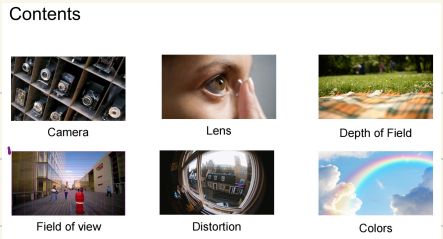
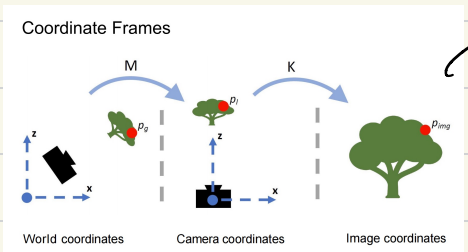
24.11.29.

01 Introduction. ✓

02 Image Formation.





→ 坐标系转换.

世界坐标系
↓ ← 外参矩阵
相机坐标系
↓ ← 内参矩阵.
图像坐标系.

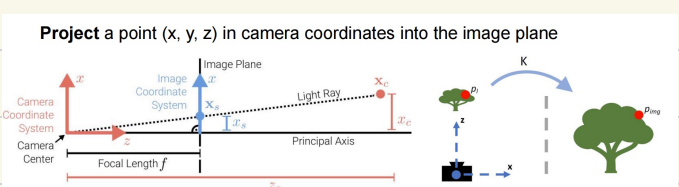
Extrinsic Camera Parameters ⇒ M.

① 世界坐标平移至相机 (Transition). ⇒ $T=\begin{bmatrix} I_{3\times3} & t_{3\times1} \\ 0 & 1 \end{bmatrix}$

② 世界坐标对齐相机方向 (Rotation). ⇒ $R=\begin{bmatrix} R_{3\times3} & 0 \\ 0 & 1 \end{bmatrix}$.

$\rightarrow M=R\cdot T=\begin{bmatrix} R_{3\times3} & t_{3\times1} \\ 0 & 1 \end{bmatrix}$

Intrinsic Camera Parameters. ⇒ K.



① 物体投影至平面 (3D→2D) ⇒ $Proj=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$.

② 平面上物到中心点后缩放, 由内参决定 ⇒ $Ins=\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$.

1: Projective
0: Orthographic.

$\rightarrow K=\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$.

$\leftarrow \frac{x_s}{f}=\frac{x_c}{z_c}$.

Summary: Camera Parameters

$$\Pi = \begin{bmatrix} f & s & c_x \\ 0 & \alpha f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{T}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}$$

identity matrix

intrinsics projection rotation translation

Extrinsic parameters including the rotation and translation
Intrinsic parameters including the focal length and the principal point coordinate

- α: aspect ratio, equals 1 (unless pixels are not square)
- s: skew, equals 0 unless pixels are shaped like parallelograms
- (c_x, c_y): principal point, equals (w/2,h/2) unless optical axis doesn't intersect the image center

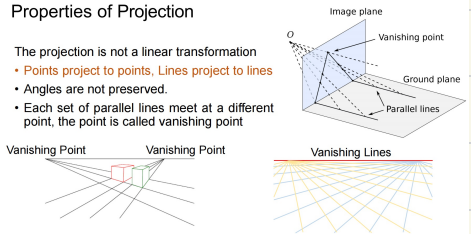
Projection Models → 投影模型.

① 正交投影, Orthographic Projection.

② 透视投影, Perspective Projection.

Properties of Projection

The projection is not a linear transformation
- Points project to points, Lines project to lines
- Angles are not preserved.
- Each set of parallel lines meet at a different point, the point is called vanishing point



投影不是线性变换.

⇒① 点投影成点, 线投影成线.

② 角度不被保证不变 (不保角).

③ 所有的平行线最终会汇于一条线上的不同点 → Vanishing Point.

→ Vanishing Line.

计算:

Figure 2 is a picture with 4096 × 3072 pixels of our campus. We can establish a right-handed world coordinate system where the ground is the $z = 0$ plane, the edge of the roadside lawn is the x-axis, and the edge below the flowerbed serves as the y-axis. All measurements are in meters (m). The height of the first flowerbed (red rectangle) is 0.76 meters, and the camera's coordinates are $(13.4, 4.5)$ in the x and y directions. Additionally, the edge point A of the hexagonal flowerbed has coordinates $(0, 3.74, 0.76)$ in the world coordinate system.

2: Camera Parameters

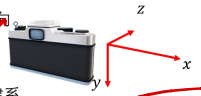
- 2.1

- 找两条平行线，两个灭点
- 根据灭点连灭线
- 灭线到坐标架的垂线高即为相机高度 ($pixelwise$)
- 参照物（花盆点A）作垂线得到高度 ($pixelwise$)
- 已知花坛真实高度 (m)，可作比例

$$\frac{380}{190} \cdot 0.76 = 1.52\ (m)$$

2: Camera Parameters

- **2.2 Camera Intrinsic**
- **建立相机坐标系：**相机朝向世界坐标系的$-x$方向。
  以世界坐标系的y方向为相机坐标系的x方向，
  以世界坐标系的$-z$方向为相机坐标系的y方向，
  以世界坐标系的$-x$方向为相机坐标系的z方向建系。



调整世界坐标系朝向.

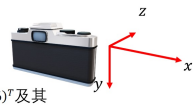
- 相机在世界坐标系的坐标是(13.4, 4.5, 1.52)。
  可以得到R、T矩阵：

$$R=\begin{pmatrix}0&1&0&0\\0&0&-1&0\\-1&0&0&0\\0&0&0&1\end{pmatrix}\quad T=\begin{pmatrix}1&0&0&-13.4\\0&1&0&-4.5\\0&0&1&-1.52\\0&0&0&1\end{pmatrix}$$

- 以图片左上角为原点，水平向右为 x 轴，
  竖直向下为 y 轴建立图像坐标系

2: Camera Parameters

- **2.2 Camera Intrinsic**
- 由图像像素值可知 $C_x = 2048, C_y = 1536$
- 内参矩阵：$\begin{pmatrix}f&0&2048\\0&f&1536\\0&0&1\end{pmatrix}$



- 根据花坛参考点A在世界坐标系坐标$(0,\ 3.74,\ 0.76)^T$及其在图像坐标系下的坐标$(1894,\ 1954)^T$

$$k\cdot\begin{pmatrix}1894\\1954\\1\end{pmatrix}=\begin{pmatrix}f_x&0&C_x\\0&f_y&C_y\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}0&1&0&0\\0&0&-1&0\\-1&0&0&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&-13.4\\0&1&0&-4.5\\0&0&1&-1.52\\0&0&0&1\end{pmatrix}\begin{pmatrix}0\\3.74\\0.76\\1\end{pmatrix}$$

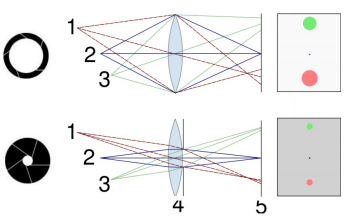
可计算f.

透视成像公式：$\frac{1}{u}+\frac{1}{v}=\frac{1}{f}$.

概念：DOF (Depth of Field) 景深

The aperture size affects DOF

- Smaller aperture → large DOF
- Small aperture reduces amount of light → need to increase exposure, more noisy images



DOF↓.

大光圈，只在小范围内清晰，景深范围较小.

小光圈，细节很清晰，景深较大，成像范围较大

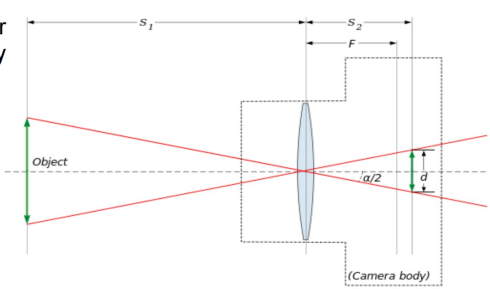
DOF↑

FOV (Field of View) 视场.

The field of view is the angular extent of the world observed by the camera.

$$\tan\left(\frac{\alpha}{2}\right)=\frac{d/2}{S_2}$$

where d is the size of the sensor and $S_2=(1+m)\,F$ (For infinity focus, $S_2=F$).



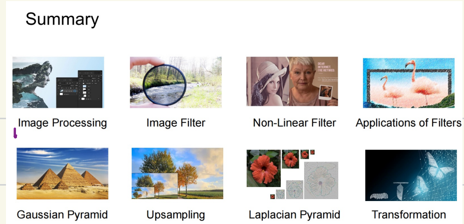
长焦：S_2↑，像角FOV.

短焦：S_2↓，广角视场FOV

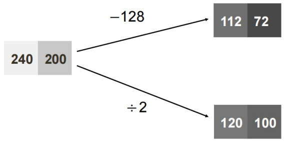
相机传感器 CCD ⇒ 景物光线，清晰较大.

CMOS ⇒ 噪声，互连输出

03 Image Processing.



Pixel Processing: **Darkening** v.s. **Contrast**



Brightness: all pixels get lighter or darker, relative difference between pixel values stays the same → 调节亮度，直接加减

Contrast: relative difference between pixel values becomes higher / lower → 调节对比度，除法（缩小/放大差距）

$y(x)=a(x)f(x)+b(x)$

$a(x)$ → contrast; $b(x)$ → brightness.

Image Filtering ⇒ 图像滤波.

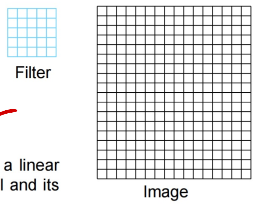
Linear Filters

$I(i,j)$: $w \times h$ digital image
$h(u,v)$: $m \times n$ "filter" or "kernel"

$$I(i,j)=\sum_{k=-p}^{p}\sum_{l=q}^{-q} I(i+k,j+l)*h(k,l)$$

Assume m and n are odd, $p=\lfloor\frac{m}{2}\rfloor, q=\lfloor\frac{n}{2}\rfloor$.

Intuition: each pixel in the output image is a linear combination of the corresponding input pixel and its neighboring pixels in the original image.



(k^2). 二维 Filter 核可以转为拆分，接到运算 $(k+k)$.

| h_1 | h_2 | h_3 | h_4 |

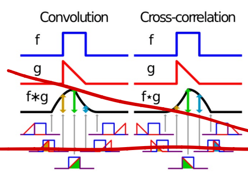
两重 ↓ 两次.

For kernel K, SVD分解 ⇒ $K=\sum_i \sigma_i u_i v_i^T$

Properties of Liner Filtering

- Correlation $g(i,j)=\sum_{k,l} f(i+k,j+l)h(k,l)$
- Convolution $g(i,j)=\sum_{k,l} f(i-k,j-l)h(k,l)$
  $=\sum_{k,l} f(k,l)h(i-k,j-l)$

→ 做计算时差一个 flip.



- Both correlation and convolution are linear shift-invariant (LSI) operators

Superposition principle: $h\circ(f_0+f_1)=h\circ f_0+h\circ f_1$

Shift invariance: $g(i,j)=f(i+k,j+l) \Leftrightarrow (h\circ g)(i,j)=(h\circ f)(i+k,j+l)$

Median Filter ⇒ 对kernel内值排序，取中间值（非线性滤波）.

可以处理椒盐噪声

Down sampling

直接 Downsampling 会导致走样 ⇒ Gaussian Pyramid.
(先 Gaussian Blur.
后 Downsampling).

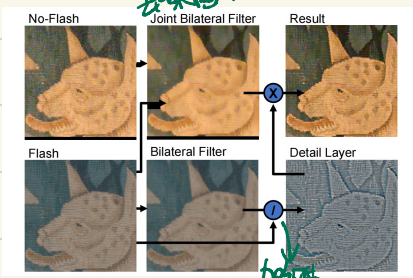
Bilateral Filter (双边滤波).

$$S(i,j)=\frac{\sum_{k,l} f(k,l)\,w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

$$w(i,j,k,l)=\exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}-\frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_r^2}\right)$$

domain kernel

Bilateral Filter.



去噪音.

加强细节

(被完美双边滤波后与原图相减).

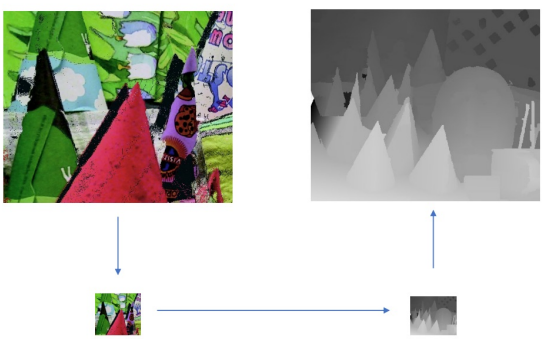
Joint Bilateral Upsampling

For significant speedup

- Downsample input image
- Estimate the depth image
- Upsample the depth image

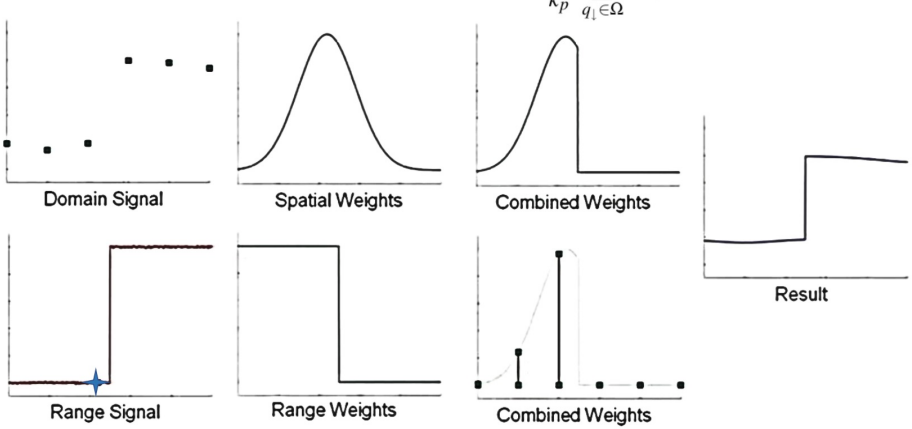
How to upsample the results?

- Reuse the edge information in the original input image

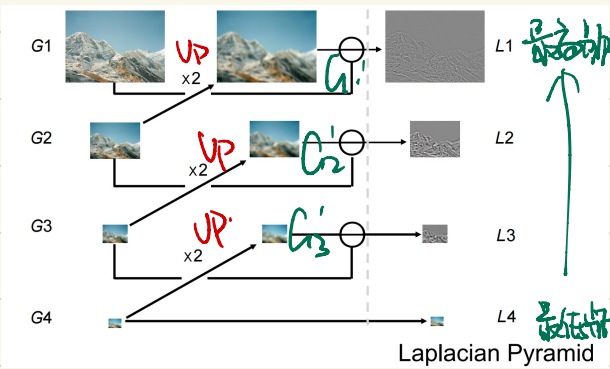


Joint Bilateral Upsampling

$$\tilde{S}_p=\frac{1}{k_p}\sum_{q_\downarrow\in\Omega} S_{q_\downarrow}\, f(\|p_\downarrow-q_\downarrow\|)\, g(\|\tilde{I}_p-\tilde{I}_q\|)$$

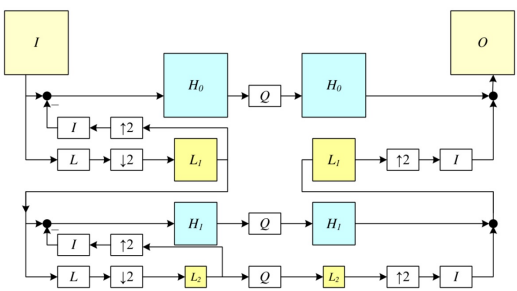


Laplacian Pyramid



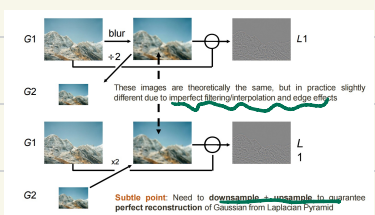
Laplacian Pyramid

Summary of Laplacian Pyramid



Gaussian pyramid
- Yellow blocks
- Downsampling
- Gaussian filter

Laplacian pyramid
- Blue blocks + L_2
- Gaussian pyramid
- Upsampling



Laplacian Pyramid

Building a Laplacian pyramid
- Create a Gaussian pyramid
- Take the difference between one Gaussian pyramid level and the next

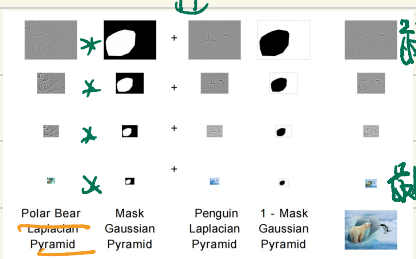
Why is it named as Laplacian Pvramid?
- Difference of Gaussian (DoG) is an approx. of Laplacian of Gaussion (LoG)
- DoG is a band-pass filter

$$G(x,y;\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

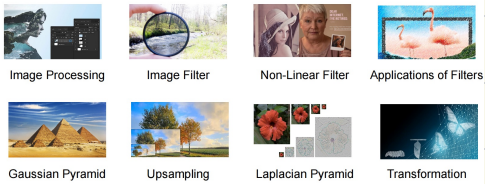
$$\nabla^2 G(x,y;\sigma) = \left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G(x,y;\sigma) = \frac{\partial G}{\partial \sigma}\frac{1}{\sigma}$$

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma}$$

$$G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2\nabla^2 G$$



Summary



Gaussian金字塔

Gaussian金字塔是一种通过对原始图像进行逐级下采样来构建的图像金字塔。每一层都是通过对上一层进行高斯模糊（低通滤波）和下采样得到的。

计算步骤：

1. **初始化**：以原始图像作为金字塔的最底层（第0层）。
2. **高斯模糊**：对当前层的图像应用高斯滤波器进行模糊处理。高斯滤波器是一种线性低通滤波器，它可以平滑图像并减少高频细节。
3. **下采样**：将模糊后的图像进行下采样，通常是删除偶数行和偶数列，使得图像的尺寸缩小为原来的一半。
4. **重复**：将下采样后的图像作为新的当前层，重复步骤2和3，直到达到所需的层数或图像尺寸变得足够小。

Laplacian金字塔

Laplacian金字塔是基于Gaussian金字塔构建的，它用于存储图像的高频信息。每一层都是通过从Gaussian金字塔中相邻两层之间计算差值得到的。

计算步骤：

1. **构建Gaussian金字塔**：首先构建一个Gaussian金字塔。
2. **上采样**：将Gaussian金字塔中的每一层（除了最顶层）进行上采样，通常是插入零值来扩大图像尺寸，然后应用高斯滤波器以消除上采样引入的混叠效应。
3. **计算差值**：对于Gaussian金字塔中的每一层（除了最顶层），计算其与上采样后相邻较高层之间的差值。这个差值就是Laplacian金字塔的当前层。
4. **重复**：重复步骤2和3，直到处理完所有层。

数学表示：

- Gaussian金字塔的第 l 层 G_l 可以通过以下方式计算：

$$G_l = \text{downsample}(\text{gaussian\_filter}(G_{l-1}))$$

  其中 G_0 是原始图像，gaussian_filter 是高斯滤波操作，downsample 是下采样操作。

- Laplacian金字塔的第 l 层 L_l 可以通过以下方式计算：

$$L_l = G_l - \text{upsample}(G_{l+1})$$

  其中 upsample 是上采样操作，G_{l+1} 是Gaussian金字塔中 G_l 的上一层。

通过这种结构，Gaussian金字塔捕获了图像的低频信息，而Laplacian金字塔则捕获了图像的高频细节。这两者结合起来可以用于图像的重建和多种图像处理任务。

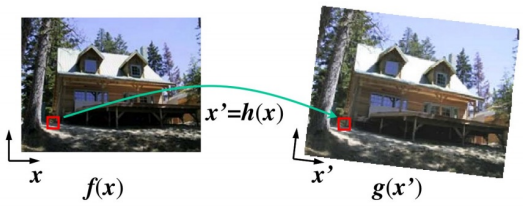
Forward Warping

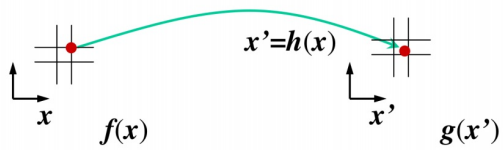
For every pixel x in f(x)

- 1. Compute the destination location x′ = h(x).
- 2. Copy the pixel f(x) to g(x′)

Limitations of forward warping

- x′ may have be non-integer
- g(x) may contain cracks and holes, especially when magnifying an image

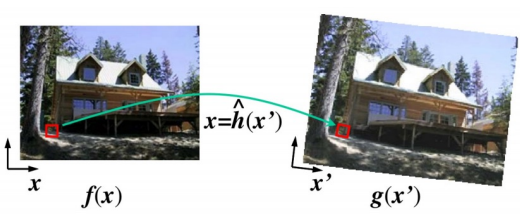




Inverse Warping

For every pixel x′ in g(x′)

- 1. Compute the source location x with $\hat{h}(x')$
- 2. Resample f(x) at location x and copy to g(x′)



Implementation details

- $\hat{h}(x)$ is the inverse of $h(x)$
- Resample with bilinear interpolation

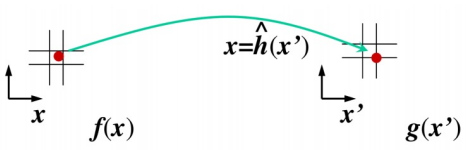
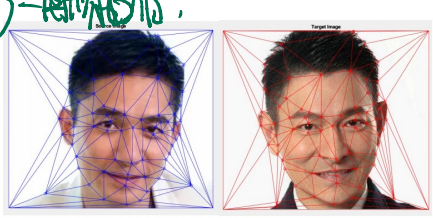


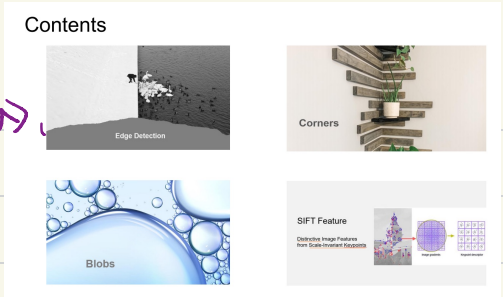
Image Morphing

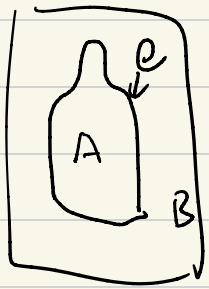
- Select feature points
- Build Delaunay triangulations
- Morph the triangulations by linear interpolate each triangle
  $$T = \mu T_s + (1-\mu)T_u$$
- Perform image warping in each triangle with the transformation in each triangle



04 Feature Detection

Edge Detection (边缘检测)





A-e-B 边缘e处满足A→e, e→B f(x)的变化.

