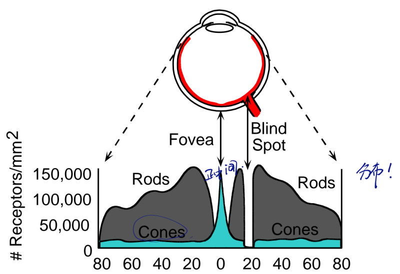
Rods
- rod-shaped
- highly sensitive
- operate at night
- gray-scale vision

夜晚 暗. ⇓ gray-scale.

两种光敏接收器



Distribution of Rods and Cones



Night Sky: why are there more stars off-center? 斜看星更密

© Stephen E. Palmer, 2002

Retina 小角度范围内分布绝大部分cone.

而其它大角度分布大量 cones

同时, 光是一种wave,

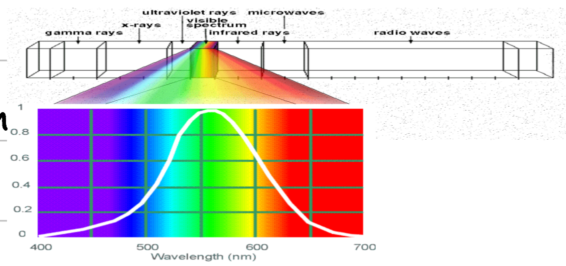
人可感知光的wavelength

有限(400–700nm) ⇒

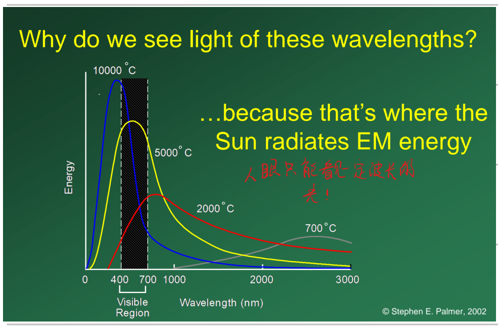
这恰好也是太阳

温度发出光线的wavelength分布中占比亮的

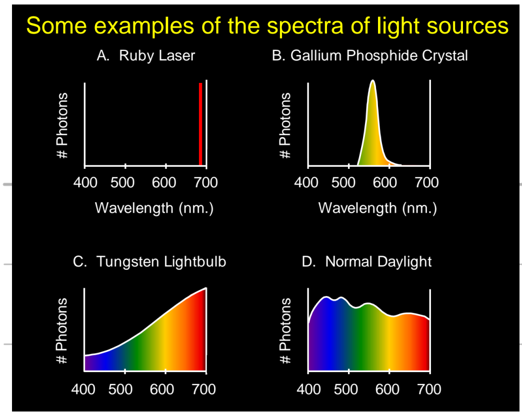
wavelength zone !!

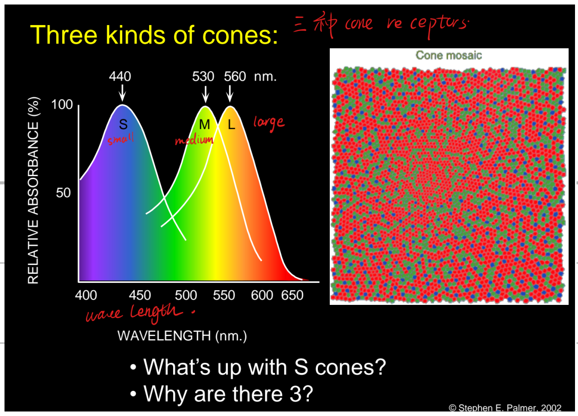


Human Luminance Sensitivity Function



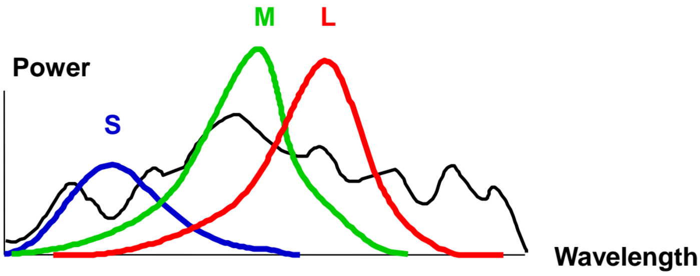
Any patch of light can be completely described physically by its spectrum: the number of photons (per time unit) at each wavelength 400 - 700 nm.





三种 cone，吸收波长范围如左图所示

Trichromacy



每个 cone 视为 filter，关心"真实光谱"中有多少能量落在 cone 的敏感区域内

☆: Each cone produces one number

⇒ Represent spectrum with 3-dim data

Rods and cones act as *filters* on the spectrum

- To get the output of a filter, multiply its response curve by the spectrum, integrate over all wavelengths
  - ☆ Each cone yields one number
  - 不能用三种就表达所有光谱
- How can we represent an entire spectrum with 3 numbers?
- We can't! A lot of the information is lost
  - As a result, two different spectra may appear indistinguishable
    - » such spectra are known as **metamers** 同色异谱物

Slide by Steve Seitz

Metamers：两种光，物理波长不同，但人眼看颜色相同

颜色恒常性 ⇒ 让我们感知物体真实颜色

Color Constancy: the ability to perceive the invariant color of a surface despite ecological Variations in the conditions of observation.

而非反射到我们眼中的物理颜色

Another of these hard inverse problems:
Physics of light emission and surface reflection underdetermine perception of surface color

同时，我们对颜色的感知也可能受环境/心理影响。"看颜色"并非"所见即所得"

感知到物体表面不变的颜色尽管观察的环境条件（如光照）在不断变化

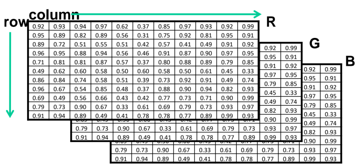
Camera White Balancing

（自动）白平衡

- Manual
  - Choose color-neutral object in the photos and normalize
- Automatic (AWB)
  - Grey World: force average color of scene to grey
  - White World: force brightest object to white

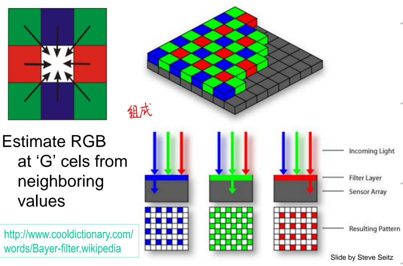
Image representation

- Images represented as a matrix
- Suppose we have a NxM RGB image called "im"
  - im(1,1,1) = top-left pixel value in R-channel
  - im(y, x, b) = y pixels down, x pixels to right in the b^th channel
  - im(N, M, 3) = bottom-right pixel in B-channel



RGB Bayer Grid

Practical Color Sensing: Bayer Grid



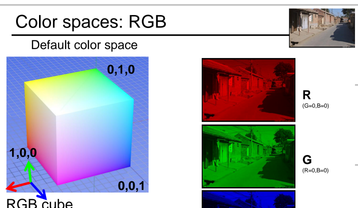
Estimate RGB at 'G' cels from neighboring values

http://www.cooldictionary.com/words/Bayer-filter.wikipedia

Slide by Steve Seitz

Color spaces: RGB

Default color space



RGB cube

- Easy for devices
- But not perceptual (t,t,t) t取0/1 黑 白
- Where do the grays live?
- Where is hue and saturation?

Image from: http://en.wikipedia.org/wiki/File:RGB_color_solid_cube.png

RGB color space：Gray：(t,t,t) t取0/1（黑/白）

Hue：绕着 Gray Diagonal 角度.

Saturation：到 Gray Diagonal 的距离.

Lec3. Pixels and Images

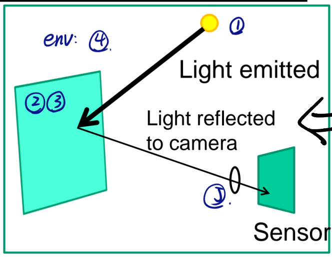
We can think of an **image** as a function, f, from R^2 to R: 强度.

- $f(x, y)$ gives the **intensity** at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - $f: [a,b]\times[c,d] \rightarrow [0,1]$

What is an image now? We use RGB! So for (x, y), we need intensity of RGB channels. So the intensity of light matters. How a pixel get its value? See left.

How does a pixel get its value?

Major factors
1. Illumination strength and direction
2. Surface geometry
3. Surface material
4. Nearby surfaces
5. Camera gain/exposure



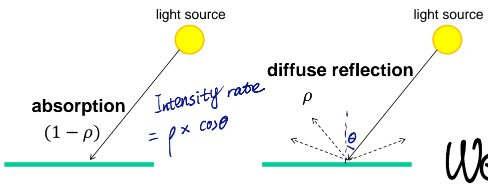
How does ② & ③ decide how intense the reflected light is?

Some light is absorbed (function of albedo ρ)
Remaining light is scattered (diffuse reflection)
Examples: soft cloth, concrete, matte paints

⇒ A simple and practical model:

Lambertian Reflectance Model

⚠: Suitable Material: soft cloth, concrete, matte paints

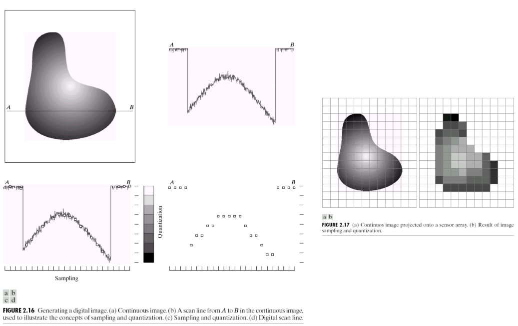


We know light can specular (镜面反射) or Diffuse (漫反射), in Lambertian we suppose purely diffuse:

$$I = I_{source} \cdot \rho\,(\text{albedo}) \cdot (\vec{N}\cdot\vec{L}), \quad \vec{N}: \text{normal vector},$$

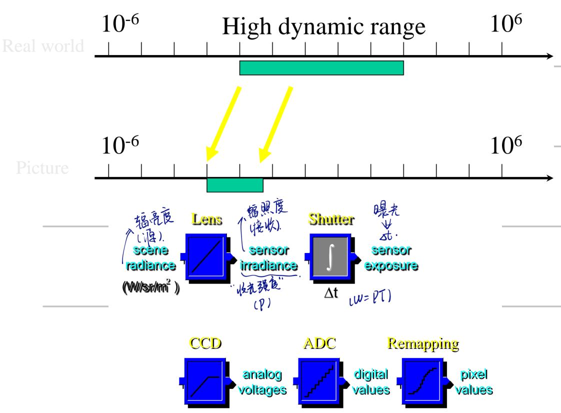
$\vec{L}$: light direction vector, $\|\vec{N}\|_2 = \|\vec{L}\|_2 = 1$ $\cos\theta$

Sampling and Quantization



有了Intensity后,用pixel来sampling.因为real world是连续的.但image, i.e., pixel space, 是离散的.

⇒ In fact, images can be regarded as signals! So sampling frequency (resolution) is important. We will talk about it later.



After getting intensity, have to map it to range [0, 255].

And we are interested about how we can enhance it, like 'white balancing'.

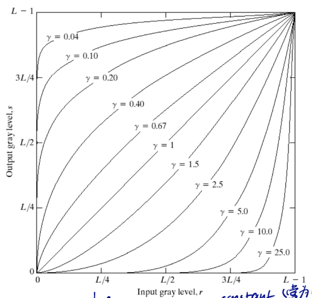


FIGURE 3.6 Plots of the equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases).

$s = cr^{\gamma}$

output pixel value $\in [0,1]$; constant (常列); gamma, 控制它就能控制亮度; Input pixel value $\in [0,1]$

Method 1: Power-law transformations.

$s = cr^{\gamma}$, $r \in [0,1]$, $s \in [0,1]$

Method 2: Negative : $s = 1 - r$

(Mostly Applied in Medical)

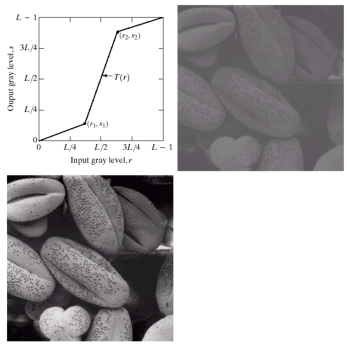


FIGURE 3.10 Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

Method 3: Contrast Stretching

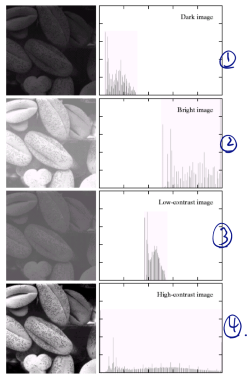
Method 4: Log : $s = c \cdot \log(1+r)$

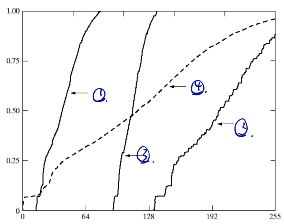
Method 5: Image Histogram

$S = 255 \times CDF(r)$,

$r \in [0, 255]$, $r \in \mathbb{Z}$

$CDF(r) \in [0,1]$.





s = T(r)

FIGURE 3.15 Four basic image types: dark, light, low contrast, high contrast, and their corresponding histograms.

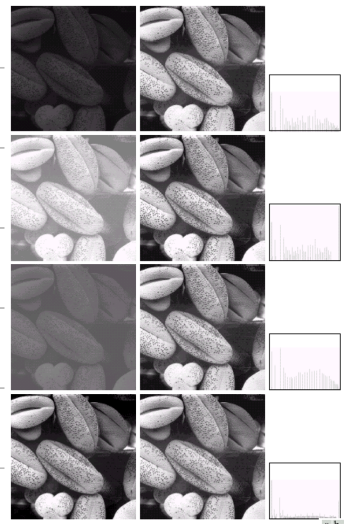
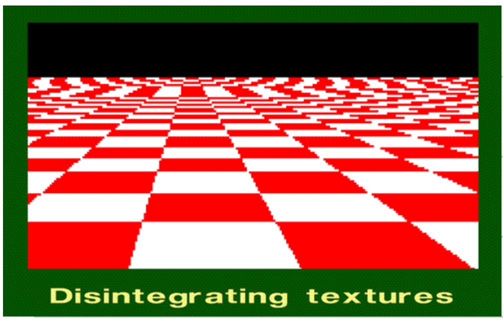


FIGURE 3.17

Now, back to sampling stuff. According to Nyquist Theorem



$f_s > 2 \cdot f_{max}$

If failed:

Aliasing !!

How to ease aliasing ?

Sample more often

- Join the Mega-Pixel craze of the photo industry
- But this can't go on forever

⇒ We can get rid of high freq!

Make the signal less "wiggly"

- Get rid of some high frequencies*
- Will loose information
- But it's better than aliasing

缺部分 info.

- Transformations on signals; e.g.:
  - bass/treble controls on stereo
  - blurring/sharpening operations in image editing
  - smoothing/noise reduction in tracking
- Key properties
  - linearity: filter($f + g$) = filter(f) + filter(g)
  - shift invariance: behavior invariant to shifting the input
    - delaying an audio signal
    - sliding an image around
- Can be modeled mathematically by *convolution*

We can use Convolution as filter to image, like a box filter:

$h[\cdot,\cdot]$

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

Cross-correlation

In Convolution, the operation sequence on entry is reversed (Actually in DL, we don't flip)

Let F be the image, H be the kernel (of size 2k+1 x 2k+1), and G be the output image

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

This is called a **cross-correlation** operation:

$$G = H \otimes F$$

- Can think of as a "dot product" between local neighborhood and kernel for each pixel
  = convolution if kernel is symmetric

Lec4: Convolution and Derivatives.