⇒ 检测偏导数变化. 把2D Image视为 f(x,y).

↳ $\frac{\partial f(x,y)}{\partial x}$, $\frac{\partial f(x,y)}{\partial y}$.

→ $\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y)-f(x,y)}{1}$ → kernel: [-1 | 1]

or $\approx \frac{f(x+1,y)-f(x-1,y)}{2}$ kernel: [-1 | 0 | 1]

⇒ Partial Kernel

Partial Derivatives of an Image

- Partial Derivatives of an Image can be computed as image filtering

Other Differentiation Operations	Horizontal	Vertical
Prewitt	$\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix}$
Sobel	$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$

发现容易受噪声影响 ⇒ 可以先用 Gaussian Filter 再做 Partial

⇒ Canny 边缘提取.

Canny Edge Detector

1. **Preprocessing**: Grayscale conversion + Gaussion blur

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

①转灰度图 + Gaussian blur

2. Use **central differencing** to compute gradient image (instead of first forward differencing), which is more accurate.

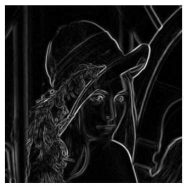
$$\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}$$

②Sobel算子.

3. **Calculate gradient magnitude**

Gx

Gy



Gradient Magnitude

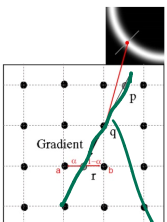
③计算 $\sqrt{\left(\frac{\partial f}{\partial x}\right)^2+\left(\frac{\partial f}{\partial y}\right)^2}$

$\theta=\arctan\left(\frac{\partial f/\partial y}{\partial f/\partial x}\right)$

Sobel Edge Detector thresholds the gradient magnitude to obtain edges. But thresholds are brittle, and the thresholds are hard to choose.

4. **Non-Maximum Suppression**

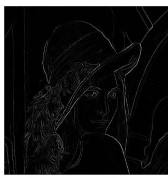
- The gradient magnitude produced results in thick edges. we perform non-maximum suppression to thin out the edges.
- Go through all the points on the gradient intensity matrix, **find and only keep** the pixels with the maximum value in the edge directions
- Example: keep q if $q > p$ and $q > r$



$r = \alpha b + (1-\alpha)a$

④在这条线上找极大值

Edges are too thick, make them thinner.



Non Maximum Suppression

④非极大值抑制

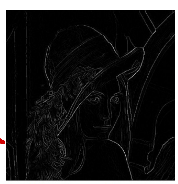
把θ分到几个方向(如8个)上，取最下的梯度强度和对应的θ；然后在一个方向上判断其是否为最大值。

↓

确保边缘为单像素宽度

5. **Double threshold**

- Pixels with a high value (e.g., 0.7) are most likely to be edges.
- Pixels with a low value (e.g., 0.3) are most likely not to be edges.
- Other pixels are defined as weak edges
- Weak edges are further classified as edge or not in the next step



Non Maximum Suppression

Double Thresholding

6. **Edge Tracking by Hysteresis**

Weak edges that are connected to strong edges will be actual/real edges.
Weak edges that are not connected to strong edges will be removed.

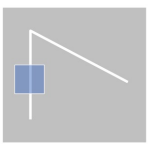
⑤双阈值处理：把边缘分成三类.

边缘追踪：与强边缘相连的弱边缘像素记为边缘.

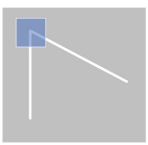
⇒Corners 角点检测

Harris Corner Detector.

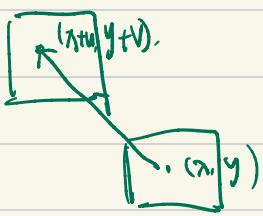
Flat region: no change in all directions



Edge: no change along the edge direction



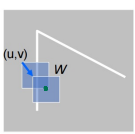
Corner: significant change in all directions



Measure intensity change when shifting a window

- Consider shifting the window W by (u,v),
- The intensity change of each pixel before and after shifting can be defined by summing up the squared differences (SSD)

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



- $I(x,y)$ is the pixel value and $w(x,y)$ is the pixel weight
- However, it is slow to compute exactly for each pixel and each offset (u,v)

The Sum of Squared Differences with Taylor Series

$$E(u,v) = \sum_{(x,y)\in W} \big(I(x+u,y+v) - I(x,y)\big)^2 \quad \rightarrow \quad w(x,y) \text{ is set to 1 for simplicity}$$

$$\approx \sum_{(x,y)\in W} \big(I(x,y) + I_x u + I_y v - I(x,y)\big)^2$$

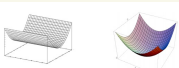
$$= \sum_{(x,y)\in W} \big(I_x u + I_y v\big)^2 \qquad I_x = \frac{\partial I}{\partial x},\ I_y = \frac{\partial I}{\partial y},$$

$$= \sum_{(x,y)\in W} I_x^2 u^2 + 2 I_x I_y uv + I_y^2 v^2$$

$$= [u,v]\boldsymbol{M}[u,v]^T \qquad \rightarrow \quad \boldsymbol{M} \text{ is called the second moment matrix}$$

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

Interpreting Eigenvalues of M



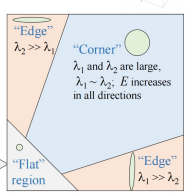
$$R = \det(\boldsymbol{M}) - \alpha\, trace(\boldsymbol{M})^2 = \lambda_1\lambda_2 - \alpha(\lambda_1+\lambda_2)^2$$

α: constant (0.04 to 0.06)

$$R = \frac{\det(\boldsymbol{M})}{trace(\boldsymbol{M}) + \epsilon} = \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2+\epsilon}$$

Calculating eigenvalues is usually slow! Determinant and trace have closed form solution.

λ_1 and λ_2 are small; E is almost constant in all directions



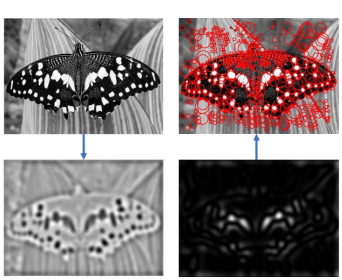
角点检测:
→特征值不变
① 平移等变、旋转等变.
② 像素变换 $I_{new} = \alpha I_{old} + \beta$.
M矩阵与β无关, α影响M
α会增大特征对比度.
③对尺度缩放不具有等变性
Solution: Gauss Pyramid. 对图像不同的图像做角点检测

⇒ Blob. 水滴检测

(斑点)

Summary of 2D Blob Detection

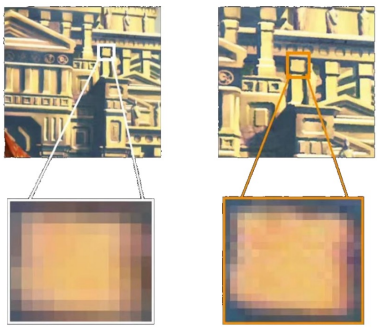
- Filter the image $I(x,y)$ NLOG with different scales $L(x,y;\sigma) = \sigma^2 \nabla G_\sigma(x,y) * I(x,y)$
- The blobs are detected by the extremums of $L(x,\sigma)$: $(\hat{x},\hat{\sigma}) = arg\ MinMax\ L(x,y;\sigma)$
- Here $\hat{x}$ and $\hat{\sigma}$ are the position and the characteristic size of the blob.



(图像块)
Blob 检测动机 ⇒ ① Edge 难定位
② Corners 可定位但描述性不强.

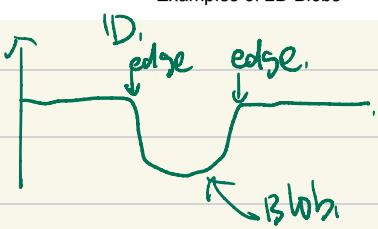
Blobs

- **Blobs** are regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions.
- Blobs have fixed **positions** and **sizes**, can be **localized,** and are good interest points.
- Blob detection is often used to obtain regions of interest for further processing (the precursor of the famous SIFT feature).



Examples of 2D Blobs

Blob detection



Edge Detection with Gaussian Derivatives

Extremum of $f * h'$ is an edge

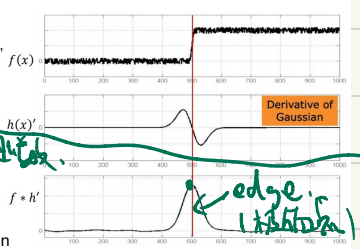
- Set $f(x) = u(x - x_0)$ as a step function, $h(x) = G_\delta(x)$ is a Gaussian function:

$$\begin{aligned}(f * h)' &= f * h' = f' * h \\ &= G_\delta(x) * \delta(x - x_0) \\ &= G_\delta(x - x_0)\end{aligned}$$

- The edge is at the location of x_0

$$G_\delta(0) = \frac{1}{\sqrt{2\pi}\delta}$$

- Use normalized Gaussian derivative to make the extremum does not depend on δ, *i.e.*, $f * \delta\, G_\delta{}'$



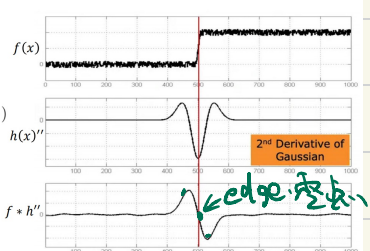
Edge Detection with the 2nd Derivative of Gaussian

Extremum of $f * h'$ is an edge

→ Zero-crossing of $f * h''$ is an edge

$$\begin{aligned}u(x - x_0) * \frac{d^2 G_\sigma(x)}{dx^2} &= \frac{d}{dx}(u(x - x_0) * \frac{dG_\sigma(x)}{dx}) \\ &= \frac{dG_\sigma(x - x_0)}{dx} \\ &= \frac{-(x - x_0)}{\sqrt{2\pi}\sigma^3} e^{\frac{-(x-x_0)^2}{2\sigma^2}}\end{aligned}$$

- The response is 0 when $x = x_0$



接下来要对 edge 面议对
因为 edge1 blob edge2

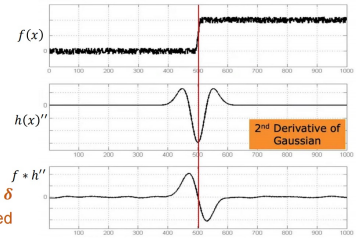
Edge Detection with the 2nd derivative of Gaussian

- Let's inspect the extremum of $f * h''$

$$u(x-x_0) * \frac{d^2 G_\sigma(x)}{dx^2} = \frac{-(x-x_0)}{\sqrt{2\pi}\sigma^3} e^{\frac{-(x-x_0)^2}{2\sigma^2}}$$

$$\frac{d}{dx}\left(u(x-x_0) * \frac{d^2 G_\sigma(x)}{dx^2} \right) = -(1 - \frac{(x-x_0)^2}{\sigma^2}) \frac{1}{\sqrt{2\pi}\sigma^3} e^{\frac{-(x-x_0)^2}{2\sigma^2}}$$

- The extremums of $f * h''$ are at $x = x_0 \pm \delta$ with values $\pm \frac{1}{\sqrt{2\pi}\delta^2} e^{-\frac{1}{2}}$. Use the normalized 2nd derivative of Gaussian: $f * \delta^2 G_\delta''$.



Blob Detection with the 2nd Derivative of Gaussian

- Filter with the 2nd Derivative of Gaussian

$$u(x-x_0) * \frac{d^2 G_\sigma(x)}{dx^2} = \frac{-(x-x_0)}{\sqrt{2\pi}\sigma^3} e^{\frac{-(x-x_0)^2}{2\sigma^2}}$$

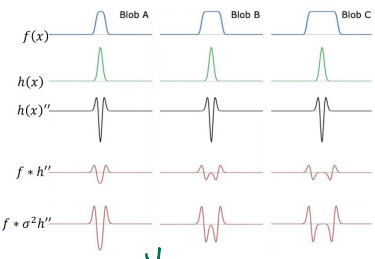
- Define a 1D Blob as

$$f(x) = u(x-x_0) - u(x-x_1)$$

- Filter the 1D Blob

$$f(x) * \sigma^2 \frac{d^2 G_\sigma(x)}{dx^2} = -\frac{(x-x_0)}{\sqrt{2\pi}\sigma} e^{\frac{-(x-x_0)^2}{2\sigma^2}} + \frac{(x-x_1)}{\sqrt{2\pi}\sigma} e^{\frac{-(x-x_1)^2}{2\sigma^2}}$$

- When $\sigma = \frac{x_1 - x_0}{2}$, the extremums coincide at $x = x_0 + \sigma = x_1 - \sigma = \frac{x_1 + x_0}{2}$

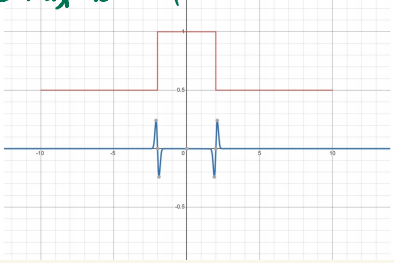


极值点重合，峰值强。

→ * normalized.
使极值点与σ无关。
检测大的Blob需要大的σ，
小的Blob需要小的σ。

Summary of 1D Blob Detection

- Filter the function $f(x)$ with multiscale normalized the 2nd derivative of Gaussians, the result is denoted as $L(x, \sigma)$.
- The blobs are detected by the extremums of $L(x, \sigma)$: $(\hat{x}, \hat{\sigma}) = arg\,MinMax\, L(x, \sigma)$
- Here $\hat{x}$ and $\hat{\sigma}$ are the position and the characteristic size of the blob.
- Characteristic size ∝ Blob size



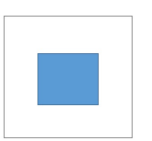
2D Blob Detection: Laplacian of Gaussian

- Intuitively, we can detection blobs in the x direction with the 2nd derivative of Gaussian, $G''(x;\sigma)$, then in the y direction with $G''(y;\sigma)$, and sum the response.

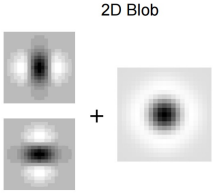
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad G(x,y;\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\nabla^2 G(x,y;\sigma) = \left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x,y;\sigma)$$

- 2D blobs can be detected with multiscale Normalized Laplacian of Gaussian (NLoG: $\sigma^2 \nabla^2 G(x,y;\sigma)$), i.e., filter images with different σ values, and extract extremums.



2D Blob



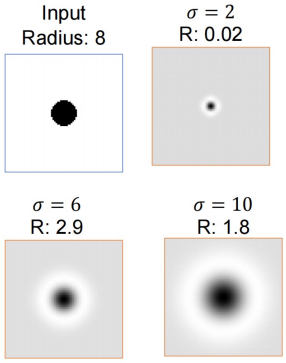
Laplacian of Gaussian

2D Blob Detection: Scale Selection

Given a binary circle and NLoG filter of scale σ, we can compute the response as a function of the scale $I(x, y; \sigma)$.

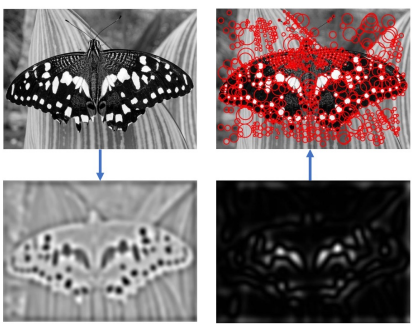
```
for y in range(i-1,i+1+1):
  for x in range(j-1,j+1+1):
    for sigma in range(k-1,k+1+1):
        # find c to ensure the following
        # line to be true
        I[y, x, c] < I[i, j, sigma]
```

R is the response of NLoG on the circle center.



Summary of 2D Blob Detection

- Filter the image $I(x, y)$ NLOG with different scales
  $L(x, y; \sigma) = \sigma^2 \nabla G_\sigma(x, y) * I(x, y)$
- The blobs are detected by the extremums of $L(x, \sigma)$:
  $(\hat{x}, \hat{\sigma}) = arg\ MinMax\ L(x, y; \sigma)$
- Here $\hat{x}$ and $\hat{\sigma}$ are the position and the characteristic size of the blob.



SIFT Feature:

① 加速：DOG 近似 NLoG.
Difference of Gaussian → normalized Laplacian of Gaussian

② Efficient Construction of Gaussian Pyramid.

$$f_Z(z) = \int_{-\infty}^{\infty} f_Y(z-x) f_X(x) dx = \frac{1}{\sqrt{2\pi}\sigma_Z} \exp\left[-\frac{(z-(\mu_X+\mu_Y))^2}{2\sigma_Z^2}\right]$$

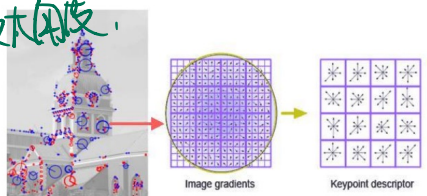
$$\sigma_Z = \sqrt{\sigma_X^2 + \sigma_Y^2}$$

HoG vs SIFT.

Summary of SIFT

→ DOG, 梯度, Gauss Pyramid.

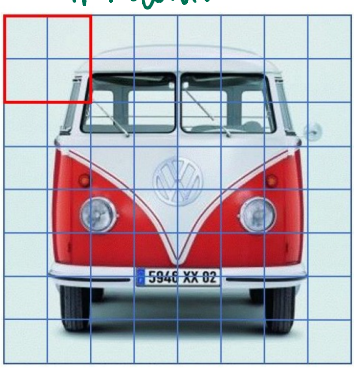
- Blob detection
  - Scale invariant 尺度不变性
- Align the principal orientation → 对齐取大角度.
  - Rotation invariant
- Normalized histograms of gradients
  - Distinctive 区分性
  - Robust to noise and illumination change 归一化.
- Leverage image pyramid → 对高度. 对梯度不敏感.
  - Efficient to compute
- Used for Image matching and recognition



Histogram of Oriented Gradients (HoG) Feature

→ 非 rotation-invariant.

- Local object appearances and shapes can be characterized with the distribution of gradients or even without precise knowledge of the gradient positions.
- 1. Resize the image to 128x64, and calculate the gradients at each pixel.
- 2. The unsigned gradients (magnitudes and angles) are divided into 8x8 cells, and a 9-point histogram is calculated.
- 3. Group 2x2 cells with a stride of 1.
- 4. Normalize the histograms (robust to illumination and shadings)



HoG vs SIFT

- For the computation
  - HoG features are computed in dense grids at a single scale without dominant orientation alignment
  - SIFT featrures are computed at a sparse set of scale-invariant key points, rotated to align their dominant orientations
- For the Usage
  - SIFT features are optimized for sparse wide baseline matching,
  - HOG features are used for dense robust coding of spatial form.

Of Image Stitching

p —Transformation→ p'

Linear: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ (*)

Un-Linear: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ (*)

T-matrix

如何从 p, p' 中还原出 T-matrix.

Homography

对于线性(*):

$$\begin{cases} x' = ax + by + c \\ y' = dx + ey + f \end{cases}$$

$$\therefore \begin{pmatrix} x_1' \\ y_1' \\ \vdots \\ x_n' \\ y_n' \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ & & & \vdots & & \\ x_n & y_n & 0 & 0 & 1 & 0 \\ 0 & 0 & x_n & y_n & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}$$

$$b_{2n\times 1} = A_{2n\times 6} \times t_{6\times 1}$$

分段求解 $t_{6\times 1}$.

未知变量数!

$E = |At - b|_2^2 = t^T A^T A t - 2 t^T A^T b + b^T b$

令 $A^T b = A^T A t$. (对左乘于 t 求导)

$\hookrightarrow t = (A^T A)^{-1} A^T b$. (Least Squares)

对于非线性(*):

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \simeq \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\rightsquigarrow \begin{cases} x_i' = \dfrac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y_i' = \dfrac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{cases}$$

$$\rightsquigarrow x_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02},$$
$$y_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12}$$

$$\Rightarrow \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ \vdots & & & & & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$A_{2n\times 9}$ $h_{9\times 1}$ $0_{2n\times 1}$

拉格朗日法

$$\Rightarrow E = \|Ah\|^2 + \lambda(\|h^2\| - 1) = h^TA^TAh + \lambda h^Th - \lambda.$$

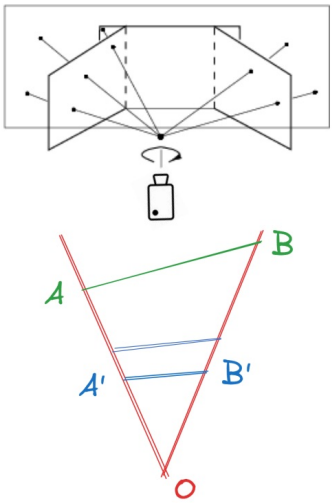
更通俗的说法：

$$\therefore A^TAh = \lambda h, \quad E = \lambda.$$

对 h 求偏导.

h 为 A^TA 最小的特征值对应的特征向量

Homography



Why is the homography matrix a 3×3 matrix?

- Transform image AB to $A'B'$, where $A'B'$ is the desired image plane.
- The transformation matrix between OAB and $OA'B'$ is apparently a full rank 3×3 matrix.
- Note that the camera center O is the shared camera center.
- If we transform AB to an image plane in parallel with $A'B'$, we get the same image. Therefore **the degree of freedom** of homography is **8**.