Cross-correlation vs. Convolution

cross-correlation: $G = H \otimes F$ $\quad H \otimes F$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

(symmetric)

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

It is written:

$$G = H \star F \qquad H * F.$$

Notation: $b = c \star a$

Convolution is a multiplication-like operation

- commutative $a \star b = b \star a$
- associative $a \star (b \star c) = (a \star b) \star c$
- distributes over addition $a \star (b + c) = a \star b + a \star c$
- scalars factor out $\alpha a \star b = a \star \alpha b = \alpha (a \star b)$
- identity: unit impulse e = […, 0, 0, 1, 0, 0, …]

  $a \star e = a$

Conceptually no distinction between filter and signal

Usefulness of associativity

- often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
- this is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

A Gaussian kernel gives less weight to pixels further from the center of the window

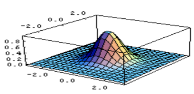
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x,y]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$H[u,v]$

$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



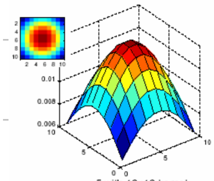
This kernel is an approximation of a Gaussian function:

Slide by Steve Seitz

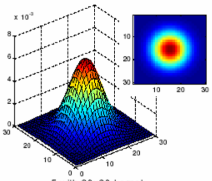
31

Instead of box filter, we can use Gaussion to blur!

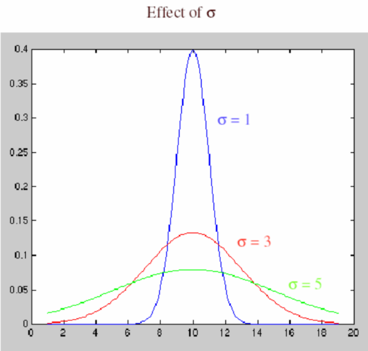
The Gaussian function has infinite support, but discrete filters use finite kernels



σ = 5 with 10x10 kernel

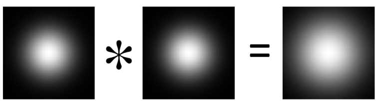


σ = 5 with 30x30 kernel



Removes "high-frequency" components from the image (low-pass filter)

Convolution with self is another Gaussian



Convolving twice with Gaussian kernel of width σ = convolving once with kernel of width $\sigma\sqrt{2}$

Consider: If we want to reduce the size of an image, we will see aliasing! So we can blur it with Gaussian first, then down sample to reduce size.

Gaussian (lowpass) pre-filtering

欲良好的 downsampling ⇒ 先 gaussian 一遍

Gaussian 1/2 G 1/4 G 1/8

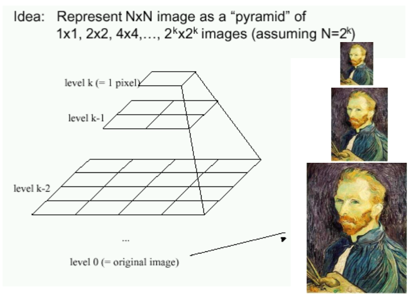
Solution: filter the image, *then* subsample

- Filter size should double for each ½ size reduction. Why?

Slide by Steve Seitz

Image Pyramids

This forms an Image Pyramid



Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]

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

Image is function f(x,y)

Remember: $\dfrac{\partial f(x,y)}{\partial x} = \lim\limits_{\epsilon \to 0} \dfrac{f(x+\epsilon,y) - f(x,y)}{\epsilon}$

Approximate: $\dfrac{\partial f(x,y)}{\partial x} \approx \dfrac{f(x+1,y) - f(x,y)}{1}$

-1	1

Another one: $\dfrac{\partial f(x,y)}{\partial x} \approx \dfrac{f(x+1,y) - f(x-1,y)}{2}$

-1	0	1

$\dfrac{\partial f(x,y)}{\partial x}$

-1	1

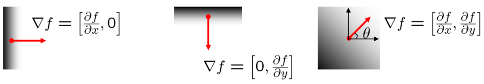
$\dfrac{\partial f(x,y)}{\partial y}$

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

Which shows changes with respect to x?

Convolution can also help to find the gradient of picture.

The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



The gradient points in the direction of most rapid increase in intensity

- How does this direction relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

The gradient direction is given by $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

Source: Steve Seitz

This information can be very useful

Also we have:

$$\frac{\partial}{\partial x}(h * f) = \left(\frac{\partial h}{\partial x}\right) * f$$

Finally: Image − Smoothed = Details

$\therefore$ Image + $\alpha \cdot$ Details = Sharpened

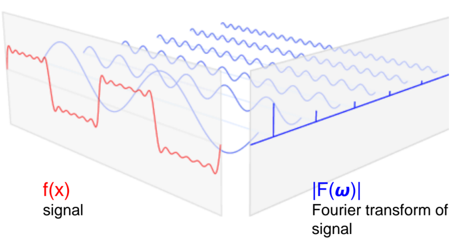
Correspond to kernel: $f + \alpha(f - f * g) = (1+\alpha)f - \alpha f * g = f * ((1+\alpha)e - \alpha g)$

image, blurred image, unit impulse (identity)

g: gaussian

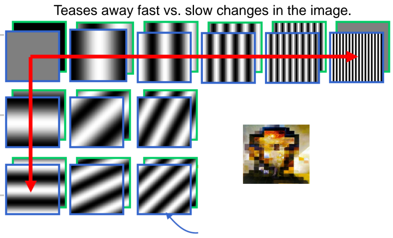
And we define: Unit impulse − Gaussian ≈ Laplacian of Gaussian

Lec5: The frequency domain



A nice set of basis

Teases away fast vs. slow changes in the image.



This change of basis has a special name…

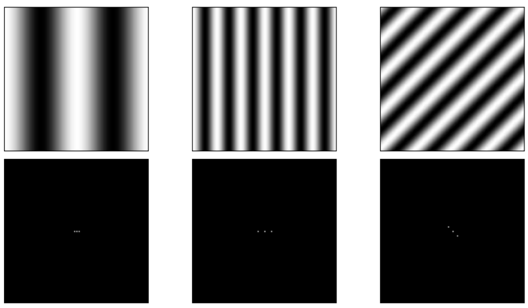
Fourier transformation: 用不同频率基信号组装出原信号。2D image也可以!

Fourier Transform : $F(\omega) = \displaystyle\int_{-\infty}^{+\infty} f(x) e^{-i\omega x} dx$

Inverse Fourier Transform : $f(x) = \dfrac{1}{2\pi} \displaystyle\int_{-\infty}^{+\infty} F(\omega) e^{i\omega x} d\omega$

Intensity Image

Fourier Image



- The Fourier transform of the convolution of two functions is the product of their Fourier transforms

$$F[g * h] = F[g]F[h]$$

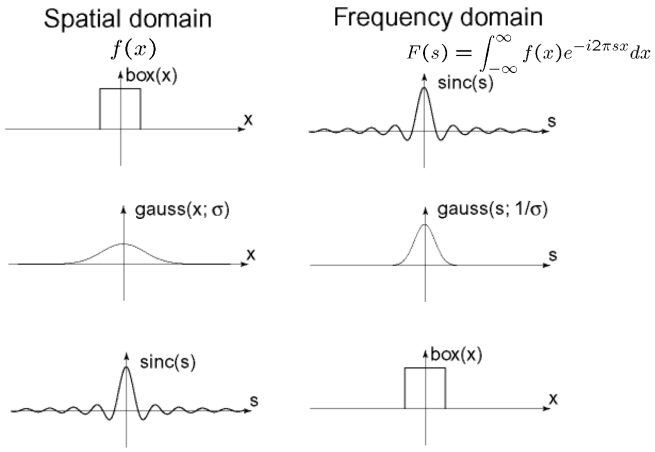
- The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms

$$F^{-1}[gh] = F^{-1}[g] * F^{-1}[h]$$

- **Convolution** in spatial domain is equivalent to **multiplication** in frequency domain!

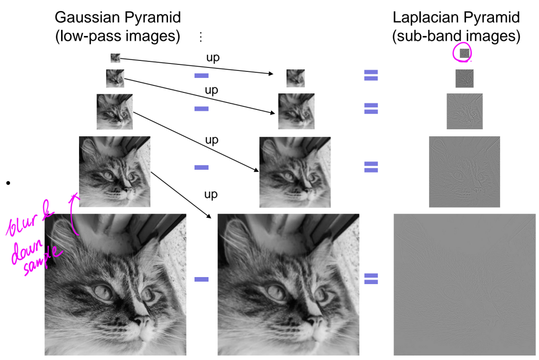
读懂 frequency domain 图 —— Spectrum
中心点 (DC分量): 图正中心，代表图像平均亮度
离中心距离: 越近，代表频率越低
点亮度: 代表该频率在原始图中能量
纹理: Spectrum 中水平亮线，对应 figure 中竖直方向边缘。竖直，对应水平边缘

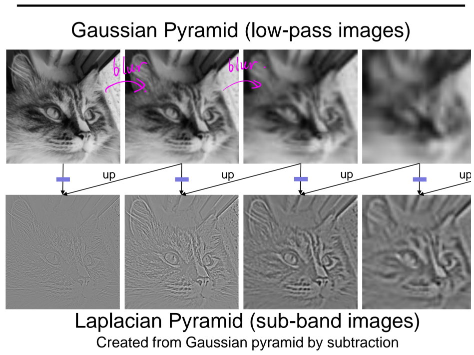
Why does the Gaussian give a nice smooth image, but the square filter give edgy artifacts?

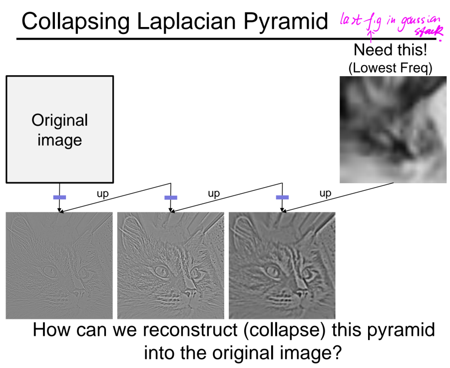


This can be used to explain this ↑
Box-filter Spatial Domain 上规整，但 frequency domain 上不好，故卷积时 Frequency domain 乘法 (点乘)
会有很多 artifact ; Gaussian 反之 ; 甚至有 sinc(s) 这种低通滤波

Lec6 Pyramid Blending, Templates, NL Filters
下面将介绍如何完成上述三个任务 : ① Blending







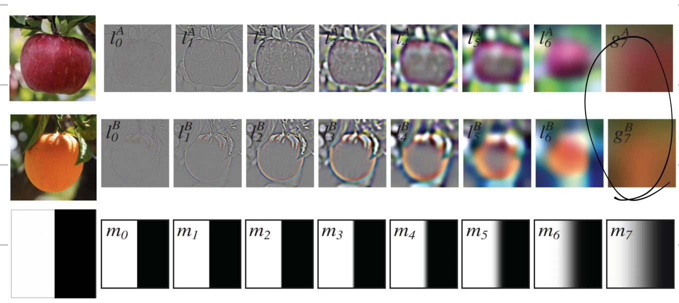
Intro to how Gaussian & Laplacian Pyramid & Stack are built.

Then: Pipeline:

Image Blending with the Laplacian Pyramid

Build Laplacian pyramid for both images: LA, LB
Build Gaussian pyramid for mask: G
Build a combined Laplacian pyramid L
Collapse L to obtain the blended image

Note: Laplacian Stack Collapse 把所有的加起来即可 (迭代)
而 Pyramid 中还需要不断地:
$I_i = L_i + \text{Upsample}(I_{i+1})$,
("滚雪球"式地重建回原始图像)



$$l_k = l_k^A * m_k + l_i^B * (1 - m_k)$$

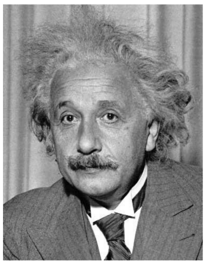
② Template:

Template matching

Goal: find in image

Main challenge: What is a good similarity or distance measure between two patches?

- Correlation
- Zero-mean correlation
- L2 distance
- Normalized Cross Correlation



Goal: find in image

f = image
g = filter

Method 1: filter the image with zero-mean eye

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

mean of g

Input · Filtered Image (scaled) · Thresholded Image

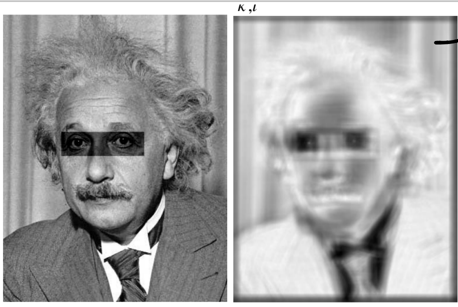
($-\bar{g}$)是为亮度不自增长 (kernel sum = 0)

Goal: find in image

Method 2: L2 distance (sum of squared diffs)

$$h[m,n]=\sum_{k,l}(g[k,l]-f[m+k,n+l])^2$$

Input · 1- sqrt(SSD) · Thresholded Image



Input · 1- sqrt(SSD)

→致命缺点: intensity sensitive

⇒ Inspire correlation

Good!

Goal: find in image

Method 3: Normalized cross-correlation

mean template · mean image patch

$$h[m,n]=\frac{\sum_{k,l}(g[k,l]-\bar{g})(f[m+k,n+l]-\bar{f}_{m,n})}{\left(\sum_{k,l}(g[k,l]-\bar{g})^2\sum_{k,l}(f[m+k,n+l]-\bar{f}_{m,n})^2\right)^{0.5}}$$

Input · Normalized X-Correlation · Thresholded Image

Zero-mean filter: fastest but not a great matcher

L2/SSD: next fastest, sensitive to overall intensity

Normalized cross-correlation: slowest, invariant to local average intensity and contrast

Summary to different methods.

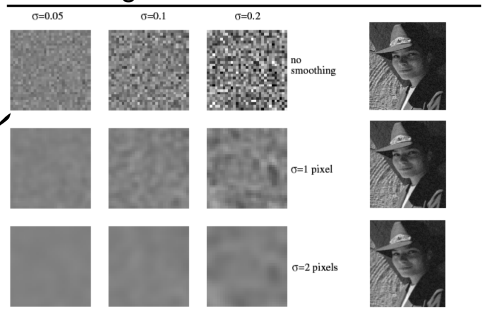
③ Denoise.

Gaussian can mitigate, but can't eliminate.

Also it can blur image at the same time!

⇒ Solution: "median" filter:

Reducing Gaussian noise

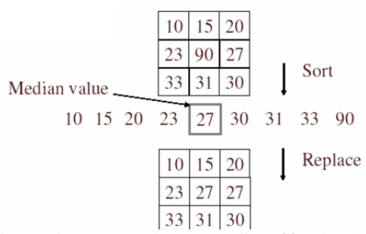


Smoothing with larger standard deviations suppresses noise, but also blurs the image

⇓ It's non-linear!! (NL)

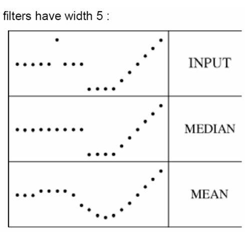
Alternative idea: Median filtering

A **median filter** operates over a window by selecting the median intensity in the window



What advantage does median filtering have over Gaussian filtering?

- Robustness to outliers



Median vs. Gaussian filtering

(复杂度高)

Take tons of time, not linear but good!!