Summary of Solving Homographies

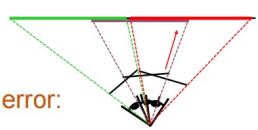
Summary

- Build the matrix ***A*** according to the homography transformation.
- Compute the Eigenvector of $\boldsymbol{A^T A}$ with the smallest eigenvalue.

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Caveat

- $||\mathbf{Ah}||^2$ measures the *algebraic error* for optimization.
- Use the following function to measure the geometric error:



$$\sum_{i=1}^{k} \|[x_i', y_i'] - T([x_i, y_i])\|^2 + \|[x_i, y_i] - T^{-1}([x_i', y_i'])\|^2$$

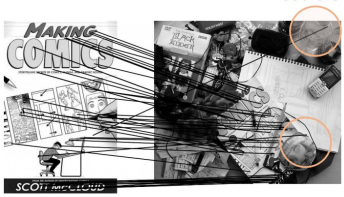
⇒ Feature Matching,

Given images A and B

- Compute image features for A and B
- Match features between A and B with the nearest neighborhood searching
- Compute homography between A and B on the set of matches

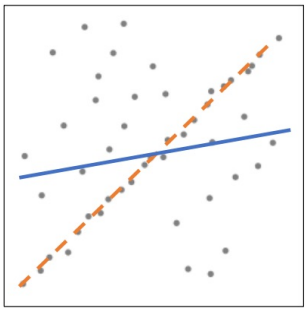
- What could go wrong?

outliers



The Affect of Outliers

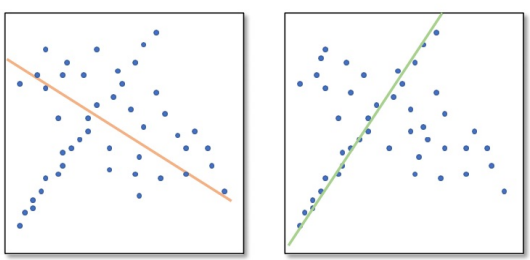
- Linear Regression
  - $y = ax + b$
  - $E = \sum_i (ax_i + b - y_i)^2$
  - Minimize E to find $a,\ b$



- If there are a lot of outliers, we get a result far from the ground truth.
- The key is to eliminate the affect of outliers

RANSAC: RANdom SAmple Consensus

- Randomly choose s samples
  - Typically s = minimum sample size that lets you fit a model
- Fit a model to those samples
  - Count the number of inliers that approximately fit the model
- Repeat N times
- Choose the model that has the largest set of inliers
- Fit the model with all inliers



Inliers: 3 Inliers: 20

RANSAC: RANdom SAmple Consensus

- How to find inliers?
  - After fitting the model, use an inlier threshold related to the amount of noise
  - We can use the geometric error of homography to measure the inliers/outliers
- When to terminate?
  - The number of iterations is related to the percentage of outliers, and the probability of success we'd like to guarantee
  - Suppose the proportion of inliers is G and the model needs P pairs to fit, the probability that we have not picked a set of inliers after N iterations: $(1 - G^P)^N$.
  - If $G = 50\%, P = 4, N = 100$, then $0.5^4 = 0.0625,\ (1 - 0.5^4)^{100} = 0.00157$
  - If $G = 30\%, P = 4, N = 100$, then $0.3^4 = 0.0081,\ (1 - 0.3^4)^{100} = 0.44339,$ $(1 - 0.3^4)^{500} = 0.017137,\ (1 - 0.3^4)^{1000} = 0.000293$
  - If we want a failure probability of at most e, then $N > \frac{\log e}{\log(1 - G^P)}$.

RANSAC: RANdom SAmple Consensus

- Pros
  - Simple and general
  - Applicable to many problems
  - Often works well in practice
- Cons
  - Has parameters to tune
  - May fail for low inlier ratios
  - Requires too many iterations
- Other methods to deal with outliers
  - Robust statistics

outliers

⇒ Image Blending.

① Image Blending with Laplacing Pyramid

Review: Image Blending with Laplacian Pyramid

- Blend high frequency bands over smaller spatial ranges

- Blend lower frequency bands over larger spatial ranges.
- The long range blending may mix contents from two image.

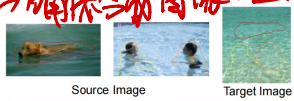
② Possion Image Editing

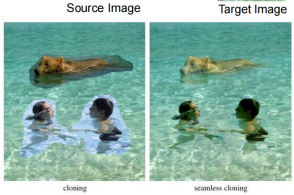
Poisson Image Editing

- The tones of the source images are not compatible with the target images.
- Key Idea: the gradients of the source images should be kept.

$$E = \min_f \sum_{(i,j)\in\Omega} \|\nabla f(i,j) - \nabla g(i,j)\|_2^2$$
$$f(i,j) = f^*(i,j) \ \ for\ (i,j) \in \partial\Omega$$

- where g is the source image, and f^* is the targe image, f is the output image, Ω is the region for blending.



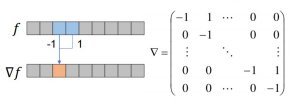


Poisson Image Editing

- How to solve this optimization problem?

$$E = \min_f \sum_{(i,j)\in\Omega} \|\nabla f(i,j) - \nabla g(i,j)\|_2^2 \qquad f(i,j) = f^*(i,j) \ \ for\ (i,j) \in \partial\Omega$$

- ∇f can be written in a matrix form



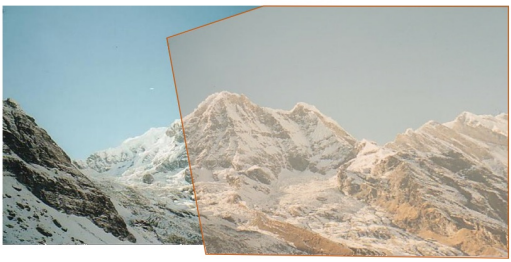
Least Squares

$$\min_f \|Af - G\|^2 + \lambda\|Bf - F\|^2$$
$$(A^TA + \lambda B^TB)f = b$$

Large-scale Sparse Matrix

Poisson Image Editing for Panorama

- Solve the homography
- Warp the source image to the reference image
- Warp the mask image to the reference image
- Run the Poisson Editing algorithm by keeping the gradient in the mask region



06 3D Vision and Camera Calibration

3D Vision: Reconstruct 3D structures from Images.

Key Challenge: Single-view Ambiguity. (无视觉信息)

Solution: ① Shoot light out of sensors. (无感器信息)

② Stereo: use 2 calibrated cameras (双目视觉)

③ Multi-view geometry: move camera (移动摄像机)

④ Shape from shading: 固定相机，获得不同光照下的照片

⑤ Learn from data: 神经网络

Overview of 3D Vision

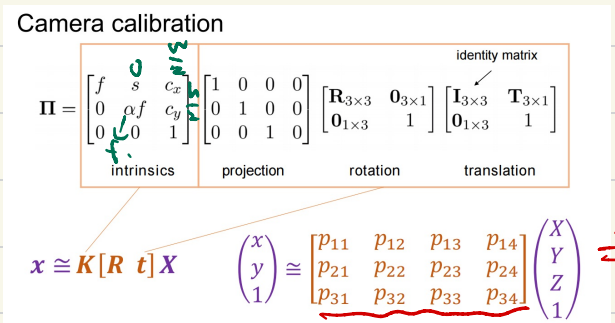
Camera calibration
- Intrinsic matrix

Stereo
- Depth map from 2 images

Structure from motion
- Cameras, pointcloud from 2+ images

Shape from shading
- 3D geometry from 2+ images

Camera calibration



$$x \cong K[R\ t]X \qquad \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cong \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\Rightarrow \begin{cases} x_i = \dfrac{p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}} \\ y_i = \dfrac{p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}} \end{cases}$$

$$\Rightarrow A_{2n\times 12} \cdot P_{12\times 1} = 0_{2n\times 1}$$

$$E = \|AP\|^2 + \lambda(\|P\|^2 - 1) = P^T A^T A P + \lambda P^T P - \lambda$$

$$A^T A h = \lambda h$$

h 为 $A^T A$ 最小特征值对应的特征向量.

$E = \lambda$.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cong \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad \text{vs.} \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} \cong K[R|t] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

可以进行 RQ 分解.

Camera Calibration: Linear vs. nonlinear

- Linear calibration is easy to formulate and solve, but it doesn't directly tell us the camera parameters
- In practice, non-linear methods are preferred
  - Write down objective functions as sum of squared distances between measured 2D points and estimated projections of 3D points:

$$\sum_i \|\text{proj}(K[R \mid t]X_i) - x_i\|_2^2 \qquad \text{proj: } x' = x / z \text{ and } y' = y / z$$

  - We can include radial distortion and constraints or other constraints
  - Minimize the error using non-linear optimization packages
  - Initialize solution with output of linear method (use RQ decomposition to get K and R from P)

先用线性结果初始化.

Triangulation

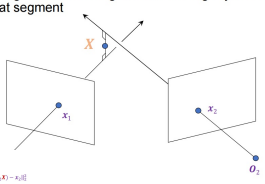
前面我们求了 已知3D点，图像点(2D)，求相机内参。

Triangulation 讲的 已知2D点，相机参数，求3D点。（一张图片很不准确，会至多个相机）！

→ Calibration.

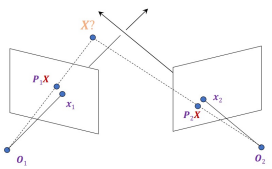
Triangulation #1: Geometric Approach

- Find shortest segment connecting the two viewing rays and let X be the midpoint of that segment



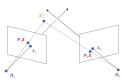
Triangulation #2: Nonlinear Approach

- Find $\boldsymbol{X}$ that minimizes $\|\text{proj}(\boldsymbol{P}_1\boldsymbol{X}) - \boldsymbol{x}_1\|_2^2 + \|\text{proj}(\boldsymbol{P}_2\boldsymbol{X}) - \boldsymbol{x}_2\|_2^2$



Triangulation #2: Nonlinear Approach

- Find X that minimizes $\|\text{proj}(P_1X) - x_1\|_2^2 + \|\text{proj}(P_2X) - x_2\|_2^2$



.inear Optimization

$$\boldsymbol{x}_1 \cong \boldsymbol{P}_1\boldsymbol{X} \qquad \boldsymbol{x}_1 \times \boldsymbol{P}_1\boldsymbol{X} = \boldsymbol{0} \qquad [\boldsymbol{x}_{1\times}]\boldsymbol{P}_1\boldsymbol{X} = \boldsymbol{0}$$
$$\boldsymbol{x}_2 \cong \boldsymbol{P}_2\boldsymbol{X} \qquad \boldsymbol{x}_2 \times \boldsymbol{P}_2\boldsymbol{X} = \boldsymbol{0} \qquad [\boldsymbol{x}_{2\times}]\boldsymbol{P}_2\boldsymbol{X} = \boldsymbol{0}$$

Cross product as matrix product

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [\boldsymbol{a}_\times]\boldsymbol{b}$$

$$\begin{matrix}[\boldsymbol{x}_{1\times}]\boldsymbol{P}_1\boldsymbol{X} = \boldsymbol{0} \\ [\boldsymbol{x}_{2\times}]\boldsymbol{P}_2\boldsymbol{X} = \boldsymbol{0}\end{matrix} \rightarrow \begin{matrix}([\boldsymbol{x}_{1\times}]\boldsymbol{P}_1)\boldsymbol{X} = \boldsymbol{0} \\ ([\boldsymbol{x}_{2\times}]\boldsymbol{P}_2)\boldsymbol{X} = \boldsymbol{0}\end{matrix} \rightarrow$$

Two equations per camera for 3 unknows in X, solve with constrained Least Squares.

Triangulation – Linear Optimization

- Prefer X = (X, Y, Z, W) in practice with constrain $\|X\|^2 = 1$, why?

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cong \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix}$$

$$x_j = \frac{p_{11}^{(j)}X + p_{12}^{(j)}Y + p_{13}^{(j)}Z + p_{14}^{(j)}W}{p_{31}^{(j)}X + p_{32}^{(j)}Y + p_{33}^{(j)}Z + p_{34}^{(j)}W} \rightarrow \text{Solve with Eigen decomposition}$$

$$y_j = \frac{p_{21}^{(j)}X + p_{22}^{(j)}Y + p_{23}^{(j)}Z + p_{24}^{(j)}W}{p_{31}^{(j)}X + p_{32}^{(j)}Y + p_{33}^{(j)}Z + p_{34}^{(j)}W}$$

Calibration from vanishing points

- We align the world coordinate system with three orthogonal vanishing directions in the scene.

$$e_1 = \begin{pmatrix}1\\0\\0\end{pmatrix}, \quad e_2 = \begin{pmatrix}0\\1\\0\end{pmatrix}, \quad e_3 = \begin{pmatrix}0\\0\\1\end{pmatrix} \qquad v_i \cong P\begin{pmatrix}e_i\\0\end{pmatrix} = K[R|t]\begin{pmatrix}e_i\\0\end{pmatrix}$$

- Can we get more constrains from axes? $e_i^T e_j = 0$

$$v_i \cong KRe_i \qquad e_i \cong R^T K^{-1} v_i \qquad \underbrace{v_i^T K^{-T}}_{e_i^T} RR^T K^{-1} \underbrace{v_j}_{e_j} = 0$$

$$v_i^T K^{-T} K^{-1} v_j = 0 \quad \text{Only contains the intrinsic matrix}$$

Calibration from vanishing points

$$v_i^T K^{-T} K^{-1} v_j = 0$$

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad K^{-1} = \begin{bmatrix} g & 0 & -gc_x \\ 0 & g & -gc_y \\ 0 & 0 & 1 \end{bmatrix} \quad K^{-T} = \begin{bmatrix} g & 0 & 0 \\ 0 & g & 0 \\ -gc_x & -gc_y & 1 \end{bmatrix} \quad g = \frac{1}{f} \quad v_i = (x_i, y_i, w_i)$$

$$v_i^T K^{-T} K^{-1} v_j = g^2(x_i x_j + y_i y_j) - g^2 c_x(w_i x_j + w_j x_i) - g^2 c_y(w_i y_j + w_j y_i) + g^2(c_x^2 + c_y^2) w_i w_j + w_i w_j = 0$$

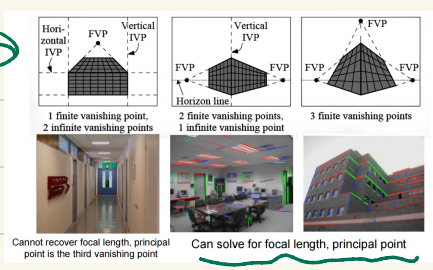
- A couple of complications
  - The constraints are nonlinear, but it's not hard to solve
  - At least two **finite** vanishing points are needed to solve for both f and c_x, c_y

Calibration from vanishing points

- How many constraints do we get? $v_i^T K^{-T} K^{-1} v_j = 0$
  - 3: one for each pair of vanishing points.
- How many unknown parameters does K have?
  - K has 3 parameters.

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- A couple of complications
  - The constraints are nonlinear, but it's not hard to solve



Rotation from vanishing points

- Constraints on vanishing points: $v_i \cong KRe_i$
- We just use the orthogonality constraints to solve for R
- Now we have: $K^{-1}v_i \cong Re_i$
- Notice: $Re_1 = [r_1 \quad r_2 \quad r_3]\begin{pmatrix}1\\0\\0\end{pmatrix} = r_1$
- Thus, $r_i \cong K^{-1}v_i$
- The scale ambiguity goes away since we require $\|r_i\|^2 = 1$

Calibration with Vanishing Points

Homework.

Calibration from Vanishing Points

- 1. Solve for intrinsic parameters (focal length, principal point) using three orthogonal vanishing points
- 2. Get extrinsic parameters (rotation) directly from vanishing points once calibration matrix is known
- Advantages
  - No need for calibration chart, 2D-3D correspondences
  - Could be completely automatic
- Disadvantages
  - Only applies to certain kinds of scenes
  - It is tricky to accurately localize vanishing points, and need at least two finite vanishing points

1. 使用三个正交消失点求解内参（intrinsic parameters）

内参：

- 焦距（focal length）
- 主点（principal point），即成像平面中心的位置

步骤：

- **选择消失点**：在图像中找到三条相互垂直的直线的消失点。这些直线通常对应于场景中的三维直线投影，例如建筑物的边缘或道路的标记。
- **建立方程**：由于这些直线在三维空间中是相互垂直的，它们的消失点在归一化图像平面（归一化焦距为1）上也会形成直角。利用这一点，可以建立方程来求解相机的内参。
- **求解内参**：通过解这些方程，可以得到相机的焦距和主点位置。通常，这涉及到求解一个线性或非线性优化问题。

2. 知道校准矩阵后，直接从消失点获取外参（extrinsic parameters）

外参：

- 旋转（rotation）
- 平移（translation）：但在消失点方法中通常不直接估计平移。

步骤：

- **使用内参**：一旦获得了内参，就可以构建相机的校准矩阵。
- **估计旋转**：通过观察消失点的位置和它们与图像原点的相对位置，可以直接估计相机的旋转。这是因为消失点的位置受到相机旋转的影响。

Summary

- Reconstruct 3D structures from images
  - Key: Resolve the single-view ambiguity
- Camera matrix: $\boldsymbol{x} \cong \boldsymbol{P}\boldsymbol{X} = \boldsymbol{K}[\boldsymbol{R}\ \boldsymbol{t}]\boldsymbol{X}$
- Calibration with known grid points
- Calibration with vanishing points
- Given point pairs in correspondence, we can derive equations to solve for $\boldsymbol{P}$ (calibration) or $\boldsymbol{X}$ (triangulation)

- Reference:
  - Computer Vision: Algorithms and Applications, Chapter 11

07 Epipolar Geometry.

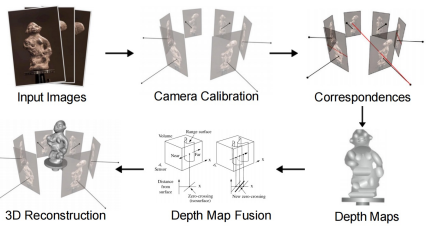
Contents


Two-View Stereo


Epipolar Geometry

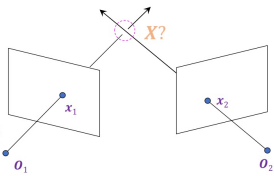
Essential / Fundamental Matrix

Typical 3D Reconstruction Pipeline



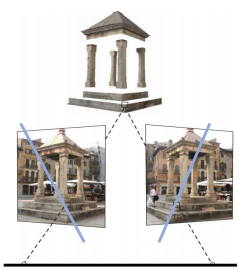
Review: Triangulation

- Given projections of a 3D point in two or more images (with known camera matrices), find the coordinates of the point.
- Can the setup of two cameras provide more for 3D estimation?

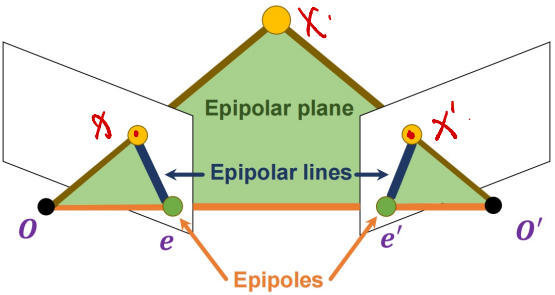


Two-view Stereo: Key Idea

- Consider the projections of camera centers and visual rays into the other view without explicit 3D reasoning
- We can calibrate the two cameras.
- We can find constraints for easier correspondence and easier 3D reconstruction.



Epipolar Geometry.

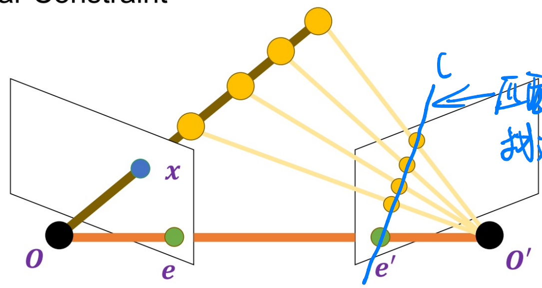


O, O' : 카메라 중심 (렌즈 중심).

e, e' : Epipoles,

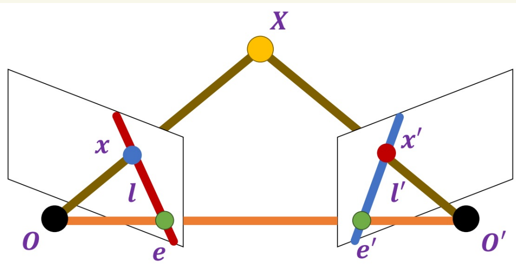
Essential Matrix & Fundamental Matrix.

Epipolar Constraint



- Where can we find the x' corresponding to x in the other image?

l
c ← 匹配线是确定的。
找对应点只需考虑匹配线(极线)即可。

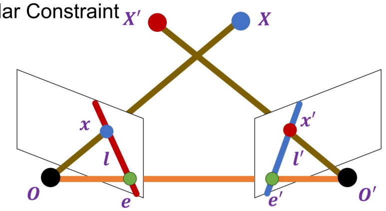


- Whenever two points x and x' lie on matching epipolar lines l and l', the visual rays corresponding to them meet in space, i.e., x and x' could be projections of the same 3D point X

⇒ Epipolar Constraint.
① x → l'. x'在l'上
② x' → l， x在l上.

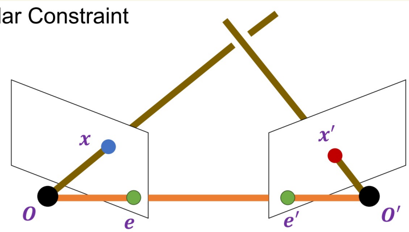
↓ 仍存在问题.

Epipolar Constraint



- Caveat: if x and x' satisfy the epipolar constraint, this doesn't mean they *must be* projections of the same 3D point

Epipolar Constraint

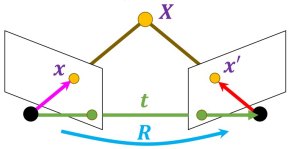


- Remember: in general, two rays *do not* meet perfectly in space due to noise and inaccurate calibration!