Lec 7. Image Transformation + Warp

Previous image filtering center on change range of image.
now warp focus on change domain of image

What a 2×2 matrix can do ···

2-D Rotation

This is easy to capture in matrix form: 左旋θ度

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

Is this a linear transformation? Yes

Even though sin(θ) and cos(θ) are nonlinear functions of θ,

- ***x' is a linear combination of x and y***
- ***y' is a linear combination of x and y***

What is the inverse transformation?

- Rotation by –θ
- For rotation matrices $\mathbf{R}^{-1} = \mathbf{R}^T$

☆ (orthogonal matrix)

2D Rotate around (0,0)?

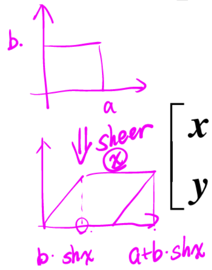
$$x' = \cos\Theta * x - \sin\Theta * y$$
$$y' = \sin\Theta * x + \cos\Theta * y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Shear?

$$x' = x + sh_x * y$$
$$y' = sh_y * x + y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Linear transformations are combinations of …

- Scale,
- Rotation,
- Shear, and
- Mirror/Reflection

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Properties of linear transformations:

- Origin maps to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Only linear 2D transformations can be represented with a 2x2 matrix

But can't bring in Translation
⇒ 3×3 ? ⇒ with additional 1
if last row: 0 0 1
Then it is affine transformation
⇐

Affine transformations are combinations of …

- Linear transformations, and
- Translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Properties of affine transformations:

- Origin does not necessarily map to origin (translation!) exactly the def of affine transformation
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition
- Models change of basis

Will the last coordinate *w* always be 1? Yes!

But if it's not:
its perspective Transformations!

Projective transformations …

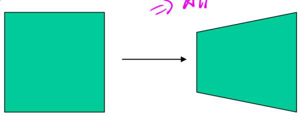
- Affine transformations, and
- Projective warps

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

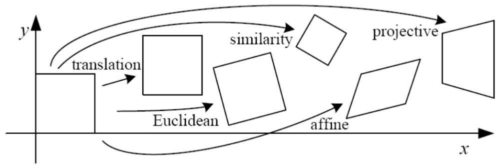
Properties of projective transformations:

- Origin does not necessarily map to origin have more freedom to allow perspectives
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis

⇒ Affine can't do this.

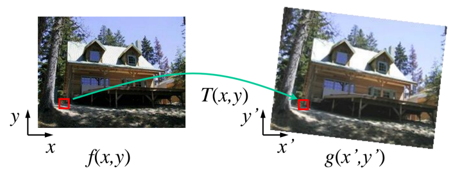


2D image transformations



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2		
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3		
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4		
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6		
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines only	

So how to warp? First: f(source) = target ---



Given a coordinate transform $(x',y') = T(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = f(T(x,y))$?

Send each pixel $f(x,y)$ to its corresponding location $(x',y') = T(x,y)$ in the second image

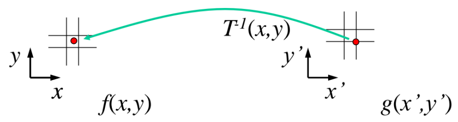
Q: what if pixel lands "between" two pixels?

A: distribute color among neighboring pixels (x',y')

– Known as "splatting" (Check out `griddata` in Matlab)

– Generally, a very bad idea. Why? loss of detail

Maybe we can try: f^{-1}(target) = ? source ---



Get each pixel $g(x',y')$ from its corresponding location $(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

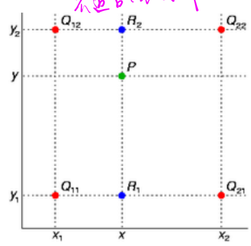
A: *Interpolate* color value from neighbors

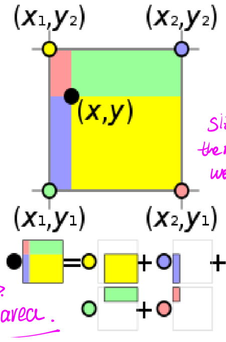
– nearest neighbor, bilinear, Gaussian, bicubic

– Check out `interp2` in Matlab / Python

Bilinear Interpolation

易记算，更证明splatting 不适合、不好算





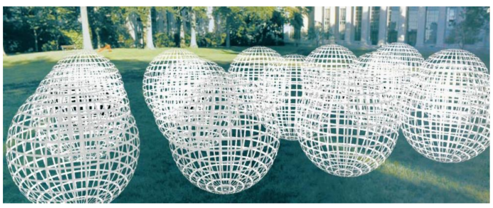
size is the natural weight

normalized area.

Use bilinear Interpolation for deciding numerical value.

Lec 8 Mosaic

The Plenoptic Function



$P(\theta,\phi,\lambda,t,V_X,V_Y,V_Z)$

- Can reconstruct every possible view, at every moment, from every position, at every wavelength
- Contains every photograph, every movie, everything that anyone has ever seen! it completely captures our visual reality! Not bad for a function…

Plenoptic Function (全光函数)描述了从一个特定观察点，在任意时间和方向上能看到的任意波长光的强度

Let's think our vision plane is a ball, receiving light at every angle ⇒ Spherical Panorama!

以该视角审视照片image的形成，深度信息都可抛去。*

Spherical Panorama

All light rays through a point form a ponorama

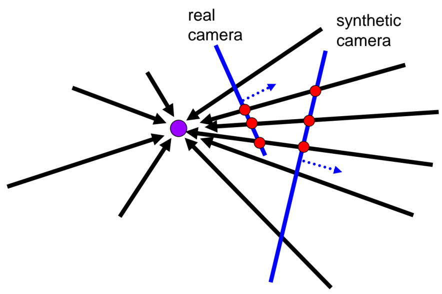
Totally captured in a 2D array -- $P(\theta,\phi)$

Where is the geometry???

As long as COP don't move, we can synthesize any camera view whose COP is identical

*（深度信息需要多角度视角以获取）

A pencil of rays contains all views



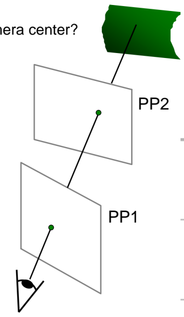
Can generate any synthetic camera view as long as it has **the same center of projection**!

How to relate two images with the same COP? Find corresponding points and warp!

Image reprojection

Basic question
- How to relate two images from the same camera center?
  - how to map a pixel from PP1 to PP2

Answer
- Cast a ray through each pixel in PP1
- Draw the pixel where that ray intersects PP2

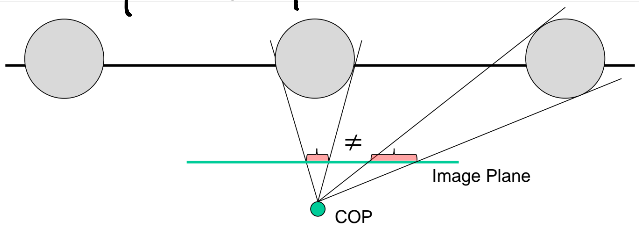


But don't we need to know the geometry of the two planes in respect to the eye?

Observation:
Rather than thinking of this as a 3D reprojection, think of it as a 2D **image warp** from one image to another

⇒ Homography

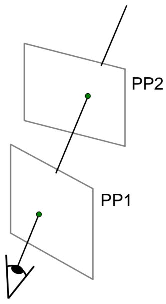
Another topic: Perspective Distortion:



Less noticeable with long focal length (i.e. you see distortion more with wide-angle camera)

A: Projective – mapping between any two PPs with the same center of projection
- rectangle should map to arbitrary quadrilateral
- parallel lines aren't
- but must preserve straight lines
- same as: unproject, rotate, reproject

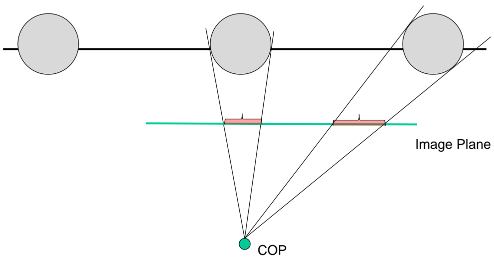
called Homography



$$\underbrace{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}}_{\mathbf{p'}} = \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p}}$$

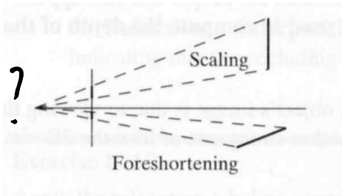
To apply a homography **H**
- Compute **p' = Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates

Scale & Foreshortening



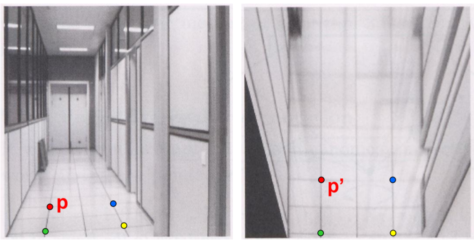
- When a line (or surface) is parallel to the image plane, the effect of perspective projection is *scaling*.

- When an line (or surface) is not parallel to the image plane, we use the term *foreshortening* to describe the projective distortion (i.e., the dimension parallel to the optical axis is compressed relative to the frontal dimension).



At last: How to work out Homography Matrix?

Image rectification



- Need Point Pairs
- Find Optimal matrix.

H's DOF = 8

p' = Hp

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Can set scale factor i=1. So, there are 8 unkowns.
Set up a system of linear equations:

Ah = b

where vector of unknowns h = $[a,b,c,d,e,f,g,h]^T$
Need at least 8 eqs, but the more the better…
Solve for h. If overconstrained, solve using least-squares:

$$\min \|Ah - b\|^2$$

How to Compute? Suppose $h_{33} = 1$:

For one point: $(x_p, y_p, 1) \xrightarrow{H} (x, y, 1)$

$$A = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x \cdot x_p & -y \cdot x_p \\ 0 & 0 & 0 & x & y & 1 & -x \cdot y_p & -y \cdot y_p \end{bmatrix} \quad h = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ \vdots \\ h_{31} \\ h_{32} \end{bmatrix} \quad b = \begin{bmatrix} x_p \\ y_p \end{bmatrix}$$

可见 1 point ⇒ 2 data ∴ 3 points fully determine

≥4 points overconstrained: $\min_H \|AH - b\|_2^2$

$H^* = (A^TA)^{-1}A^Tb$ (close form solution)

Lec 9 & 10: How to auto-find matching points?

Brute way — H has DOF of 8.

Iterate through them and find best H.

But:

Not realistic
- Search in O(N⁸) is problematic
- Not clear how to set starting/stopping value and step

What can we do?
- Use pyramid search to limit starting/stopping/step values

Alternative: gradient decent on the error function
- i.e. how do I tweak my current estimate to make the SSD error go down?
- Can do sub-pixel accuracy
- BIG assumption?
  - Images are already almost aligned (<2 pixels difference!)
  - Can improve with pyramid
- Same tool as in **motion estimation**

Can be improved with pyramid & GD

Suitable way: Find feature points, and then match. How?

Feature-based alignment

1. **Feature Detection**: find a few important features (aka Interest Points) in each image separately
2. **Feature Matching**: match them across two images
3. **Compute image transformation**: as per Project 4, P

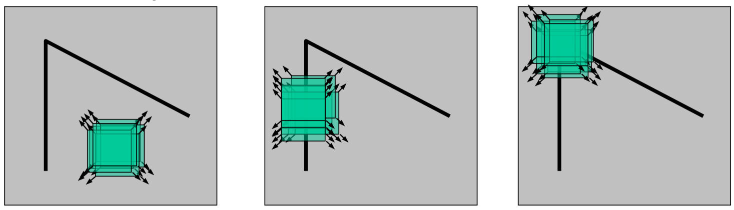
How do we match the features between the images?
- Need a way to <u>describe</u> a region around each feature
  - e.g. image patch around each feature
- Use successful matches to estimate homography
  - Need to do something to get rid of outliers

Issues:
- What if the image patches for several interest points look similar?
  - Make patch size bigger
- What if the image patches for the same feature look different due to scale, rotation, etc.
  - Need an invariant descriptor

We should easily recognize the point by looking through a small window

Shifting a window in *any direction* should give *a large change* in intensity

① Detecting "Features": Harris Corner



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

If u, v are small (small shift:)

- First-order Taylor approximation for small motions [u, v]:

$$I(x+u, y+v) = I(x,y) + I_x u + I_y v + \text{higher order terms}$$
$$\approx I(x,y) + I_x u + I_y v$$
$$= I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

plug in

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$
$$\approx \sum_{(x,y)\in W} [I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)]^2$$
$$= \sum_{(x,y)\in W} \left(\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \right)^2$$
$$= \sum_{(x,y)\in W} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

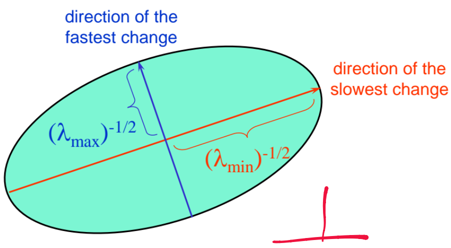
M

Consider a horizontal "slice" of $E(u, v)$: $\begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$

This is the equation of an ellipse.

Diagonalization of M: $M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$

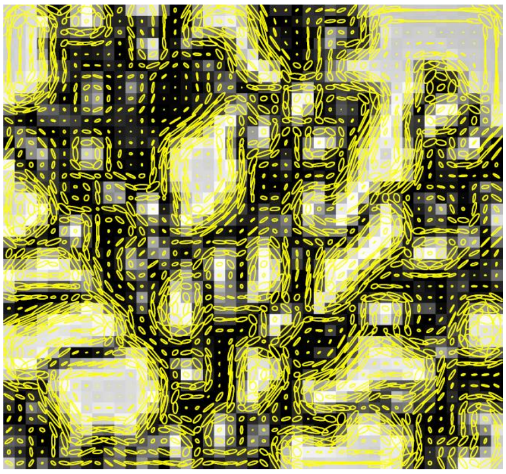
The axis lengths of the ellipse are determined by the eigenvalues and the orientation is determined by R



$\frac{1}{\sqrt{\lambda}}$

Diagonalize M & get eigenvectors.

Can use ellipse to visualize!



⇒ Gradient ↑, axis ↓

⇒ Good to reflect "change intensity"

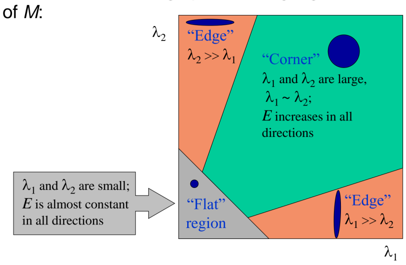
Measure of corner response:

$$R = \frac{\det M}{\operatorname{Trace} M}$$

$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

Interpreting the eigenvalues

Classification of image points using eigenvalues of M:



$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

Harris detector: Steps

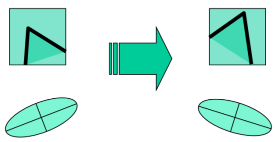
1. Compute Gaussian derivatives at each pixel
2. Compute second moment matrix M in a Gaussian window around each pixel
3. Compute corner response function R
4. Threshold R
5. Find local maxima of response function (nonmaximum suppression)

Pipeline shown on the right:

① Compute each point's I_x, I_y

② For each $(\hat{x}, \hat{y})$, draw window W. Use results in ① to compute M

Some property of harris detector:



Ellipse rotates but its shape (i.e. eigenvalues) remains the same

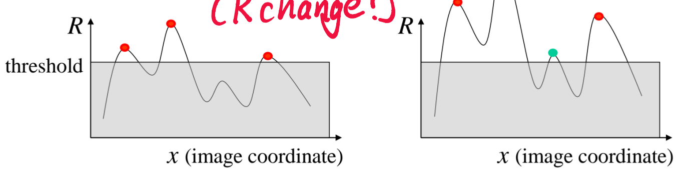
Corner response R is invariant to image rotation

✓ Only derivatives are used => invariance to intensity shift $I \to I + b$

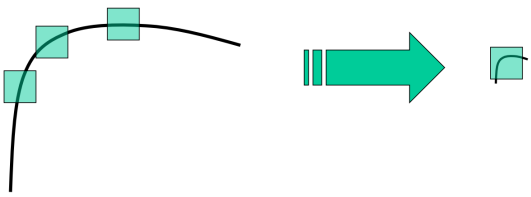
✓ Intensity scale: $I \to a\,I$

(R change!)