Epipolar Constraint: Calibrated Case

- Suppose camera intrinsic parameters are known, and the world coordinate system is set to that of the first camera.
- Then the projection matrices are given by $K[I \mid 0]$ and $K'[R \mid t]$
- We can multiply the projection matrices and the image points by the inverse calibration matrices to get *normalized* image coordinates:

$$x = K^{-1}x_{\text{pixel}} \cong [I \mid 0]X$$
$$x' = K'^{-1}x'_{\text{pixel}} \cong [R \mid t]X$$



已知条件

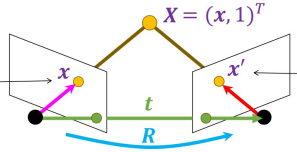
- **相机内部参数**: 两个相机的内部参数矩阵 (K, K') 已知。 P36
- **世界坐标系**: 世界坐标系与第一个相机坐标系重合。 P36
- **图像点**: 在第一个相机图像中观测到一个点 x。

求解目标

- **寻找对应点**: 找到在第二个相机图像中与 x 对应的点 x′。
- **三维重建**: 利用 x 和 x′ 的对应关系以及相机参数，重建三维点 X 的坐标。

Epipolar Constraint: Calibrated Case

$$x \cong [I \mid 0]X$$
$$x \cong [I \mid 0]\begin{pmatrix} X \\ 1 \end{pmatrix}$$



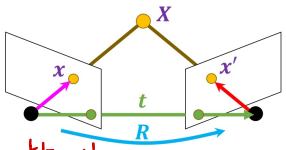
$$x' \cong [R \mid t]X$$
$$x' \cong [R \mid t]\begin{pmatrix} X \\ 1 \end{pmatrix} \cong Rx + t$$

- We have $x' \cong Rx + t$
- This means the three vectors x', Rx, and t are *linearly dependent*
- This constraint can be written using the *triple product*: $x' \cdot [t \times (Rx)] = 0$

Epipolar Constraint: Calibrated Case

$$x' \cdot [t \times (Rx)] = 0$$
$$\Downarrow$$
$$x'^T[t_\times]Rx = 0$$
$$\Downarrow$$
$$x'^T Ex = 0$$
$$\Downarrow$$

E is the Essential Matrix



$$(x', y', 1)\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$$

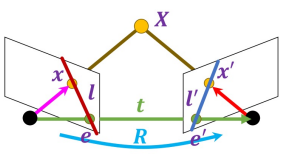
Epipolar Constraint: Calibrated Case

E is the Essential Matrix

$$x'^T Ex = 0$$

- The epipolar line associated with x: $x'^T Ex = x'^T l' = 0, \ \text{where } l' = Ex$
- The epipolar line associated with x': $x'^T Ex = l^T x = 0, \ \text{where } l = E^T x'$

$Ee = 0$ and $E^T e' = 0$



Note: a line is given by $ax + by + c = 0$ or $l^T x = 0$ where $l = (a, b, c)^T$ and $x = (x, y, 1)^T$

Epipolar Constraint: Calibrated Case

- E is singular with a rank of 2
  - $[t_\times]$ has a rank of 2
  - R has a rank of 3
- E has 5 degrees of freedom
  - Translation: 3 degrees of freedom
  - Rotation: 3 degrees of freedom
  - Scaling ambiguity

$$E = [t_\times]R$$
$$= \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}$$

Epipolar Constraint: Uncalibrated case

- Assume the calibration matrices K and K' of the two cameras are unknown
- Recall the epipolar constraint in terms of normalized coordinates is

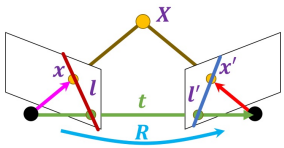
$$x'^T E x = 0,$$

where $x = K^{-1}x_{pixel}$, $x' = K'^{-1}x'_{pixel}$

- We can get the following constraint

$$x'^T_{pixel} F\, x_{pixel} = 0$$

where $F = K'^{-T}EK^{-1}$, F is called the fundamental matrix.

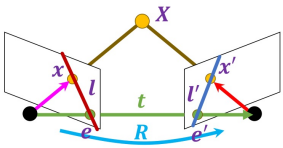


Epipolar Constraint: Uncalibrated Case

- F is the fundamental matrix

$$x'^T F x = 0$$

- The epipolar line associated with x: $x'^T F x = x'^T l' = 0$, *where* $l' = Fx$
- The epipolar line associated with x': $x'^T F x = l^T x = 0$, *where* $l = F^T x'$
- $Fe = 0$ and $Fe' = 0$



Note: a line is given by $ax + by + c = 0$ or $l^T x = 0$ where $l = (a, b, c)^T$ and $x = (x, y, 1)^T$
Note: We use x to replace x_{pixel} for simplicity.

Epipolar Constraint: Uncalibrated Case

- F is singular with a rank of 2 $F = K'^{-T}[t_\times]RK^{-T}$
  - $[t_\times]$ has a rank of 2
  - R has a rank of 3
  - K is of full rank

$$= K'^{-T}\begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}K^{-T}$$

- F has 7 degrees of freedom
  - Det(F) = 0
  - Scaling ambiguity

Fundamental Matrix

- Given correspondences $x = (x, y, 1)^T$ and $x' = (x', y', 1)^T$
- Constraint: $x'^T F x = 0$

$$(x', y', 1)\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0 \Rightarrow (x'x, x'y, x', y'x, y'y, y', x, y, 1)\begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} = 0$$

- Given correspondences $x = (x, y, 1)^T$ and $x' = (x', y', 1)^T$
- Constraint: $x'^T F x = 0$

$$\underbrace{\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'x & x'y & x' & y'x & y'y & y' & x & y & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}}_{U\ n\times 9}\begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} = \mathbf{0}$$

- Use homogeneous least squares to find f:

$$\arg\min_{\|f\|=1} \|Uf\|_2^2 \longrightarrow$$ Eigenvector of $U^T U$ with the smallest eigenvalue

Enforcing the Rank-2 Constraint

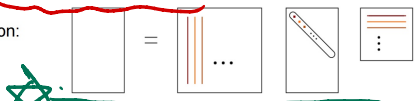
- F needs to be singular/rank 2. How do we force it to be singular?
- SVD decomposition: $M_{m\times n} = U_{m\times m} \cdot \Sigma_{m\times n} \cdot V^T_{n\times n}$

Proof by construction:

$$Mv_i = \sigma_i u_i$$
$$MV = U\Sigma$$
$$M = U\Sigma V^T$$

How to find U and V

$$M^T M = V\Sigma^2 V^T$$
$$MM^T = U\Sigma^2 U^T$$



$\min_{M_k} \|M - M_k\|_F^2$, where k is the rank of M_k

Low Rank Approximation: $M = \sum_i \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$, keep the first k components after SVD

https://en.wikipedia.org/wiki/Low-rank_approximation

Enforcing the Rank-2 Constraint

- F needs to be singular/rank 2. How do we force it to be singular?
- Solution: take SVD of the initial estimate and throw out the smallest singular value.

$$F_{\text{init}} = U\Sigma V^T$$

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \Sigma' = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$F = U\Sigma' V^T$$

对任意m行n列的矩阵A，奇异值分解的综合理解如下：

$$A = U\Sigma V^T = (\text{orthogonal})(\text{diagonal})(\text{orthogonal}).$$

正交矩阵U：m行m列，该矩阵的每一个列向量都是AA^T的特征向量；

正交矩阵V：n行n列，该矩阵的每一个列向量都是A^TA的特征向量；

对角阵Σ：m行n列，将A^TA或AA^T的特征值开根号，得到的就是该矩阵主对角线上的元素，也可以看成矩阵A的奇异值。

九. 例题分析：秩为2

对以下矩阵A进行SVD分解，并分析相关性质：

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

解：

显然，矩阵A的秩为2，易运算AA^T如下：

$$AA^T = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

可以分析出该方阵的特征值为3和1.

对原始矩阵A进行SVD分解如下：

$$\begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} = U\Sigma V^T = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{matrix} /\sqrt{6} \\ /\sqrt{2} \\ /\sqrt{3} \end{matrix}$$

可以发现该矩阵的奇异值为$\sqrt{3}$和$\sqrt{1}$.

矩阵U的每一列可以看成A的左奇异向量，也可以看成AA^T的单位特征向量；

矩阵V的每一列可以看成A的右奇异向量，也可以看成A^TA的单位特征向量；

对应

Homography Matrix：单应矩阵：反映图像间像素点的关系

↳ 两张图片 对应点 $p_1 \xrightarrow{H} p_1'$

↳ 用于图像拼接。

Essential Matrix：本质矩阵：反映空间一点P在不同视角相机下相机坐标系中的表示之间的关系。

↳ 空间一点P，在相机A下 $X_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$，相机B下 $X_1' = \begin{bmatrix} x_1' \\ y_1' \\ z_1' \end{bmatrix}$.

⇒ 有 $X_1'^T E X_1 = 0$. （E为极线）

Fundamental Matrix：基础矩阵：反映空间一点P在不同视角相机下的像素坐标之间的关系。

↳ 空间一点P，相机A下图像坐标 $x_1 = \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$，相机B下图像坐标 $x_1' = \begin{bmatrix} u_1' \\ v_1' \\ 1 \end{bmatrix}$.

⇒ 有 $x_1'^T F x_1 = 0$

E/F的计算：根据 x_i, x_i' (pair) 计算.
⇒ 由得到的 E/F 还原出 R, t.

From Epipolar Geometry to Camera Calibration

- Estimating the fundamental matrix is known as "weak calibration"
- If we know the intrinsic matrices of the two cameras, we can estimate the essential matrix: $E = K'^{T} F K$
- The essential matrix gives us the relative rotation and translation between the cameras, or their extrinsic parameters
- Alternatively, if the intrinsic matrices are known (or in practice, if good initial guesses of the intrinsics are available), a five-point algorithm can be used to estimate relative camera pose

One Application of the Essential Matrix

- The essential matrix E describes the relative motion of two cameras.
- Given an estimated E, we can get the relative rotation and translation, i.e., the extrinsic matrices (Reference: Multiple View Geometry in Computer Vision, *Section 9.6.2*)
- Then we can use the intrinsic and extrinsic matrices to do calculate the 3D positions via triangulation.

$$E = [t_\times]R$$

$$= \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}$$

Decompose R, t from an Essential Matrix

- Observing $t^T E = t^T [t_\times] R = 0$, we can get t via SVD (up to a scale) ⇒ 这就得出t的方向.

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \mathbf{t} \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0^T \\ \mathbf{v}_1^T \\ \mathbf{v}_2^T \end{bmatrix}$$

- $[t_\times]$ projects a vector onto a set of orthogonal basis vectors that include t, zeros out the t component, and rotates the other two by 90°

$$[\hat{\mathbf{t}}]_\times = \mathbf{S}\mathbf{Z}\mathbf{R}_{90^\circ}\mathbf{S}^T = \begin{bmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \hat{\mathbf{t}} \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & \\ 1 & 0 & \\ & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{s}_0^T \\ \mathbf{s}_1^T \\ \hat{\mathbf{t}}^T \end{bmatrix}$$

$$\mathbf{E} = [\hat{\mathbf{t}}]_\times \mathbf{R} = \mathbf{S}\mathbf{Z}\mathbf{R}_{90^\circ}\mathbf{S}^T\mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad \text{where } \hat{\mathbf{t}} = \mathbf{s}_0 \times \mathbf{s}_1$$

- From the following equation we know that $S = U$, since $\Sigma = Z$

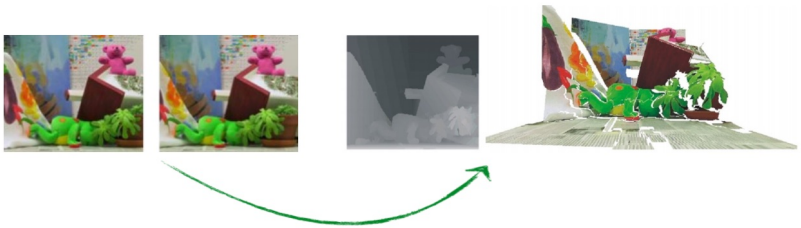
$$\mathbf{E} = [\hat{\mathbf{t}}]_\times \mathbf{R} = \mathbf{S}\mathbf{Z}\mathbf{R}_{90^\circ}\mathbf{S}^T\mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

- Therefore: $\mathbf{R_1} = \mathbf{U}\mathbf{R}_{90^\circ}^{\mathbf{T}}\mathbf{V}^{\mathbf{T}}$ or $\mathbf{R_2} = \mathbf{U}\mathbf{R}_{-90^\circ}^{\mathbf{T}}\mathbf{V}^{\mathbf{T}}$
- There are 4 final solutions $(\mathbf{R_1}, \mathbf{t})$, $(\mathbf{R_1}, -\mathbf{t})$, $(\mathbf{R_2}, \mathbf{t})$, $(\mathbf{R_2}, -\mathbf{t})$
- Determine the final one by checking the positive depth constraint.
  - For a 3D point to be valid, it must be in front of both cameras
  - By triangulating the points with each possible solution and checking their depths, we can identify which configuration is correct.

08 Two-View Stereo. 双目视觉匹配.

Two-View Stereo

- **Input**: a stereo pair with known camera matrices (calibrated)
- **Output**: a dense depth map



Basic Stereo Matching Algorithm

三角测量：通过不同位置观测一个路标点观察，从观察到的角度来推断路标点的深度。

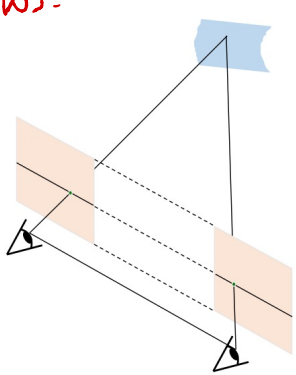
- With a pair of calibrated cameras, we can reconstruct 3D by:
  - Correspondence + Triangulation
- For each pixel in the first image
  - Find the corresponding epipolar line in the right image
  - Examine all pixels on the epipolar line and pick the best match
  - Triangulate the matched points to get depth information

找匹配点.

在对极线，找匹配点..

A Simple Stereo System (简化).

- The simplest case: epipolar lines = corresponding scanlines
- Image planes of cameras are parallel to each other and to the baseline
- Camera centers are at the same height
- Focal lengths are the same
- Then epipolar lines fall along horizontal scan lines of the images



The Essential Matrix for Simple Stereo

Epipolar constraint for simple stereo:

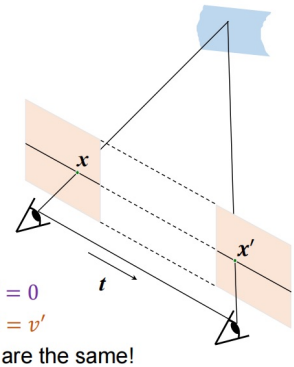
$$x'^T E x = 0, \qquad E = [t_\times]R$$

$$R = I, \quad t = (t, 0, 0)$$

$$E = [t_\times]R = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t \\ 0 & t & 0 \end{bmatrix}$$

$$(u'\ v'\ 1)\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t \\ 0 & t & 0 \end{bmatrix}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0 \Rightarrow -tv + tv' = 0$$

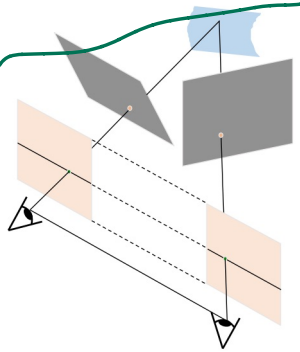
$$v = v'$$



The v-coordinates of corresponding points are the same!

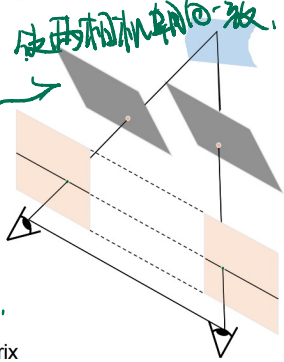
A General Stereo System

- For a general stereo system, the image planes are usually not parallel
- We can use the fundamental matrix to find the homographies to project each view onto a common plane parallel to the baseline (Stereo Rectification)
- Reference:
  - Computing Rectifying Homographies for Stereo Vision. CVPR 1999
  - Multiple View Geometry in Computer Vision, Section 11



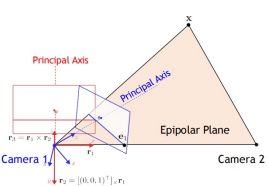
Stereo Image Rectification: A Simple Implementation

- Compute R_1 from the essential or fundamental matrix
- Rotate the right camera by R_1 to align the orientation of two cameras
- Rotate the two cameras by R_{rect} so that the image planes are parallel to the baseline, i.e., map the original epipoles to infinity
- Scale both images by H to reduce the distortion



Calculating the Rectifying Rotation Matrix

- Compute the epipole e_1 with SVD since $Fe_1 = 0$
- Construct the $R_{rect} = (r_1, r_2, r_3)^T$
  - $r_1 = \frac{e_1}{\|e_1\|}$
  - $r_2 = \frac{[(0,0,1)^T]_\times r_1}{\|[(0,0,1)^T]_\times r_1\|}$
  - $r_3 = r_1 \times r_2$
- $R_{rect}e_1 = (\|e_1\|, 0, 0)^T$, which is an infinite point



(简化版)

现在找到了匹配点，如何根据匹配点绘制深度图？

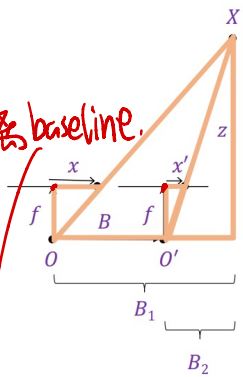
Depth from Disparity

$$\frac{x}{f} = \frac{B_1}{z} \qquad \frac{x'}{f} = \frac{B_2}{z}$$

$$\frac{x - x'}{f} = \frac{B_1 - B_2}{z}$$ → OS'间的距离 baseline.

$$x - x' = \frac{fB}{z} \Rightarrow z = \frac{fB}{x - x'}$$

Disparity is inversely proportional to depth!



(两个相似).

Effect of Baseline on Stereo Results

$$z = \frac{fB}{x - x'}$$

Disparity变小，误差小了.

- Larger baseline
  - \+ Smaller triangulation error
  - – Matching is more difficult

- Smaller baseline
  - – Higher triangulation error
  - \+ Matching is easier

匹配难了，难匹配！(差异变大).

Local Stereo Matching

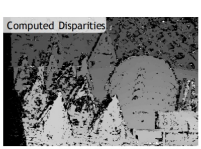
Basic Stereo Matching Algorithm

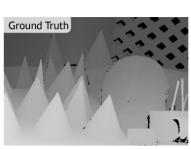
- With a pair of calibrated cameras, we can reconstruct 3D by:
  - Correspondence + Triangulation
- *Rectify two stereo images if required*
- For each pixel in the first image
  - Examine all pixels on the scanline and pick the best match
  - Compute the disparity $x - x'$ and set $z = Bf/(x - x')$

校正

立体视觉了来.

Local Stereo Matching





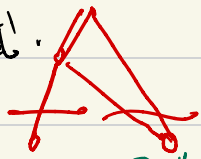
- Choose disparity range [0, D]
- For each pixel, x = (x, y) compute the best disparity
- Do this for both images and apply left-right consistency check to remove outliers

→ NCC/SAD/SSD.

Local ⇒ ① Halt occlusion：遮挡，一侧能看见多侧信息.

② Effect of window size：noise. detail.

③ Failure Cases：重复图样，无纹理区域，高光反射.



↘ ⇝ Non-local Constraints：

① Uniqueness：左侧图一个像素对应右侧图像素

② Order：从左往右，保序.

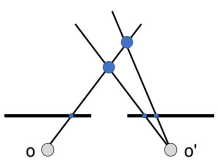
③ Smoothness：左侧相邻点，视差也相近.

Global Stereo Matching.

- Local stereo finds the best match for each window independently.
- Need non-local constraints and optimize the matching globally.

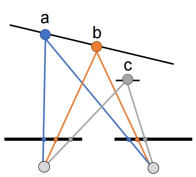
Non-Local Constraints: Uniqueness

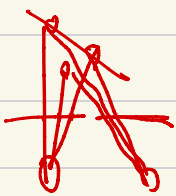
- Each point in one image should match at most one point in the other image.
- **When might this not be true?**



Non-Local Constraints: Smoothness

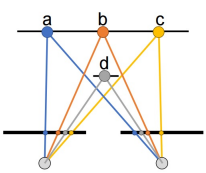
- Neighboring points should have similar disparity values, i.e., the disparities should be smooth.
- **When might this not be true?**





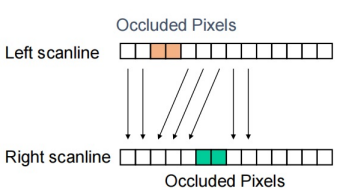
Non-Local Constraints: Ordering

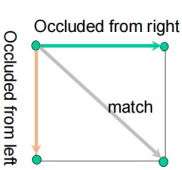
- Corresponding points should appear in the same order.
- **When might this not be true?**



Stereo Matching with Dynamic Programming.

Correspondence Types in the Disparity Space Image

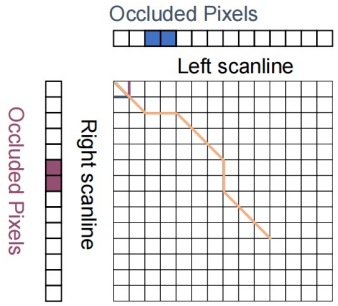




- Matched – add cost of match (small if intensities agree)
- Occluded – add cost of no match (large cost)
- Denote the matching cost as *e(i, j)*, *the total cost is* $C(i,j) = \sum_{(i,j)\in Path} e(i,j)$

- Scan the grids to compute the optimal cost for each node given its upper-left neighbors.
- The optimal structure is as follows:

$C(i,j) = min([$
$C(i-1,j-1) + e(i,j),$
$C(i-1,j) + occlusionConstant,$
$C(i,j-1) + occlusionConstant,])$




```
for(i=1;i≤ N;i++){
   for(j=1;j≤ M;j++){
      min1 = C(i-1,j-1)+e(z1,i ,z2,j);
      min2 = C(i-1,j)+Occlusion;
      min3 = C(i,j-1)+Occlusion;
      C(i,j) = cmin = min(min1,min2,min3);
      if(min1==cmin) B(i,j) = 1;
      if(min2==cmin) B(i,j) = 2;
      if(min3==cmin) B(i,j) = 3;
   }}
```

The complexity is $O(MN)$.