⇒ Partial invariance to affine intensity change.



But: non-invariant to *image scale*!

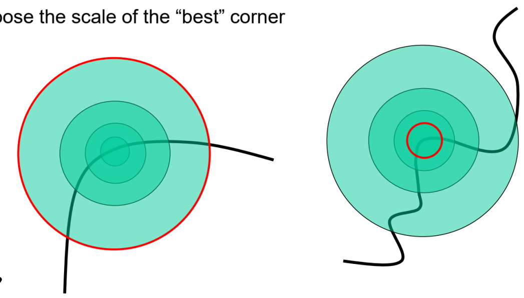


All points will be classified as edges

Corner !

The problem: how do we choose corresponding circles ***independently*** in each image?

Choose the scale of the "best" corner



⇒ 在所有可能尺度中，使角点响应函数R达到局部最大的尺度

Also, we want points are equally distributed! ⇒ ANMS.

For point set P, for $p_i \in P$, compute r_i:

$$r_i = \min_{p_j \in P,\ R_j > c \cdot R_i} \| P_j - P_i \|_2$$

, then sort using r_i and take top-p points, ② Match Harris Corners ⇒ Invariant & Distinctive

Solution: (In project) Take 40×40 patch → Blur

→ Down sample to 8×8 → Normalize: $z = \frac{x-\mu}{\sigma}$

Then to match them: Similarity between two descriptors: Euclidean

And want: $\frac{1-NN}{2-NN} < \varepsilon$, to ensure match is correct

Finally, to get rid of outlier ...

RANSAC Algorithm

*: Use all inliers! Not its original H ......

RANSAC loop:

1. Select four feature pairs (at random)
2. Compute homography H (exact)
3. Compute *inliers* where $dist(p_i{}', \boldsymbol{H}\, p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers

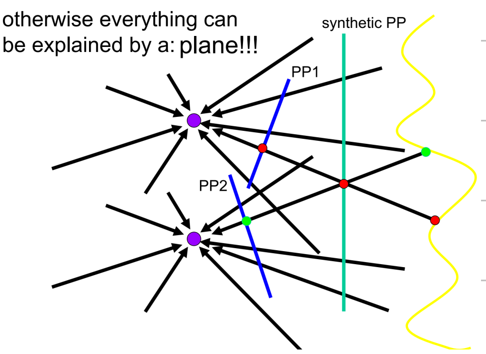
Lec11 & 12: 3D Intro & Stereo

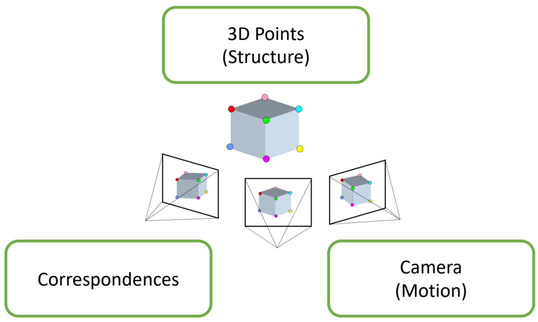
- is 3D = depth from a single image?
- 2.5D = per-pixel depth from a single image

Structure and depth are inherently ambiguous from single views.

Need two different camera center

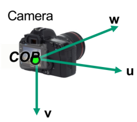
otherwise everything can be explained by a: plane!!!





There is a world coordinate frame and camera looking at the world

How can we model the geometry of a camera?



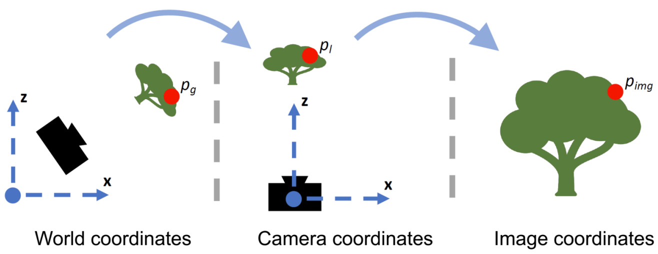
"The World"

Three important coordinate systems:

1. *World* coordinates
2. *Camera* coordinates
3. *Image* coordinates

To go from pixels to 3D location in the **world coordinates**, we need to know two things about the camera:

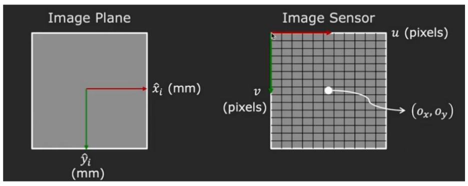
1. Position & Orientation of the camera with respect to the world (extrinsics)
2. How the camera maps a point in the world to image (intrinsics)



Want:

Image Coordinates: $\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$ ⇐ Perspective Projection (3D to 2D) ⇐ Camera Coordinates: $\mathbf{X}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$ ⇐ Coordinate Transformation (3D to 3D) ⇐ World Coordinates: $\mathbf{X}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$

Image Plane to Image Sensor Mapping



1. Account for pixel density (pixel/mm) & aspect ratio by scalars: $[m_x, m_y]$

$$m_x x_i, m_y y_i$$

2. Usually the top left corner is the origin. But in the image plane, the origin is where the optical axis pierces the plane! Need to shift by: (o_x, o_y)

$$u_i = \alpha_x x_i + o_x = \alpha_x f \frac{x_c}{z_c} + o_x$$

where $[f_x, f_y] = [m_x f, m_y f]$

Pixel Coordinates:

$$u_i = \boxed{f_x}\frac{x_c}{z_c} + \boxed{o_x} \quad v_i = \boxed{f_y}\frac{y_c}{z_c} + \boxed{o_y}$$

m_x, m_y: 每 mm 多少 pixel

Image plane: origin : center !

Image Sensor : Top-Left point !

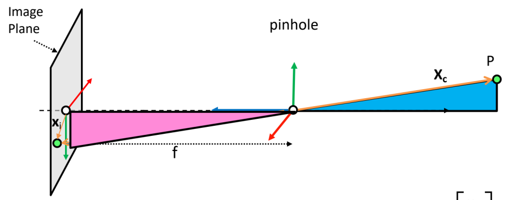
Intrinsic (with homogeneous)

Perspective projection + Transformation to Pixel Coordinates:

$$u_i = f_x \frac{x_c}{z_c} + o_x \quad v_i = f_y \frac{y_c}{z_c} + o_y$$

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

Intrinsic Matrix



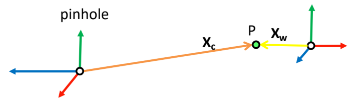
$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \qquad \frac{x_i}{f} = \frac{x_c}{z_c} \qquad \mathbf{X}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

Image Coordinates — Camera Coordinates

$$x_i = f\frac{x_c}{z_c}$$

Extrinsic (need homogeneous for translation)

Camera Transformation (3D-to-3D)



Camera Coordinates — World Coordinates

$$\mathbf{X}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \qquad \mathbf{X}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

Coordinate Transformation

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R_{3\times 3} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Extrinsic

Putting it all together

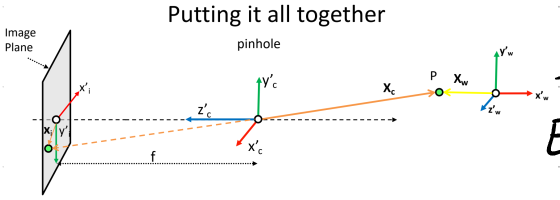


Image Coordinates — Camera Coordinates — World Coordinates

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \mathbf{X}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad \mathbf{X}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

Perspective Projection — Coordinate Transformation

$$M = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} R_{3\times 3} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}$$

M's DOF:

5 + 6 = 11

Intrinsic (4+1)

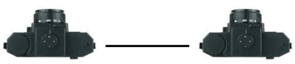
Extrinsic (6)

*: Skew: Some camera's pixel is parallelogram. Need skew.