We can leverage the matrix **B** to recover the path via backtracing.

Cox, Hingorani, Rao, Maggs, "A Maximum Likelihood Stereo Algorithm," 1996

Seam Carving for Resizing: 图像裁剪，用动态规划求最小的路径 → dp.

Stereo Matching via Global Optimization

- The dynamic programming only considers optimization in each scan-line.
- We can consider the output disparity image as a whole.

$$E(d) = E_d(d) + \lambda E_s(d)$$

Data term — Smoothness term

Want each pixel to find a good match in the other image (block matching result)

Adjacent pixels should (usually) move about the same amount (smoothness function)

$$E(d) = E_d(d) + \lambda E_s(d)$$

Data term — Smoothness term

Dynamic Programming: optimization on 1D

$$E_d(d) = \sum_{(x,y)\in I} C(x,y,d(x,y))$$

SSD between windows centered at *I(x, y)* and *I(x+ d(x,y), y)*

$$E_s(d) = \sum_{(p,q)\in\mathcal{E}} V(d_p, d_q)$$

$\mathcal{E}$: set of neighboring pixels

4–connected neighborhood

8–connected neighborhood

- Energy functions of this form can be minimized using *graph cuts*
- Y. Boykov, O. Veksler, and R. Zabih, Fast Approximate Energy Minimization via Graph Cuts, PAMI 2001

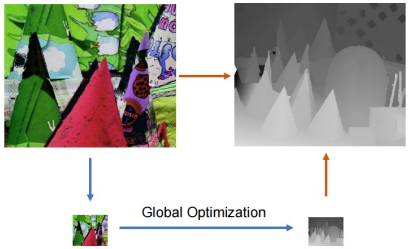
Global Optimization with Joint Bilateral Upsampling

For significant speedup

- Downsample input image
- *Estimate the depth image*
- Upsample the depth image

How to upsample the results?

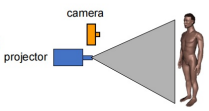
- Reuse the edge information in the original input image



Activate stereo ✓

Active Stereo with Structured Light

- Project "structured" light patterns onto the object
  - Simplifies the correspondence problem
  - Use one camera and one projector

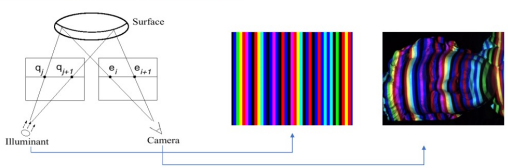


→ 打一些带结构的光，增强辨识度，提升匹配精度

L. Zhang, B. Curless, and S. M. Seitz. Rapid Shape Acquisition Using Color Structured Light and Multi-pass Dynamic Programming. *3DPVT* 2002

Active Stereo with Structured Light

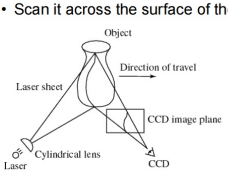
- Project "structured" light patterns onto the object
  - Simplifies the correspondence problem
  - Use one camera and one projector

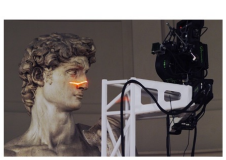


http://en.wikipedia.org/wiki/Structured-light_3D_scanner

Laser Scanning

- Optical triangulation: structured light scanning with high precision
  - Project a single stripe of laser light
  - Scan it across the surface of the object





Digital Michelangelo Project
http://graphics.stanford.edu/projects/mich/

09 Structure from Motion. → 从无序2D集对未知的相机参数、3D位置.

Structure from Motion

- **Input:** many images captured by unknown cameras
- **Output**: camera parameters (motion) and a 3D model of the scene (structure)

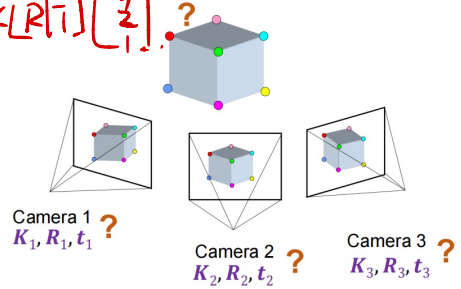
VS.

Two-View Stereo

- **Input**: a stereo pair with known camera matrices (calibrated)
- **Output**: a dense depth map

Structure from Motion vs Calibration & Triangulation

- Calibration 标定 $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K[R|t]\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$?
  - **Input**: point pairs from 3D to 2D
  - **Output**: camera parameters
- Triangulation 三角测量.
  - **Input**: camera parameters, point pairs from 2D to 2D
  - **Output**: 3D points
- Structure from motion
  - **Input:** point pairs from 2D to 2D
  - **Output:** 3D points and camera parameters



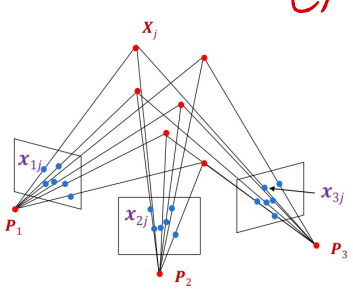
Structure from Motion: Problem formulation (问题表述).

- **Input**: m images of n fixed 3D points (ignoring visibility) such that

$$x_{ij} \cong P_i X_j \quad (i = 1, \ldots, m;\ j = 1, \ldots, n)$$

- **Output**: estimate m projection matrices P_i and n 3D points X_j from the mn correspondences x_{ij}

m个投影矩阵 P_i.



Sfm 问题的不确定性.

The Ambiguity of Structure from Motion

- Scale the scene by a factor of k and, scale the camera matrices by $1/k$, then the projections of the scene remain exactly the same:

$$x \cong PX = \left(\frac{1}{k}P\right)(kX)$$

- Without a reference measurement, it is impossible to recover the absolute scale of the scene!

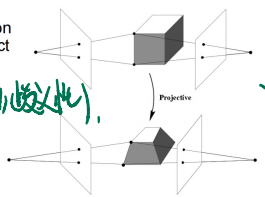
- In general, if we transform the scene using a transformation Q and apply the inverse transformation to the camera matrices, then the image observations do not change:

$$x \cong PX = (PQ^{-1})(QX)$$

- With no constraints on the camera calibration matrices or on the scene, we can reconstruct up to a *projective* ambiguity:

$$x \cong PX = (PQ^{-1})(QX)$$

where Q is a general full-rank 4×4 matrix

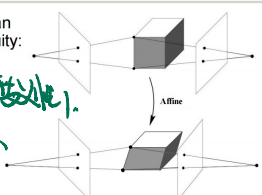


- If we impose parallelism constraints, we can get a reconstruction up to an affine ambiguity:

$$x \cong PX = (PQ^{-1})(QX)$$

$$Q_A = \begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}$$

where A is a 3×3 full-rank matrix, t is a 3×1 translation vector

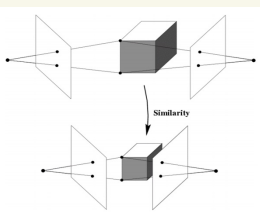


- A reconstruction that obeys orthogonality constraints on camera parameters and/or scene (a.k.a. metric reconstruction)

$$x \cong PX = (PQ^{-1})(QX)$$

$$Q_S = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix}$$

where R is a 3×3 rotation matrix, t is a 3×1 translation vector



Affine Camera

- Under orthographic projection, the camera matrix can be written as follows.
- We call this camera as affine camera, with which we will perform SFM.

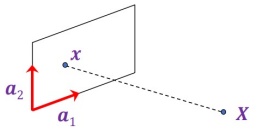
$$P = \begin{bmatrix} A_{2D} & t_{2D} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{3D} & t_{3D} \\ 0 & 1 \end{bmatrix} \Rightarrow P = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3x3 affine matrix in 2D | 3x4 orthographic projection | 4x4 affine matrix in 3D

Affine Camera

- The projection matrix is a 3D-to-2D linear mapping plus translation:

$$\boldsymbol{P} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$$



$\boldsymbol{a}_1, \boldsymbol{a}_2$: rows of projection matrix

- In non-homogeneous coordinates:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{t}$$

$\boldsymbol{t}$ is the projection of the world origin

Affine Structure from Motion

- **Given**: m images of n fixed 3D points such that

$$\boldsymbol{x}_{ij} = \boldsymbol{A}_i \boldsymbol{X}_j + \boldsymbol{t}_i, \quad i = 1, \dots, m, \; j = 1, \dots, n$$

- **Problem**: use the mn correspondences $\boldsymbol{x}_{ij}$ to estimate m projection matrices $\boldsymbol{A}_i$ and translation vectors $\boldsymbol{t}_i$, and n points $\boldsymbol{X}_j$
- The reconstruction is defined up to an arbitrary *affine* transformation $\boldsymbol{Q}$, which has 12 degrees of freedom:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \rightarrow \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \boldsymbol{Q}^{-1}, \quad \begin{pmatrix} \boldsymbol{X}_j \\ 1 \end{pmatrix} \rightarrow \boldsymbol{Q} \begin{pmatrix} \boldsymbol{X}_j \\ 1 \end{pmatrix}$$

ASFM 的重建结果只能确定到 一个任意仿射变换 Q 的范围内。 P21

这意味着，即使我们得到了相机参数和三维结构的估计值，我们仍然无法确定场景的真实尺度、方向和形状。

仿射变换 Q 有 12 个自由度，包括： P21

- 3 个平移参数
- 3 个旋转参数
- 3 个缩放参数
- 3 个剪切参数

因此，ASFM 的重建结果存在以下不确定性：

- **尺度不确定性**：无法确定场景的真实大小。 P10
- **方向不确定性**：无法确定场景的真实方向。
- **形状不确定性**：无法确定场景的真实形状。

Affine Structure from Motion

- **Given**: m images of n fixed 3D points such that

$$\boldsymbol{x}_{ij} = \boldsymbol{A}_i \boldsymbol{X}_j + \boldsymbol{t}_i, \quad i = 1, \dots, m, \; j = 1, \dots, n$$

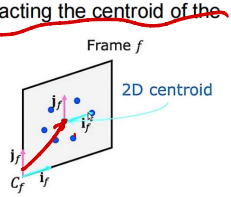
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{t}$$

- How many knowns and unknowns for m images and n points?
  - $2mn$ knowns ($\boldsymbol{x}_{ij}$) and $8m + 3n$ unknowns ($\boldsymbol{A}_i$, $\boldsymbol{t}_i$, and $\boldsymbol{X}_j$)
  - To be able to solve this problem, we must have $2mn \geq 8m + 3n - 12$, since the affine ambiguity takes away 12 degree of freedom.
  - E.g., for two views, we need four point correspondences

Affine Structure from Motion

- Simplifying by centering: center the data by subtracting the centroid of the image points in each view:

$$\begin{aligned} \hat{\boldsymbol{x}}_{ij} &= \boldsymbol{x}_{ij} - \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{x}_{ik} \\ &= \boldsymbol{A}_i \boldsymbol{X}_j + \boldsymbol{t}_i - \frac{1}{n}\sum_{k=1}^{n} (\boldsymbol{A}_i \boldsymbol{X}_k + \boldsymbol{t}_i) \\ &= \boldsymbol{A}_i \left(\boldsymbol{X}_j - \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{X}_k \right) \\ &= \boldsymbol{A}_i \hat{\boldsymbol{X}}_j \end{aligned}$$



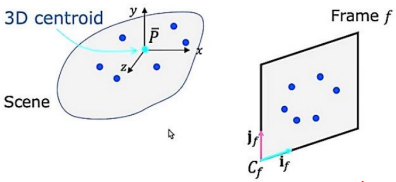
Set the origin to mean of points for each frame

Affine Structure from Motion

- After centering, each normalized 2D point $\hat{x}_{ij}$ is related to the 3D point by

$$\hat{x}_{ij} = A_i X_j$$

- Get rid of the need to center the 3D data (and the translation ambiguity) by defining the origin of the world coordinate system as the centroid of the 3D points.



规定三维坐标系原点位于中心

Affine Structure from Motion

- Let's create a $2m \times n$ measurement matrix D according to $\hat{x}_{ij} = A_i X_j$

n points

m cameras, $\hat{x}_{ij}$ is 2D vector. The row is $2m$.

$$\begin{bmatrix} \hat{x}_{11} & \hat{x}_{12} & \cdots & \hat{x}_{1n} \\ \hat{x}_{21} & \hat{x}_{22} & \cdots & \hat{x}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{x}_{m1} & \hat{x}_{m2} & \cdots & \hat{x}_{mn} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} \begin{bmatrix} X_1 & X_2 & \cdots & X_n \end{bmatrix}$$

$$D_{2m\times n} \qquad M_{2m\times 3} \qquad S_{3\times n}$$

- Decompose M, S from D according to the matrix rank: $D = MS$
- The rank of D is at most 3, why?

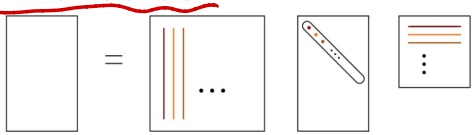
$A=BC$, $\text{rank}\, A \le \min(\text{rank}\, B, \text{rank}\, C)$

Factorizing the Measurement Matrix

- Decompose M, S from D according to the matrix rank.
- SVD decomposition: $M_{m\times n} = U_{m\times m} \cdot \Sigma_{m\times n} \cdot V^T_{n\times n}$

Proof by construction:

$$Mv_i = \sigma_i u_i$$
$$MV = U\Sigma$$
$$M = U\Sigma V^T$$



How to find U and V

$$M^T M = V\Sigma^2 V^T$$
$$MM^T = U\Sigma^2 U^T$$

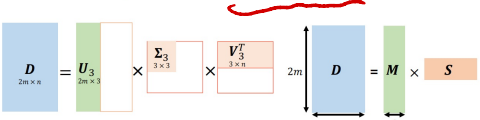
$\min_{M_k} \|M - M_k\|_F^2$, where k is the rank of M_k

Low Rank Approximation: $M = \sum_i \sigma_i u_i v_i^T$, keep the first k components after SVD

https://en.wikipedia.org/wiki/Low-rank_approximation

Factorizing the Measurement Matrix

- Perform SVD decomposition; Keep the top 3 singular values.
- What to do about Σ_3? One solution: $M = U_3 \Sigma_3^{\frac{1}{2}}, S = \Sigma_3^{\frac{1}{2}} V_3^T$.



Eliminating the Affine Ambiguity

- We have obtained one solution.
- However, there is still a ambiguity.
- We add constraints such that each camera matrix $A_i Q$ represents a orthographic projection, i.e., has orthonormal axes (rows) .

$$D = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix}_{2m \times 3} [X_1 \quad X_2 \quad \cdots \quad X_n]_{3 \times n}$$

$$\Downarrow$$

$$D = \begin{bmatrix} A_1 Q \\ A_2 Q \\ \vdots \\ A_m Q \end{bmatrix} [Q^{-1} X_1 \quad Q^{-1} X_2 \quad \cdots \quad Q^{-1} X_n]$$

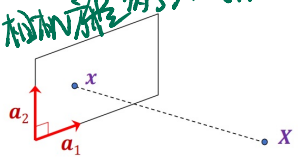
Eliminating the Affine Ambiguity

- Let a_1 and a_2 be the rows of a 2×3 orthographic projection matrix. Then

$$a_1 \cdot a_2 = 0, \quad \|a_1\|^2 = \|a_2\|^2 = 1$$

- This translates into $3m$ constraints on the 9 entries of Q:

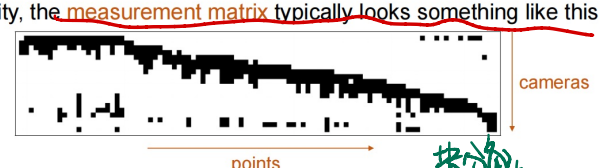
$$(A_i Q)(A_i Q)^T = A_i (Q Q^T) A_i^T = I_{2\times 2}, \qquad i = 1, \ldots, m$$



- Are the constraints linear? No!
- By defining $L = QQ^T$, we can solve L with least squares;
- Then we can recover Q from L by Cholesky decomposition.
- Update camera matrics M to MQ, update 3D point matrix S to $Q^{-1}S$

Dealing with Missing Data

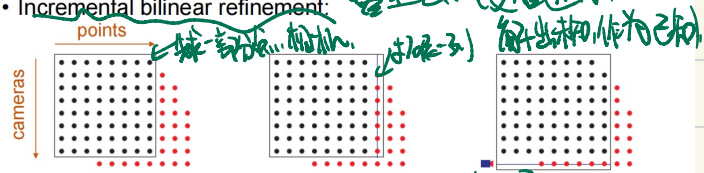
- So far, we have assumed that all points are visible in all views
- In reality, the measurement matrix typically looks something like this:



- Possible solution: decompose matrix into dense sub-blocks, factorize each sub-block, and fuse the results
- Unfortunately, finding dense maximal sub-blocks of the matrix is NP-complete (equivalent to finding maximal cliques in a graph)

Dealing with Missing Data

- Incremental bilinear refinement:



Perform factorization on a dense sub-block

Solve for a new 3D point visible by at least two known cameras – triangulation

Solve for a new camera that sees at least three known 3D points – calibration

增量双线性细化方法

增量双线性细化方法通过以下步骤处理缺失数据：

1. **寻找密集子块**：将测量矩阵 D 分解成多个密集子块，这些子块包含了尽可能多的观测值。 P32
2. **对每个子块进行分解**：对每个密集子块进行 SVD 分解，得到相机参数和三维结构的估计值。 P32
3. **三角测量**：使用三角测量算法，根据至少两个已知相机视图中的对应点，计算新点的三维坐标。 P33
4. **标定**：使用标定算法，根据至少三个已知三维点的投影，计算新相机的参数。 P33
5. **迭代更新**：重复上述步骤，逐步添加新的相机和点，并更新整个重建结果。

增量双线性细化方法的优点

- **可以处理缺失数据**：即使存在缺失值，仍然可以进行三维重建。
- **可以逐步添加新的数据**：可以根据需要逐步添加新的图像和点，从而提高重建的精度。

增量双线性细化方法的局限性

- **计算量较大**：需要对多个子块进行分解，计算量较大。
- **可能存在误差累积**：逐步添加新数据可能会导致误差累积。

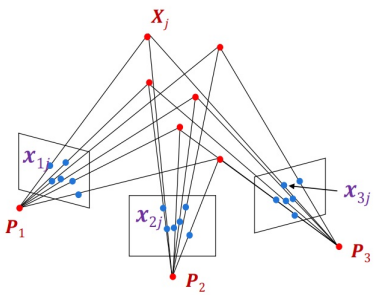
Projective Structure from Motion.

Projective Structure from Motion

- **Given**: m images of n fixed 3D points such that (ignoring visibility):

$$x_{ij} \cong P_i X_j, \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

- **Problem**: estimate m projection matrices P_i and n 3D points X_j from the mn correspondences x_{ij}.



Projective Structure from Motion

- **Given**: m images of n fixed 3D points such that (ignoring visibility):

$$x_{ij} \cong P_i X_j, \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

- **Problem**: estimate m projection matrices P_i and n 3D points X_j from the mn correspondences x_{ij}.
- With no calibration information, cameras and points can only be recovered up to a 4×4 projective transformation Q:

$$X \rightarrow QX, \qquad P \rightarrow PQ^{-1}$$

- We can solve for structure and motion when $2mn \geq 11m + 3n - 15$
- For two cameras, at least 7 points are needed

Projective SFM: Two-Camera Case

1. Estimate fundamental matrix F between the two views with normalized eight-point algorithm.
2. Set the first camera matrix to $[I \mid 0]$.
3. Then the second camera matrix is given by $[A \mid t]$ where t is the epipole ($F^T t = 0$) and $A = -[t_\times]F$.

- In practice, SFM pipelines use guesses of intrinsic parameters and the five-point algorithm.

Cameras from the Fundamental Matrix

- Define $P_1 = [I \mid 0], P_2 = [A \mid b]$, we get the fundamental matrix: $F = [b_\times]A$.
- One possible option is $A = -[t_\times]F$ and $b = t$, where t is the epipole.
- Proof:

$$[b_\times]A = -[t_\times][t_\times]F = -(tt^T - \|t\|^2 I)F = -tt^T F + \|t\|^2 F = 0 + 1 \cdot F = F$$

$$[t_\times] = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}$$

Therefore, given the fundamental matrix, we can set the first camera matrix to $[I \mid 0]$, and the second camera matrix is given by $[-[t_\times]F \mid t]$, where t is the epipole.

Cameras from the Fundamental Matrix

- Define the camera matrices as: $P_i = [A_i \mid b_i]$, then $x_{ij} = P_i X_j$.
- Because of the projective ambiguity, we can always apply a projective transformation H such that: $P_1 H^{-1} = [I \mid 0], P_2 H^{-1} = [A \mid b]$.
- Define $\widehat{P} = PH^{-1}, \widehat{X} = HX$, then $x = PX = PH^{-1}HX = \widehat{P}\widehat{X}$.
- Define x and x' the corresponding observations on two cameras, then $x \cong [I \mid 0]\widehat{X},\ x' \cong [A \mid b]\widehat{X} = Ax + b$.
- Ax, x' and b are linear dependent: $x'^T(b \times Ax) = x'^T[b_\times]Ax = 0$.
- We get another definition of fundamental matrix: $F = [b_\times]A$.
- The fundamental matrix is invariant to the projective transformation H.

Cameras from the Fundamental Matrix

- Define $P_1 = [I \mid 0], P_2 = [A \mid b]$, we get the fundamental matrix: $F = [b_\times]A$.
- One possible option is $A = -[t_\times]F$ and $b = t$, where t is the epipole.
- Proof:

$$[b_\times]A = -[t_\times][t_\times]F = -(tt^T - \|t\|^2 I)F = -tt^T F + \|t\|^2 F = 0 + 1 \cdot F = F$$

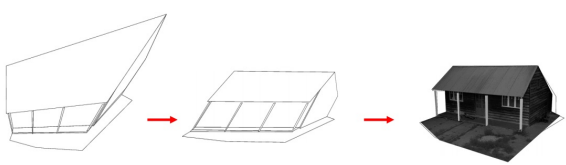
$$[t_\times] = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}$$

Therefore, given the fundamental matrix, we can set the first camera matrix to $[I \mid 0]$, and the second camera matrix is given by $[-[t_\times]F \mid t]$, where t is the epipole.

→ 通过假设增加约束来对地进行重建。

Self-calibration

- Self-calibration is the problem of recovering the metric reconstruction from the perspective (or affine) reconstruction
- We can self-calibrate the camera by making some assumptions about the cameras.

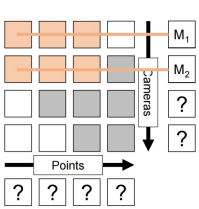


Incremental Structure from Motion. (增量法)

Incremental SFM

1. Initialize motion $M_i = [R_i, t_i]$ with fundamental matrix

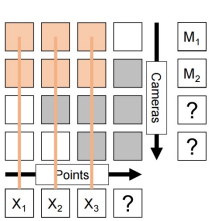
Key idea: incrementally add cameras and points



Incremental SFM

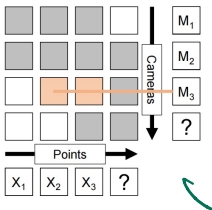
1. Initialize motion $M_i = [R_i, t_i]$ with fundamental matrix
2. Initialize structure X_j with triangulation

How could we add another camera?



Incremental SFM

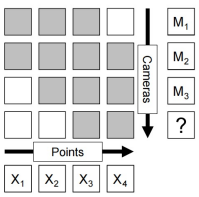
1. Solve for camera matrix using visible, known points using calibration.



Incremental SFM

Big problem: don't ever jointly consider all the 3D points and camera.

Leads to final step, called bundle adjustment.



Bundle Adjustment

- Non-linear method for refining the structure and motion.
- Minimize the reprojection error

$$\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}\, d\left(\boldsymbol{x}_{ij} - \text{proj}(\boldsymbol{P}_i \boldsymbol{X}_j)\right)^2$$

visibility flag: is point j visible in view i?

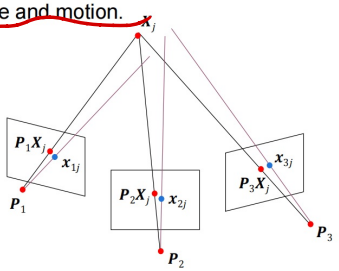


Photo Tourism: SFM Pipeline.

Feature Detection ⇒ 找到点对 (Feature Matching).

↓

用 RANSAC 得到 fundamental matrix.

↓

Incremental SFM

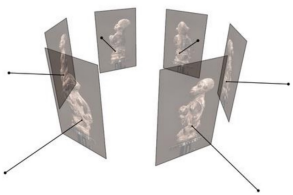
Incremental SFM

- Pick a pair of images with lots of inliers (and preferably, EXIF data)
  - Initialize intrinsic parameters (focal length, principal point) from EXIF
  - Estimate extrinsic parameters (R and t) using five-point algorithm
  - Use triangulation to initialize model points
- While remaining images exist
  - Find an image with many feature matches with images in the model
  - Run RANSAC on feature matches to register new image to model
  - Triangulate new points
  - Perform bundle adjustment to re-optimize everything periodically
  - Optionally, align with GPS from EXIF data or ground control points

10 Multi-View Stereo (MVS).

Multi-View Stereo

- **Input**: several images of the same object or scene with calibrated cameras
- **Output**: compute a representation of the corresponding 3D shape

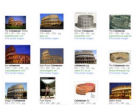


Multi-View Stereo

Two-View Stereo

Multi-View Stereo

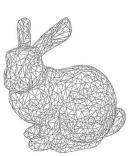
- **Input**: several images of the same object or scene with calibrated cameras
  - Arbitrary number of images (from two to thousands)
  - Arbitrary camera positions (camera network or video)
  - Often calibrate cameras with structure from motion or special devices



Multi-View Stereo

- **Output**: compute a representation of the corresponding 3D shape

Voxel



Mesh

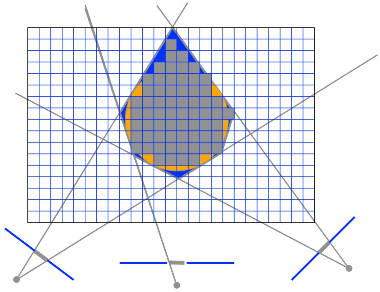
Point Cloud

Multi-View Stereo 优势: ① 捕获更多点. ② 降低重建误差.

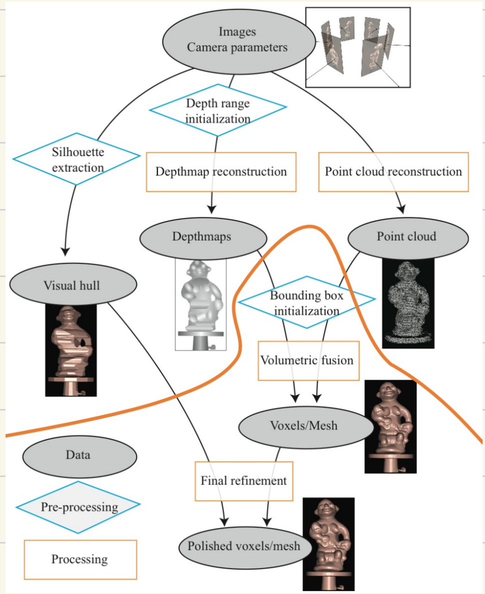
Visual Hull based MVS.

Visual Hull: 2D Example

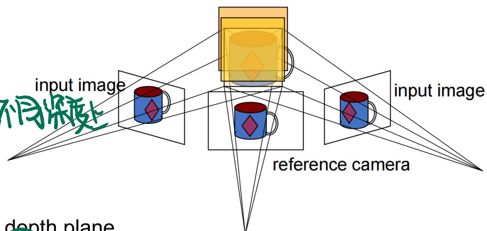
三维中对多个视角求交，重建表面体素、



Color a voxel as black if it is on silhouette in every image



Plane-Sweep Stereo

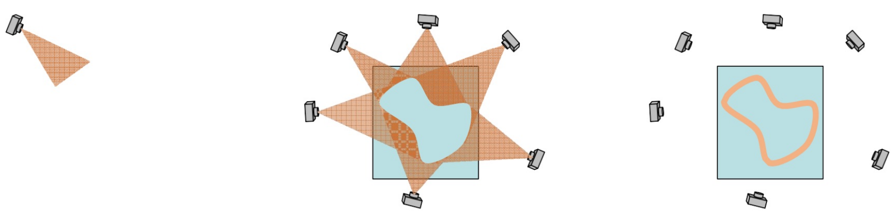


ref 投影到不同深度上

- For each depth plane
  - Compute homographies projecting each image onto that depth plane
  - For each pixel in the composite image stack, compute the variance
- For each pixel, select the depth that gives the lowest variance

→ 算各个体素的方差和（input images）若接近实际深度，则不同图的方差很小，对每个像素取最小方差对应的深度

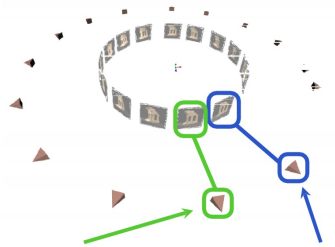
Depth based MVS.



Compute depth-maps using neighboring views | Merge depth-maps into single volume | Extract 3d surface from this volume

Compute Depth Maps

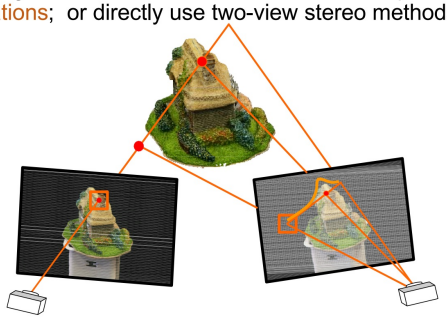
- Occlusion effects are small between neighboring views



Select a reference image and its neighboring images

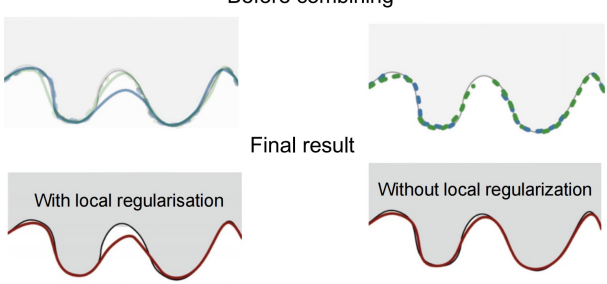
Compute Depth Maps

- Use the matching score to find the best match for each pixel without strong local regularizations; or directly use two-view stereo methods



Effect of Early Local Regularization

Before combining



It is preferable to perform optimization in 3D space after merging depth maps.

Patch based MVS.

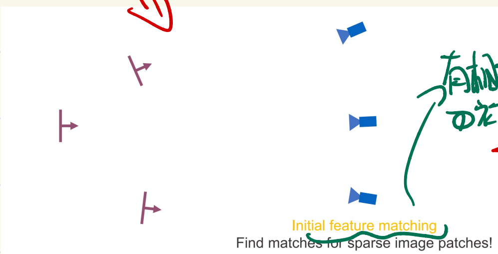
切平面近似.

Patch { position (x,y,z)
normal (nx,ny,nz),
extent (radius)

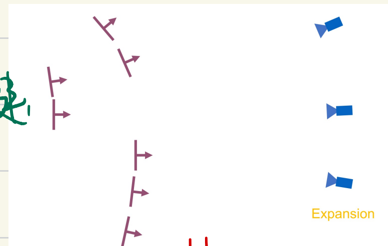
Patch-based MVS

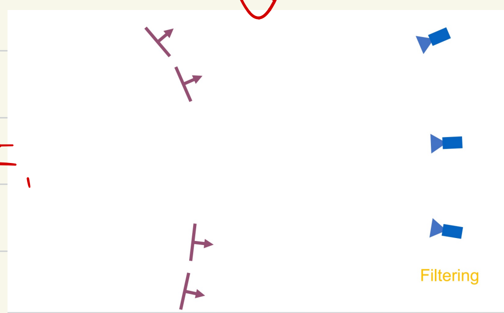
- Dense 3D reconstruction requires dense correspondence!
- Key idea: patch-based reconstruction + iterative expansion and filtering

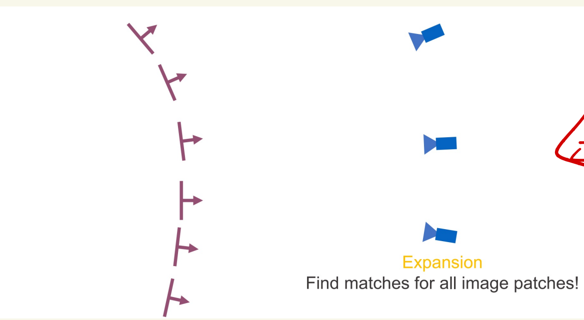
1. Feature detection; 2. Feature matching; 3. Patch expansion and filtering



有相机内参，可以方便地延续.







Patch 优势: ① Flexible to represent both objects & scenes
② Extracts pure 3d data directly.

Patch Definition

- Patch p is defined by
  - Position $c(p)$
  - Normal $n(p)$
  - Visible images $V(p)$ ← 判断是否可见
  - The extent is set so that p is roughly 9x9 pixels in $V(p)$

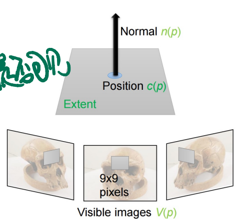
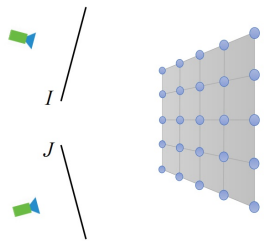


Photo Consistency

- Photo-consistency $N(I, J, p)$ of p between two images I and J



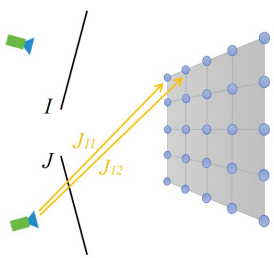
$$N(I,J,p)=\frac{\sum(I_{xy}-\overline{I_{xy}})\cdot(J_{xy}-\overline{J_{xy}})}{\sqrt{(I_{xy}-\overline{I_{xy}})^2}\sqrt{(J_{xy}-\overline{J_{xy}})^2}}$$

Photo-consistency $N(p)$ of p with visible images $V(p) = \{I_1, I_2, \ldots, I_n\}$

$$N(p)=\frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n}N(I_i,I_j,p)}{(n+1)n/2}$$

Photo Consistency

- Photo-consistency $N(I, J, p)$ of p between two images I and J



I_{xy}: pixel color in image I
J_{xy}: pixel color in image J

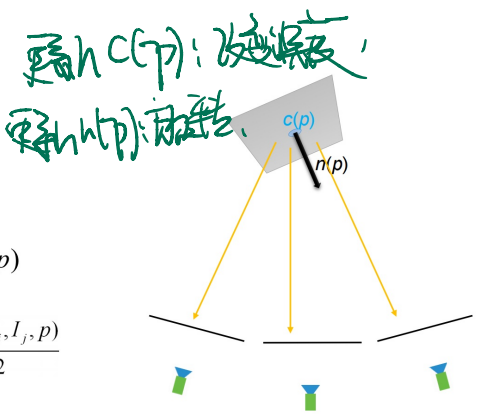
$$N(I,J,p)=\frac{\sum(I_{xy}-\overline{I_{xy}})\cdot(J_{xy}-\overline{J_{xy}})}{\sqrt{(I_{xy}-\overline{I_{xy}})^2}\sqrt{(J_{xy}-\overline{J_{xy}})^2}}$$

Reconstruct Patch p

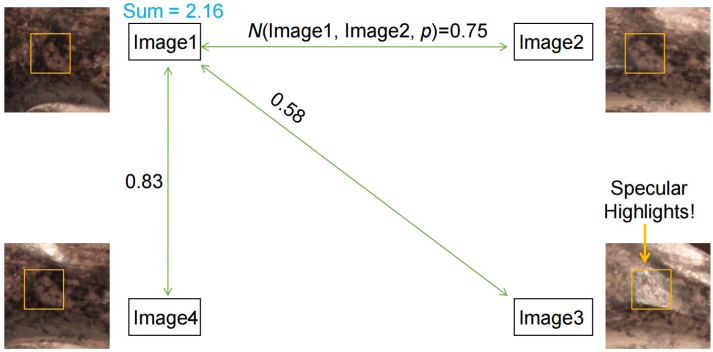
- Given initial estimates of
  - Position $c(p)$
  - Normal $n(p)$
  - Visible images $V(p)$
- Refine $c(p)$ and $n(p)$

$$\{c(p),n(p)\}=\arg\max_{\{c(p),n(p)\}}N(p)$$

$$N(p)=\frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n}N(I_i,I_j,p)}{(n+1)n/2}$$

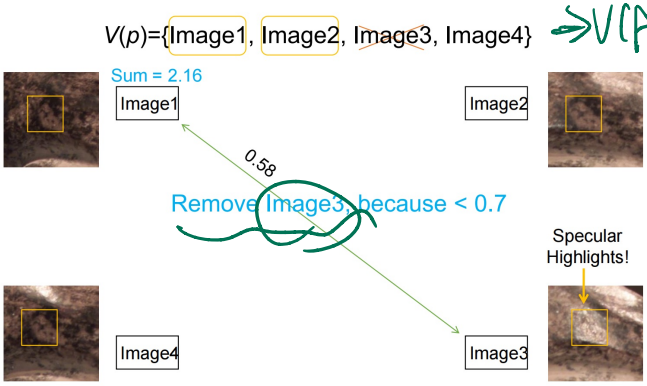


Update $V(p)$



Update $V(p)$

Update $V(p)$



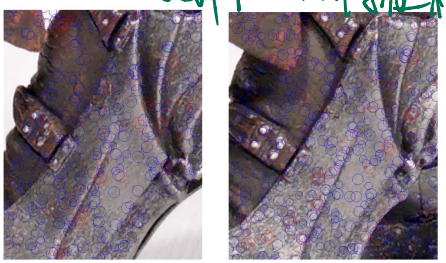
Pipeline:

Feature Detection.

Feature Detection

- Extract local maxima of Harris corner detector (corners) and Difference of Gaussian (blobs)

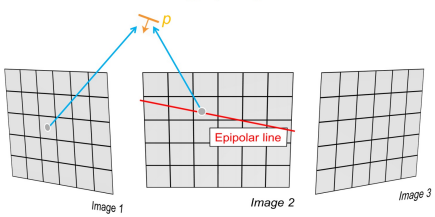
对每个patch单独检测



Feature Matching

Initial Feature Matching

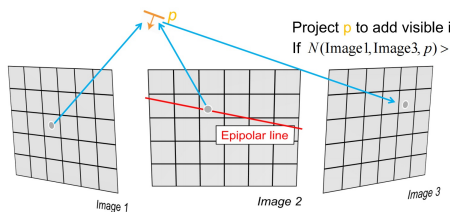
c(*p*): triangulation
n(*p*): parallel to *Image1*
V(*p*): {*Image1*, *Image2*}



Initial Feature Matching

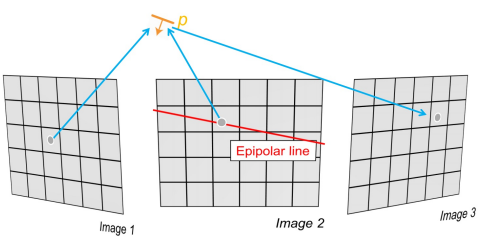
c(*p*): triangulation
n(*p*): parallel to *Image1*
V(*p*): {*Image1*, *Image2*} *Image3*}

Project p to add visible images
If $N(\text{Image1}, \text{Image3}, p) > 0.5$



Initial Feature Matching

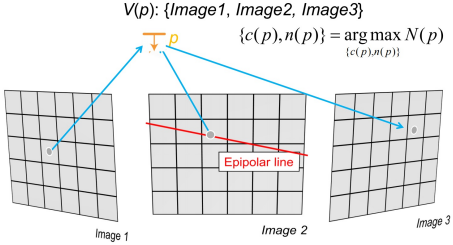
c(*p*): triangulation
n(*p*): parallel to *Image1*
V(*p*): {*Image1*, *Image2*, *Image3*}

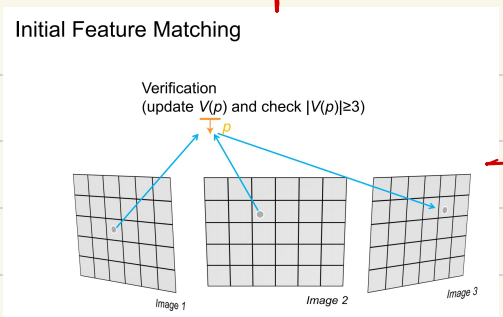


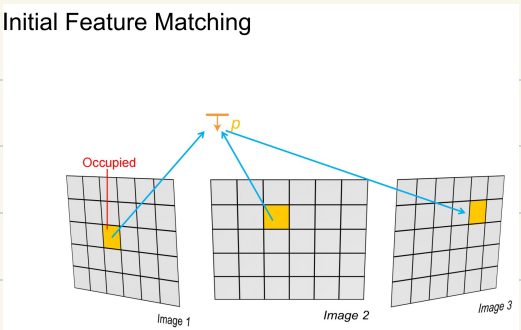
Initial Feature Matching

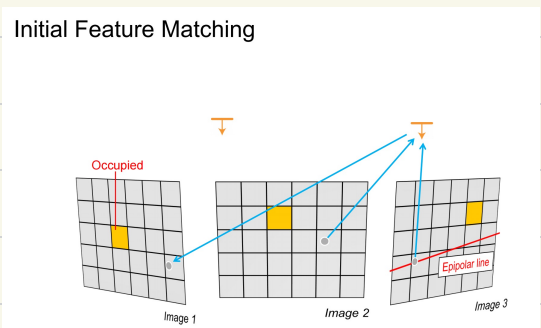
c(*p*): refine
n(*p*): refine
V(*p*): {*Image1*, *Image2*, *Image3*}

$$\{c(p), n(p)\} = \arg\max_{\{c(p), n(p)\}} N(p)$$

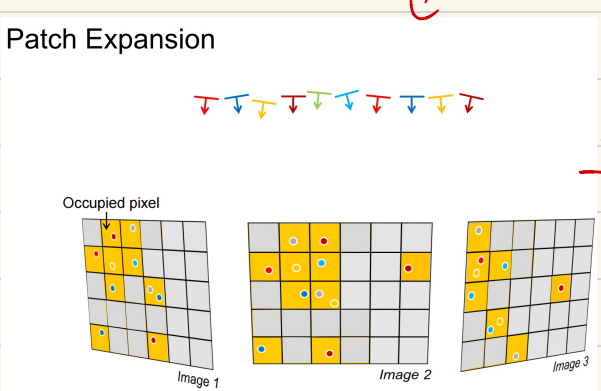


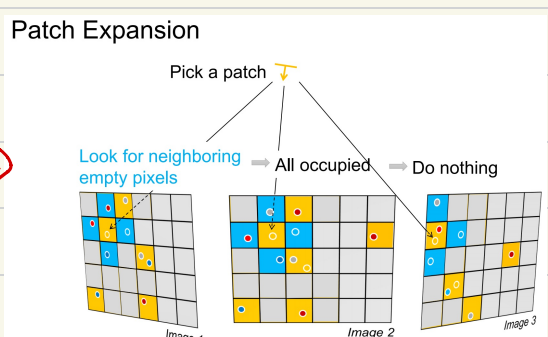






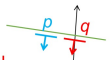
Patch Expansion.