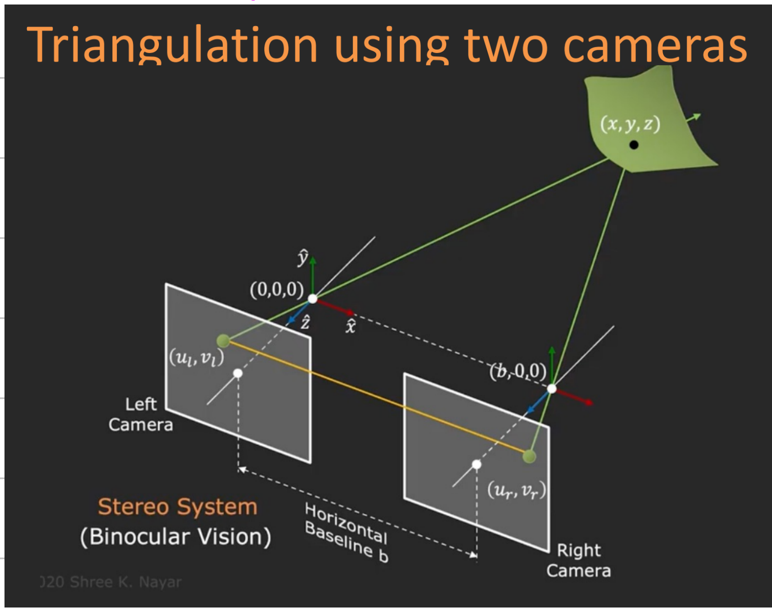
Now we can use multi-view to sense depth.

From now, suppose cameras are calibrated (In/Extrinsic Matrix Unknown)

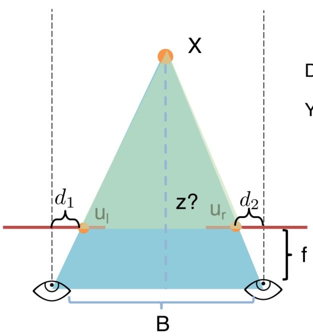
- Assume **parallel** optical axes
- Two cameras are calibrated
- Find relative depth



Key Idea: difference in corresponding points to understand shape



Solving for Depth in Simple Stereo



Do we have enough to know what is Z?

Yes, similar triangles!

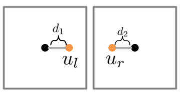
$$\frac{B - (u_l - u_r)}{z - f} = \frac{B}{z}$$

$$z = \frac{fB}{u_l - u_r}$$

disparity (how much corrsp. pixels move)

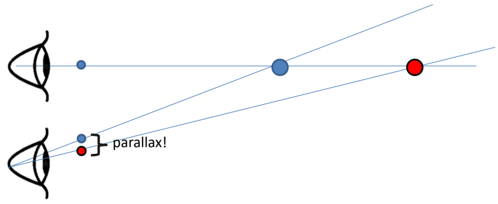
Base of : $B - (d1 + d2)$

in image coordinates: $= B - (u_l - u_r)$



$$Z\text{(depth)} \propto \frac{1}{u_l - u_r}$$ ← disparity

Parallax



Parallax = *from ancient Greek parállaxis*
= *Para* (side by side) + *allássō*, (to alter)
= *Change in position from different view point*

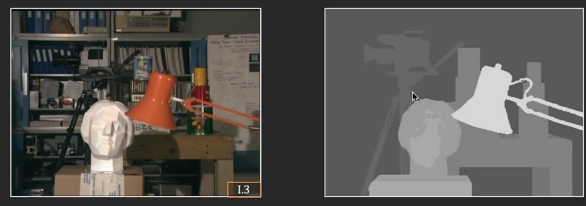
Two eyes give you parallax, you can also move to see more parallax = "Motion Parallax"

视差：disparity caused due to two eyes.

Consider task:

Stereo Matching: Finding Disparities

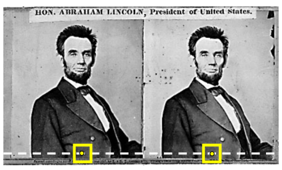
Goal: Find the disparity between left and right stereo pairs.



Left/Right Camera Images | Disparity Map (Ground Truth)

So at least we want to find pairs of points who has same disparity:

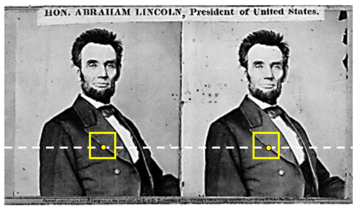
Your basic stereo algorithm



For every epipolar line:
- For each pixel in the left image
  - compare with every pixel on same epipolar line in right image
  - pick pixel with minimum match cost

Improvement: match ***windows***, *+ clearly lots of matching strategies*

Dense correspondence search



For each epipolar line
- For each pixel / window in the left image
  - compare with every pixel / window on same epipolar line in right image
  - pick position with minimum match cost (e.g., SSD, correlation)

Epipolar line:
对于左边一个点，我们知道必在右图中一条特定直线上有对应点。
(parallel movements)

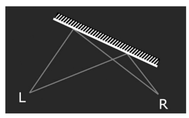
Issue: Effect of window size:

Want window large enough to have sufficient intensity variation, yet small enough to contain only pixels with about the same disparity.

Issues with Stereo

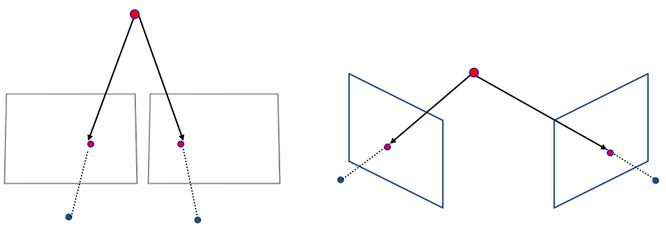
- Surface must have non-repetitive texture

- Foreshortening effect makes matching a challenge



More general case: Cameras are calibrated but doesn't have to have parallel optical axes

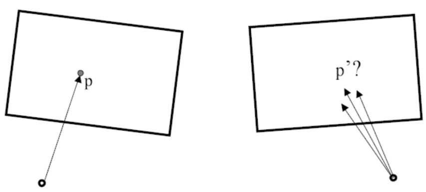
- The two cameras need not have parallel optical axes.



Option 2

1. **Solve for correspondences**
2. **Estimate camera**
   - **What is the relationship between the camera + correspondences?**
3. Triangulate

⇑
Issue:

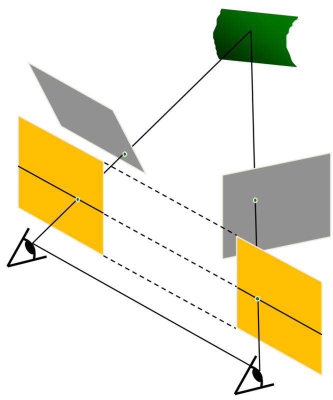


- Given p in left image, where can corresponding point p' be?

Option 1: Rectify via homography

Quite Familiar

- reproject image planes onto a common plane
  - plane parallel to the line between optical centers
- pixel motion is horizontal after this transformation
- two homographies, one for each input image reprojection
  - C. Loop and Z. Zhang. Computing Rectifying Homographies for Stereo Vision. CVPR 1999.



To be continued in future class