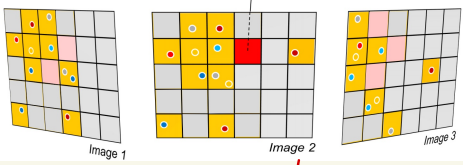


Patch Expansion

Reconstruct a patch visible in an empty pixel



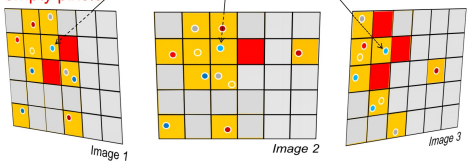
$c(q)$: {tangent plane of p intersects w/ ray}
$n(q)$:
$V(q)$:



Patch Expansion

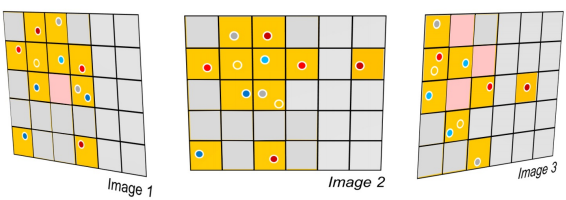
Pick a patch p

Identify neighboring empty pixels



Patch Expansion

Repeat
- for every patch
- for every neighboring empty pixel

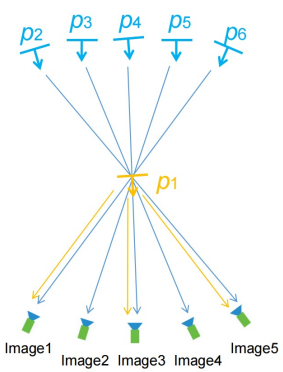


Patch Filtering

- Visibility consistency

Filter out p_1 if

$$|V(p_1)|\,N(p_1) < \sum_{i=2}^{6} N(p_i)$$



Summary of Patch-based MVS

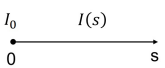
- **Pros**
  - Flexible
  - Extracts pure 3D data
  - Effective for scenes and objects
  - PMVS is a black box

- **Cons**
  - Surfaces must be Lambertian and well-textured
  - The running time is relatively slow

Neural Radiance Field (NERF).

Volume Rendering: Absorption Only

$$\frac{dI(s)}{ds} = -\tau(s)I(s)$$



$$I(s) = I_0 \exp\left(-\int_0^s \tau(t)\,dt\right)$$

$$T(s) = \exp\left(-\int_0^s \tau(t)\,dt\right) \rightarrow \text{Transparency}$$

$$\alpha = 1 - T(l) = 1 - \exp\left(-\int_0^l \tau(t)\,dt\right) \rightarrow \text{Opacity}$$

(不透明度)

Volume Rendering

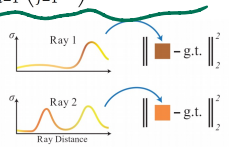
$\alpha_i = 1 - \exp(-\tau(s)\cdot s) \rightarrow$ 当 $s \rightarrow 0$.

Here σ_i is the volume density, and $\alpha_i = 1 - \exp(-\delta_i \tau_i)$ is the opacity.

NeRF from the viewpoint of color blending: $\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} \left(\prod_{j=1}^{i-1} \alpha_j \right) (1 - \alpha_i) \mathbf{c}_i$

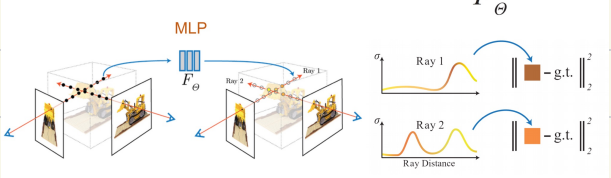
$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$



Neural Radiance Field

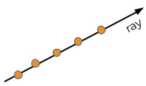
$$(x, y, z, \theta, \phi) \rightarrow F_\Theta \rightarrow (RGB\sigma)$$



Gaussian Splatting

NeRF

- Ray marching



$$C = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i$$

$$\alpha_i = (1 - \exp(-\sigma_i \delta_i)) \text{ and } T_i = \prod_{j=1}^{i-1} (1 - \alpha_i)$$

GS

- Splatting 3D Gaussians to 2D Gaussians

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

11 Photometric Stereo

Photometric Stereo

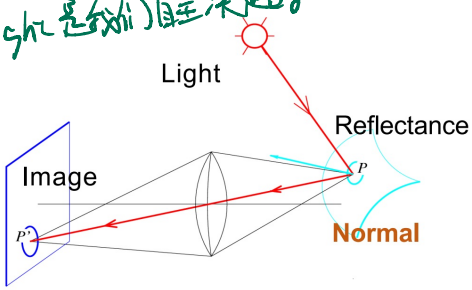
- Estimating the surface normals of objects by observing that object under different lighting conditions.

- Why surface normals?
  - Surface normals are directly related to image colors
  - We can recover 3D shapes given surface normals

Normal map

Image Intensity

- Image Intensity = f (Light, Surface Reflectance, Surface Normal)



- The **goal** is to recover normals from image intensities, which is under-constrained
- We need make assumptions or add constrains

Surface Reflectance

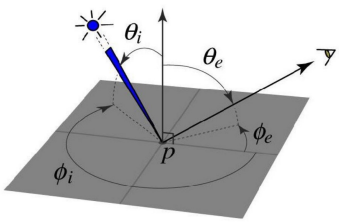
- Specular reflection: light is reflected about the surface normal.
- Diffuse reflection: light scatters equally in all directions.

Surface Reflectance: BRDF

- BRDF is short for Bidirectional Reflectance Distribution Function
- BRDF describes how bright a surface appears when viewed from one direction when light falls on it from another direction

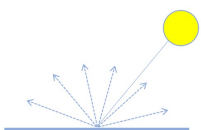
$$BRDF = f(\theta_i, \phi_i, \theta_e, \phi_e)$$

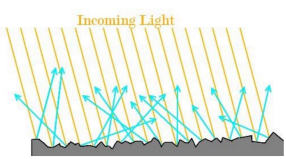
- For simplicity, we mainly concern about the diffuse reflection



Diffuse Reflection

- Light scatters equally in all directions
  - E.g., brick, matte plastic, rough wood
- For a fixed incidence angle, the BRDF is constant.
- This happens because of microfacets that scatter incoming light randomly.

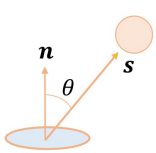




Diffuse Reflection

- Lambert's Law for diffuse reflection

$$I = \rho(\boldsymbol{n} \cdot \boldsymbol{s}) = \rho\|\boldsymbol{s}\|cos\theta$$



- I: reflected intensity, or total power leaving the surface per unit area
- ρ: albedo, the fraction of incident irradiance reflected by the surface
- $\boldsymbol{s}$: direction of the light and its magnitude is proportional to its intensity
- $\boldsymbol{n}$: unit surface normal

Surface Gradient and Normal

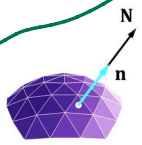
Let $z = f(x, y)$ represent a 3D surface

Surface gradient:

$$\left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}\right) = (p, q)$$

Surface normal (why):

$$N = \left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}, 1\right) = (p, q, 1)$$



$z = f(x, y)$

- What is the unit surface normal $\boldsymbol{n}$?
- How many parameters needed to represent the surface normal?

可以从图像颜色求解出法向量，再从法向量重建出曲面。

Gradient Space

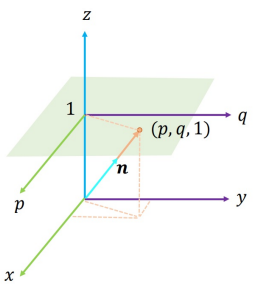
- Plane $z = 1$ is called the Gradient Space or pq Plane.
- Each point $(p,\ q)$ in the Gradient Space corresponds to a unique orientation (with normalization).

Surface normal:

$$\mathbf{n} = \frac{\mathbf{N}}{\|\mathbf{N}\|} = \frac{(p, q, 1)}{\sqrt{p^2 + q^2 + 1}}$$

Source direction:

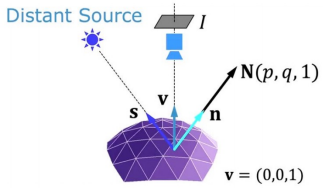
$$\mathbf{s} = \frac{\mathbf{S}}{\|\mathbf{S}\|} = \frac{(p_s, q_s, 1)}{\sqrt{{p_s}^2 + {q_s}^2 + 1}}$$



Reflectance Map

- For a given light source $\boldsymbol{s}$ and surface reflectance, the image intensity I at point (x, y) depends on the point normal $(p, q, 1)$
- The function that describes this relationship is the reflectance map:

$$I = R(p, q)$$

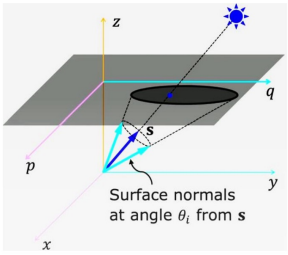


Reflectance Map for Diffuse Reflection

$$I = \rho\,(\boldsymbol{n} \cdot \boldsymbol{s})$$

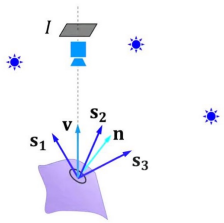
$$= \rho \frac{pp_s + qq_s + 1}{\sqrt{p^2+q^2+1}\sqrt{p_s^2+q_s^2+1}}$$

$$= R(p, q)$$



Photometric Stereo

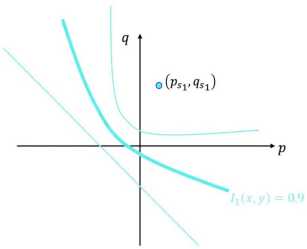
- Use multiple images under different lighting to resolve the ambiguity in surface orientation



- Direction of light source i: $\boldsymbol{s}_i = (p_{s_i}, q_{s_i})$
- Reflectance map for light source i: $R_i(p, q)$
- Image intensity produced by light source i: $I_i(x, y)$

Photometric Stereo

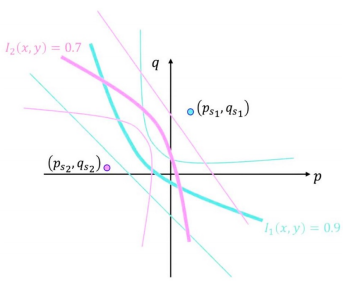
- Capture Image I_1 under light source $\boldsymbol{s}_1 = (p_{s_1}, q_{s_1})$
- For Example:
  - Let $I_1(x, y) = 0.9$



- Infinite solutions exist!

Photometric Stereo

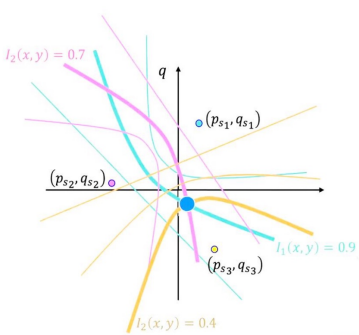
- Capture Image I_2 under light source $\boldsymbol{s_2} = (p_{s_2}, q_{s_2})$
- For Example:
  - Let $I_1(x, y) = 0.9$
  - $I_2(x, y) = 0.7$



- Two solutions exist!

Photometric Stereo

- Capture Image I_3 under light source $\boldsymbol{s_3} = (p_{s_3}, q_{s_3})$
- For Example:
  - Let $I_1(x, y) = 0.9$
  - $I_2(x, y) = 0.7$
  - $I_3(x, y) = 0.4$



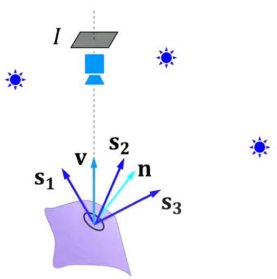
- One solutions exist!

Photometric Stereo

- General steps for photometric stereo
  - 1: Acquire K images with K known light sources.
  - 2: Using known source direction and BRDF, construct reflectance map for each source direction.
  - 3: For each pixel location (x, y), find (p, q) as the intersection of K curves, which gives the surface normal at pixel (x, y).
- What would be the smallest K?
  - It depends on the material properties.
  - For diffuse reflectance, the smallest K is 3.

Photometric Stereo

- Use multiple images under different lighting to resolve the ambiguity in surface orientation
- Direction of light source i: $\boldsymbol{s}_i = (p_{s_i}, q_{s_i})$
- Reflectance map for light source i: $R_i(p, q)$
- Image intensity produced by light source i: $I_i(x, y)$



Photometric : Lambertian Case.

Photometric Stereo: Lambertian Case

- Image intensities measured at point (x, y) under three light sources:

$$I_1 = \rho \boldsymbol{s}_1 \cdot N, \quad I_2 = \rho \boldsymbol{s}_2 \cdot N, \quad I_3 = \rho \boldsymbol{s}_3 \cdot N$$

Where $N = \begin{bmatrix} N_x \\ N_y \\ N_z \end{bmatrix}$ and $\boldsymbol{s}_i = \begin{bmatrix} s_{x_i} \\ s_{y_i} \\ s_{z_i} \end{bmatrix}$.

- We can write this in matrix form:

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \rho \begin{bmatrix} s_{x_1} & s_{y_1} & s_{z_1} \\ s_{x_2} & s_{y_2} & s_{z_2} \\ s_{x_3} & s_{y_3} & s_{z_3} \end{bmatrix} \boldsymbol{N}$$

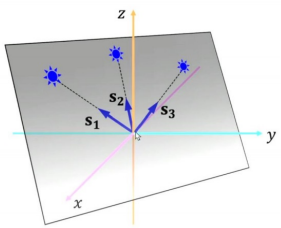
Measured Known

Photometric Stereo: Lambertian Case

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \rho \begin{bmatrix} s_{x_1} & s_{y_1} & s_{z_1} \\ s_{x_2} & s_{y_2} & s_{z_2} \\ s_{x_3} & s_{y_3} & s_{z_3} \end{bmatrix} \boldsymbol{N} \Rightarrow \boldsymbol{I} = \rho \boldsymbol{S} \boldsymbol{N} \Rightarrow \boldsymbol{N} = \boldsymbol{S}^{-1} \boldsymbol{I} \Rightarrow \widehat{\boldsymbol{N}} = \frac{\boldsymbol{N}}{\|\boldsymbol{N}\|}$$

Measured Known

- When will $\boldsymbol{S}$ not invertible?
  - One light source direction can be represented as a linear combination of the other two light sources.



Photometric Stereo: Lambertian Case

- Better results can be achieved by using more (K>3) light sources.

$$\begin{bmatrix} I_1 \\ I_2 \\ \dots \\ I_K \end{bmatrix} = \rho \begin{bmatrix} s_{x_1} & s_{y_1} & s_{z_1} \\ s_{x_2} & s_{y_2} & s_{z_2} \\ \dots & \dots & \dots \\ s_{x_K} & s_{y_K} & s_{z_K} \end{bmatrix} \boldsymbol{N} \Rightarrow \boldsymbol{I} = \rho \boldsymbol{S} \boldsymbol{N} \Rightarrow$$

Using the Least Square to solve $\boldsymbol{N}$.

最小二乘法

$$\Rightarrow \boldsymbol{N} = (\boldsymbol{S}^T \boldsymbol{S})^{-1} \boldsymbol{S}^T \boldsymbol{I} \Rightarrow \widehat{\boldsymbol{N}} = \frac{\boldsymbol{N}}{\|\boldsymbol{N}\|}$$

$$\Rightarrow \rho = \|\boldsymbol{N}\|$$

Calibration based Photometric Stereo

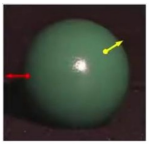
Calibration based Photometric Stereo

- The previous method is based on the Lambertian assumption
- Here we introduce one of methods for general reflectance models: use a calibration object of known size, shape (e.g. sphere) and the same reflectance as the scene objects

对与同材质球的多种光下的照片

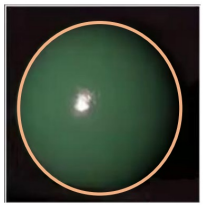
Calibration based Photometric Stereo

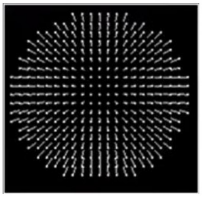
- Orientation Consistency: Points with the same surface normal produce the same set of intensities under different lighting (ignore global illuminations)



Calibration based Photometric Stereo

- **Step 1**: Capture K > 3 images under K different light sources.
- **Step 2**: Using the known size of the sphere, calculate the surface normal $(p, q, 1)$ for each point on the sphere.
- **Step 3**: Create a lookup table for the K-tuple: $(I_1, I_2, \dots, I_K) \rightarrow (p, q)$



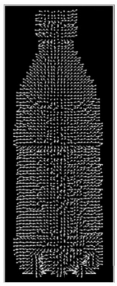


I_1	I_2	...	I_K	p	q

Calibration based Photometric Stereo

- **Step 4**: Capture K images of the scene object under the same K light sources
- **Step 5**: For each pixel in the scene, use lookup table to map $(I_1, I_2, \dots, I_K) \rightarrow (p, q)$

...



Shape from Normals

Shape from Normals: Integration-based Method

- The total differential (全微分)

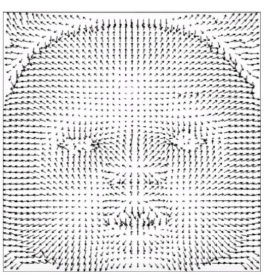
$$dz = \frac{\partial z}{\partial x}dx + \frac{\partial z}{\partial y}dy = -pdx - qdy \qquad (p,q) = \left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}\right)$$

$z(x,y)$ is the potential function, and (p,q) is a conservative field

$$\oint_{(x_0,y_0)}^{(x,y)} \frac{\partial z}{\partial x}dx + \frac{\partial z}{\partial y}dy = \oint_{(x_0,y_0)}^{(x,y)} -pdx - qdy = z(x,y) - z(x_0,y_0)$$

- $z(x,y)$ can be obtained by integration along any path from a reference point (x_0,y_0).

Shape from Normals

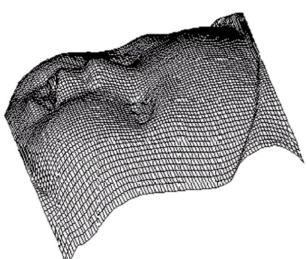


Gradient / normal map
$(p(x,y), q(x,y), 1)$

Differentiation

Integration

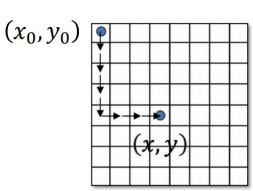
$$(p,q) = \left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}\right)$$

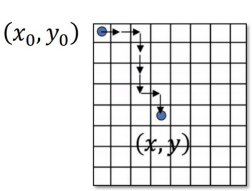


Shape / depth map
$z(x,y)$

Shape from Normals: Integration-based Method

- $z(x,y)$ can be obtained by integration along any path from a reference point (x_0,y_0)
- Define $z(x_0,y_0)$ to be any value, for example, 0





Shape from Normals: Integration-based Method

- 1. Initialize the reference depth $z(0,0) = 0$.
- 2. Compute depth values for the first column.

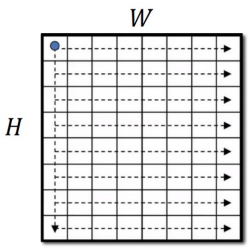
  for $y = 1$ *to* $(H-1)$

  $z(0,y) = z(0,y-1) - q(0,y)$
- 3. Compute depth values for each row.

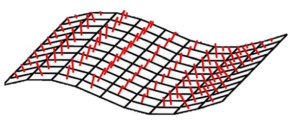
  for $y = 1$ *to* $(H-1)$

  for $x = 1$ *to* $(W-1)$

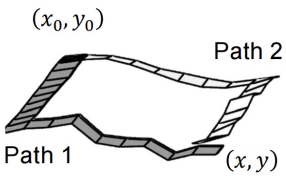
  $z(x,y) = z(x-1,y-1) - q(x,y)$



Shape from Normals: Integration-based Method



Actual surface with noisy estimates of surface gradient



Depth computed from noisy gradients depends on the integration path

One solution: Compute depth maps using different paths and find the average of computed depth maps to reduce error.

Estimating Shape using Least Squares

- Minimize the errors between the measured surface gradients and surface gradients of estimated surface.

$$D = \iint_{\text{Image}} \left(\frac{\partial z}{\partial x} + p\right)^2 + \left(\frac{\partial z}{\partial y} + q\right)^2 dxdy$$
$$= \sum_x \sum_y \left\| \nabla z(x,y) + \begin{bmatrix} p \\ q \end{bmatrix} \right\|^2$$

- Recall Poisson image blending: we can write the gradient operator (∇) as a sparse matrix, then find the optimal value via the Least Square.
- The resulting linear system is also called the Poisson equation.

Shape from Shading (A single Image)

Shape from a Single Image

- Shape from Shading, a special case of photometric stereo.
- **Input:** Image I, source direction s, and the surface reflectance model.
- **Output:** Surface gradient (p, q) for each intensity.

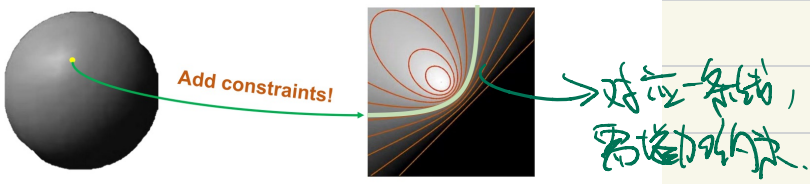
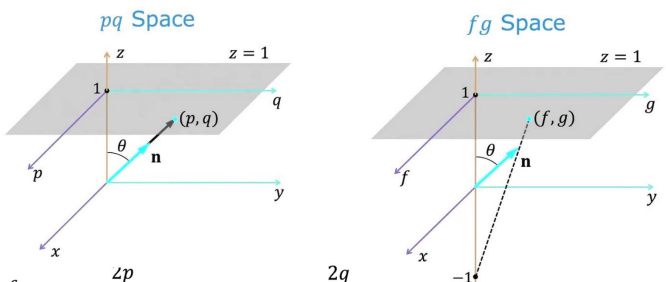


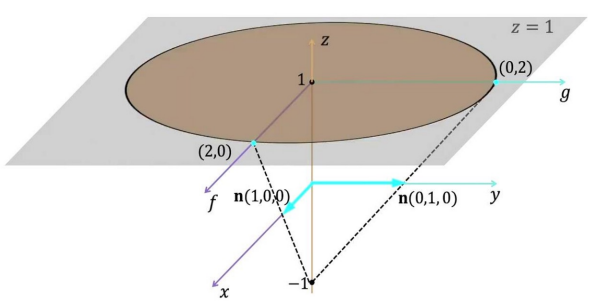
Image I Reflectance Map R

Stereographic Projection



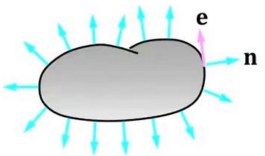
$$f = \frac{2p}{1+\sqrt{p^2+q^2+1}} \qquad g = \frac{2q}{1+\sqrt{p^2+q^2+1}}$$

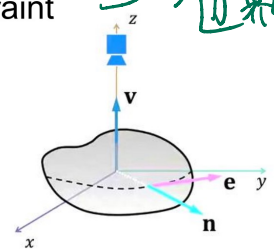
Stereographic Projection



All possible values of surface gradients visible on the hemisphere lie within a circle of radius 2 on the fg Plane: $I = R(f, g)$

Occluding Boundaries Constraint





$\mathbf{n} \perp \mathbf{v}$ $\mathbf{n} \perp \mathbf{e}$ $\mathbf{n} = \mathbf{e} \times \mathbf{v}$

- $\boldsymbol{n}$ is the surface normal on the boundary, $\boldsymbol{e}$ is the edge direction on the boundary, and $\boldsymbol{v}$ is the view direction.
- Surface gradients (f, g) on occluding boundary are known and can be used as boundary conditions

Image Intensity Constraint

- **Assumption**: Image intensity (irradiance) should equal to the reflection map: $I(x, y) = R_s(f, g)$, which can also be regarded as the data term.
- **Minimize**:

$$e_r = \iint (I(x, y) - R_s(f, g))^2 \, dx \, dy$$

- **Goal**: penalize errors between image irradiance and reflectance map.

Smoothness Constraint

- **Assumption**: Object surface is smooth. That is, the gradient values (f, g) vary slowly.
- **Minimize:**

$$e_s = \iint (f_x^2 + f_y^2) + (g_x^2 + g_y^2) \, dx \, dy$$

where: $f_x = \frac{\partial f}{\partial x}$, $f_y = \frac{\partial f}{\partial y}$, $g_x = \frac{\partial g}{\partial x}$ and $g_y = \frac{\partial g}{\partial y}$

- **Goal**: penalize rapid changes in f and g during surface estimation.

Shape from Shading

- Find surface gradients (f, g) at all image points that minimize the function:

$$e = e_s + \lambda e_r$$

- where e_s is the smoothness constraint, e_r is the image irradiance error, and λ is the weight.

- The known surface gradients (f, g) on occluding boundary are held as constants.

Shape from Shading

- Find surface gradients (f, g) at all image points that minimize the function:

$$e = e_s + \lambda e_r$$
$$= \sum_i \sum_j \left(e_{si,j} + \lambda e_{ri,j}\right)$$

Smoothness Error at point (i,j):

$$e_{s_{i,j}} = \frac{1}{4}\left(\left(f_{i+1,j} - f_{i,j}\right)^2 + \left(f_{i,j+1} - f_{i,j}\right)^2 + \left(g_{i+1,j} - g_{i,j}\right)^2 + \left(g_{i,j+1} - g_{i,j}\right)^2\right)$$

Optimize the energy function with the numerical iterative method refer to "Numerical Shape from Shading and Occluding Boundaries"

Image Irradiance Error at point (i,j):

$$e_{r_{i,j}} = \left(I_{i,j} - R_s\left(f_{i,j}, g_{i,j}\right)\right)^2$$

Summary

- Photometric Stereo
  - Reconstruct 3D from images under different lights
- Shape from Normal
  - Reconstruct 3D shape from a normal map by solving the Poisson equation
- Shape from Shading
  - Reconstruct 3D from one single image

- Reference
  - Computer Vision: Algorithms and Applications, Chapter 13
  - Papers listed in the slides.

12 Optical Flow. → 图像中颜色的变化

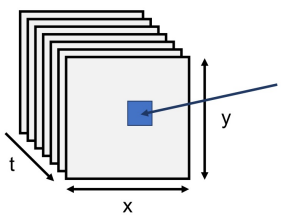
Optical Flow ⇒ 由相机运动、物体运动或光线变化引起的图像中亮度模式的明显变化.

→ 相机 / 物体 / 光源 } 运动

Optical Flow vs. Scene Flow

- Scene flow is the three-dimensional motion field of points in the world
- Input: Multiple images of a dynamic scene captured by multiple cameras
- Output: 3D shape and full 3D motion of every point at every instant in time

Optical Flow: Input



I(x,y,c,t)
- x,y – location
- c – channel
- t – time

→ 函数

Video: a sequence of image frames over time and is a function of space (x,y) and time t (and channel c).

→ 视频 = 图片序列

Optical flow ⇒ motion field 有相关.

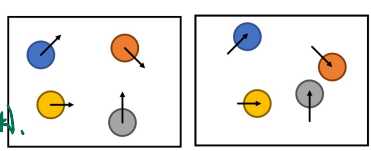
Optical Flow: Problem Definition

- Optical Flow: estimate pixel motion from image I(x,y,t) to image I(x,y,t+1).
- It can be regarded as a correspondence problem: given pixel at time t, find nearby pixels of the same color at time t+1

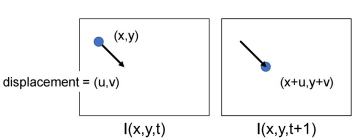
找 correspondence.

光流: 追踪特定像素的移动.

Key assumptions: 光线是恒定的.
- Color/brightness constancy: point at time t looks same at time t+1. (物体颜色不变)
- Small motion: points do not move very far away. (空间连续)
- Spatial coherence: points move like their neighbors. (相邻像素与目标像素的移动相似)



Optical Flow



Brightness constancy: $I(x, y, t) = I(x + u, y + v, t + 1)$

The displacement (u, v) is during one time unit, thus can also be regarded as the speed.

Brute force match: too costly

Optical Flow Equation

Brightness constancy: $I(x, y, t) = I(x + u, y + v, t + 1)$

Linearize the right-hand side using the Taylor expansion:

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t = I(x, y, t)$$

Optical flow equation: $I_x u + I_y v + I_t = I_t + \nabla I \cdot [u, v] = 0$

When is this approximation exact? [u,v] = [0,0]

When is it bad? u or v is too big.

Optical Flow Equation

Optical flow equation: $I_x u + I_y v + I_t = I_t + \nabla I \cdot [u, v] = 0$

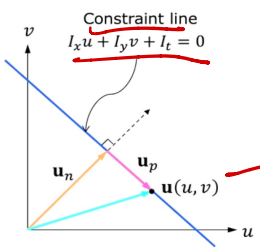
For a point (x, y), its optical flow (u, v) lies on the line: $I_x u + I_y v + I_t = 0$

Optical Flow can be split into two components.

$\mathbf{u} = \mathbf{u}_n + \mathbf{u}_p$

$\mathbf{u}_n$: Normal Flow

$\mathbf{u}_p$: Parallel Flow



Optical Flow Equation

Optical flow equation: $I_x u + I_y v + I_t = I_t + \nabla I \cdot [u, v] = 0$

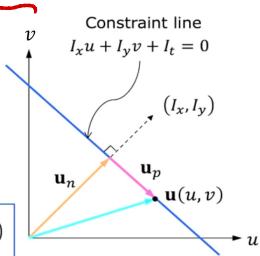
Direction of Normal Flow:

Unit vector perpendicular to the constraint line: $\hat{\mathbf{u}}_n = \frac{(I_x, I_y)}{\sqrt{I_x^2 + I_y^2}}$

Magnitude of Normal Flow:

Distance of origin from the constraint line: $|\mathbf{u}_n| = \frac{|I_t|}{\sqrt{I_x^2 + I_y^2}}$

$$\mathbf{u}_n = \frac{|I_t|}{(I_x^2 + I_y^2)}(I_x, I_y)$$

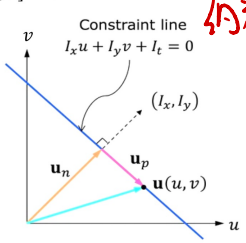


Optical Flow Equation

Optical flow equation: $I_x u + I_y v + I_t = I_t + \nabla I \cdot [u, v] = 0$

$$\mathbf{u}_n = \frac{|I_t|}{(I_x^2 + I_y^2)}(I_x, I_y)$$

- However, we cannot determine u_p, the optical flow component parallel to the constraint line
- Suppose (u, v) satisfies the constraint: $\nabla I \cdot (u, v) + I_t = 0$
- Then $\nabla I \cdot (u + u', v + v') + I_t = 0$ for any (u', v') s. t. $\nabla I \cdot (u', v') = 0$



Solving Optical Flow – Lucas Kanade

Optical flow equation: $I_x u + I_y v + I_t = 0$

- 2 unknowns [u,v], 1 equation per pixel, under-constrained.

2个未知数，一条方程

How do we get more equations?

- *Spatial coherence*: a pixel's neighbors *move together* / have the same [u,v]
- 5x5 window gives 25 new equations

连续.

$$I_t + I_x u + I_y v = 0$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

认为一个窗口内像素的移动都相同

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{25}) \end{bmatrix} \qquad \begin{matrix} \boldsymbol{A} & \boldsymbol{d} & = & \boldsymbol{b} \\ 25x2 & 2x1 & & 25x1 \end{matrix}$$

What's the solution? Least Square.

$$(\boldsymbol{A}^T \boldsymbol{A})\boldsymbol{d} = \boldsymbol{A}^T \boldsymbol{b} \quad \rightarrow \quad \boldsymbol{d} = (\boldsymbol{A}^T \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{b}$$

Intuitively, need to solve (sum over pixels in window)

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$\boldsymbol{A}^T \boldsymbol{A}$ $\qquad$ $\boldsymbol{A}^T \boldsymbol{b}$

Recall: **Harris** Corner Detector

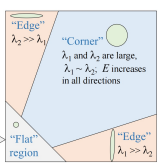
$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$\boldsymbol{A}^T \boldsymbol{A}$ $\qquad$ $\boldsymbol{A}^T \boldsymbol{b}$

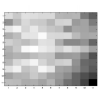
$$R = \frac{\det(\boldsymbol{M})}{trace(\boldsymbol{M}) + \epsilon} = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2 + \epsilon}$$

Estimation of optical flow is well-conditioned precisely for regions with corners

λ_1 and λ_2 are small; E is almost constant in all directions



Low Texture Region



$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \nabla I (\nabla I)^T$$

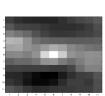
- gradients have small magnitudes
- small λ_1, small λ_2

Edges

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \nabla I (\nabla I)^T$$

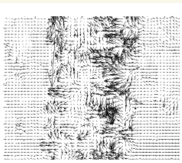
- large gradients, all the same
- large λ_1, small λ_2

High texture region



$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \nabla I (\nabla I)^T$$

- gradients are different, large magnitudes
- large λ_1, large λ_2



Failure: 当motion大于一个pixel时 =>

Solution: 降低分辨率（相当于减小motion），

多分辨率估计，迭代细化。

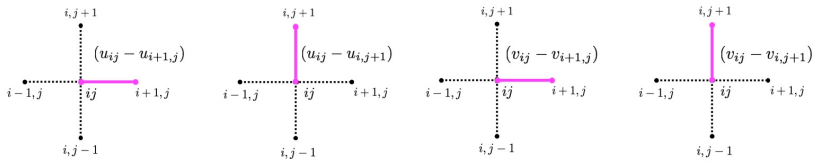
Horn-Schunck Optical Flow

- Instead of assuming the flow in a local window is the same, we can add a smoothness constraint

$$\min_{\boldsymbol{u},\boldsymbol{v}} \sum_{i,j} \left\{ E_s(i,j) + \lambda E_d(i,j) \right\} = \int\int (I_x u + I_y v + I_t)^2 + \lambda(|| \bigtriangledown u||^2 + || \bigtriangledown v||^2)$$

Brightness constancy Smoothness

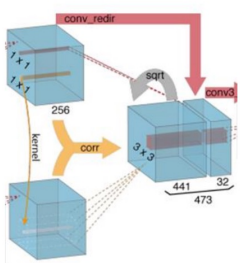
$$E_s(i,j) = \frac{1}{4}\left[(u_{ij} - u_{i+1,j})^2 + (u_{ij} - u_{i,j+1})^2 + (v_{ij} - v_{i+1,j})^2 + (v_{ij} - v_{i,j+1})^2\right]$$



FlowNet: Learning-based Optical Flow Estimation

- Correlation layer
  - For each pixel in the first image, perform the multiplicative patch comparison in the second image

$$c(\mathbf{x}_1, \mathbf{x}_2) = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f}_1(\mathbf{x}_1 + \mathbf{o}), \mathbf{f}_2(\mathbf{x}_2 + \mathbf{o}) \rangle$$



  - Denote the width and height of the image is W, H
  - For each location, limit the range by computing correlations in a neighborhood of size D
  - The output size is $W \times H \times D^2$

Summary of 3D Vision

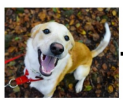
- Reconstruct 3D from 2D images: Resolve the single-view ambiguity
- Camera calibration
  - Calibration with vanishing points or with 2D-3D correpondences
- Epipolar geometry
  - Epipolar constraints; Essential / Fundamental matrix; 7/8-point algorithm
- Two-View stereo
  - Stereo Matching; Global stereo matching with dynamic programming
- Structure from motion
  - Factorization-based method; incremental SfM; Photo tourism.
- Multi-View Stereo
  - Plane sweeping; Patch-based MVS
- Photometric Stereo & Optical Flow

13 Image Recognition.

Supervised Learning

- **Data**: (x, y), where x is input / feature and y is label / target
- **Goal**: Learn a *function* $y = f(x)$

给定 tar.

 → Cat  → Dog  → Monkey

- Image classification: Predict a discrete category for each image
- Image regression: Predict a continuous value for each image

Unsupervised Learning

- **Data**: Just x, no label
- **Goal**: Learn the underlying structure in the data, or the distribution $P(x)$

不给 tar.

Clustering / Dimension reduction / Generation

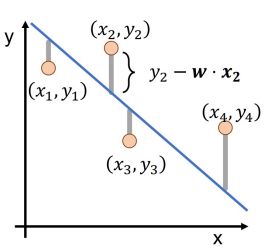
Linear Regression Model.

Linear Regression

Data:
$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N), \quad x_i, y_i \in \mathbb{R}$

Model:
$y = mx + b$
Or: $\boldsymbol{x} = (x, 1); \boldsymbol{w} = (m, b); \; y = \boldsymbol{w} \cdot \boldsymbol{x}$

Training:
$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)^2$



Solve Linear Regression with Least Squares

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)^2 = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2$$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} m \\ b \end{bmatrix} \Rightarrow \quad \boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Label: Vector of shape (N,)

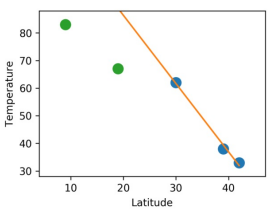
Inputs: Matrix of shape (N, 2)

Weights: Vector of shape (2,)

Solution: Solve with via Least squares

Overfitting and Regularization

- **Overfitting:** A model fits noise in the training set and doesn't generalize well to new data.
- From the view of *traditional machine learning*, a model that is too flexible is easy to overfit.
- **Regularization**: Weight decay is a commonly used regularization strategy.
- L2-Regularized Least Squares:



$$\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda\|\boldsymbol{w}\|^2 \longrightarrow \boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Fit training data

Penalize complexity

Linear Regression with Multi-Dimensional Inputs

Instead of scalar inputs $x_i \in \mathbb{R}$, it is common to have vector inputs $x_i \in \mathbb{R}^D$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)^2 = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2$$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,D} & 1 \\ \vdots & \ddots & \vdots & 1 \\ x_{N,1} & \cdots & x_{N,D} & 1 \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_D \\ b \end{bmatrix} \Rightarrow \quad \boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Label: Vector of shape (N,)

Inputs: Matrix of shape (N, D+1)

Weights: Vector of shape (D+1,)

Solution: Solve with via Least squares

Linear Regression with Multi-Dimensional Outputs

Vector inputs $x_i \in \mathbb{R}^{D_{in}}$, Vector outputs $y_i \in \mathbb{R}^{D_{out}}$

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} (y_i - \boldsymbol{W}\boldsymbol{x}_i)^2 = \arg\min_{\boldsymbol{W}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w}\|^2$$

$$\boldsymbol{Y} = \begin{bmatrix} y_{1,1} & \cdots & y_{1,D_{out}} \\ \vdots & \ddots & \vdots \\ y_{N,1} & \cdots & y_{N,D_{out}} \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,D_{in}} & 1 \\ \vdots & \ddots & \vdots & 1 \\ x_{N,1} & \cdots & x_{N,D_{in}} & 1 \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} W_{1,1} & \cdots & W_{D_{out}} \\ \vdots & \ddots & \vdots \\ W_{D_{in},1} & \cdots & W_{D_{in},D_{out}} \\ b_1 & \cdots & b_{D_{out}} \end{bmatrix}$$

Label: Vector of shape (N, D_{out})

Inputs: Matrix of shape (N, D_{in} +1)

Weights: Matrix of shape (D_{in} + 1, D_{out})

Summary of Linear Regression

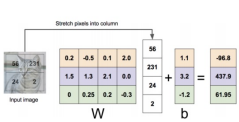
- Machine Learning is a form of data-driven programming
  - Supervised Learning maps inputs to outputs
  - Unsupervised Learning finds structure in unlabeled data
- Least Squares Linear Regression
  - Regularization can control overfitting and underfitting
  - Use the training set to find the optimal parameters
  - Use the validation set to choose hyperparameters
  - Use the test set once to estimate generalization

Linear Classifiers: 线性分类器.

Linear Classifier: Three Viewpoints

Algebraic Viewpoint

f(x,W) = Wx+b

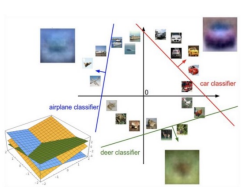


Visual Viewpoint

One template per class

Geometric Viewpoint

Hyperplanes cutting up space



Linear Classification with Least Squares

How to train the linear classifier? We need a Loss function for **optimization.**

- Solution 1: Use the L2 loss and solve with the least squares.
- $\boldsymbol{x}_i \in \mathbb{R}^D$ is the image features. $\boldsymbol{y}_i \in \mathbb{R}^C$ is the **one-hot** label: $y_{i,c} = 1$ if $\boldsymbol{x}_i$ has category c, 0 otherwise

Training ($\mathbf{x}_i$,$\mathbf{y}_i$): $\arg\min_{\boldsymbol{W}} \sum_{i=1}^{n} \|\boldsymbol{W}\boldsymbol{x}_i - \boldsymbol{y}_i\|^2$

Inference (x): $\boldsymbol{arg\,max_c}\ \boldsymbol{y} = \boldsymbol{arg\,max_c}\ \boldsymbol{Wx}$

- Using regression for classification is often effective in practice.
- The reverse (regression via discrete bins) is also common.

General Introduction of Loss Functions

- A loss function tells how good our current classifier is.
  - Low loss = good classifier; High loss = bad classifier
- Also called: objective function; cost function
- Negative loss function sometimes called reward function, profit function, utility function, fitness function

- Given a dataset $\{(x_i, y_i)\}_{i=1}^{N}$ of images x_i and labels y_i
- Loss for a single example is: $L_i(f(x_i, W), y_i)$
- Loss for the dataset is $L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)$

Cross-Entropy Loss

- Interpret classifier scores as **probabilities** and get the Cross-Entropy Loss

Classifier scores $s = f(x_i, W)$

Softmax function $p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$

Cross-Entropy Loss $L_i = -\log(p_{y_i})$

	Unnormalized log-probabilities / logits	exp	Unnormalized probabilities	normalize	Probabilities (Sum to 1)
cat	**3.2**	→	**24.5**	→	**0.13**
car	5.1	→	164	→	0.87
frog	-1.7	→	0.18	→	0.00

L_i = -log(0.13) = ***2.04***

Maximum Likelihood Estimation: Choose the weights to maximize the likelihood of observed data

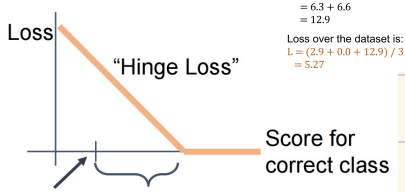
Multiclass SVM Loss

- The score of the correct class should be higher than all the other scores by at least a margin
- Given an example (x_i, y_i), where x_i is image, y_i is label, let $s = f(x_i, W)$ be scores
- Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

where $Margin = 1$

- Use weight decay !!!



$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$
$$= \max(0, 2.2 - (-3.1) + 1) + \max(0, 2.5 - (-3.1) + 1)$$
$$= \max(0, 6.3) + \max(0, 6.6)$$
$$= 6.3 + 6.6$$
$$= 12.9$$

Loss over the dataset is:
$L = (2.9 + 0.0 + 12.9) / 3$
$= 5.27$

	cat	car	frog
			
cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
loss	2.9	0.0	12.9

Cross-Entropy vs SVM Loss

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What is the cross-entropy loss? What is the SVM loss?

A: Cross-entropy loss > 0, SVM loss = 0

Q: What happens to each loss if we slightly change the scores of the last datapoint?

A: Cross-entropy loss will change; SVM loss will stay the same

- Scores
  - [10, -2, 3]
  - [10, 9, 9]
  - [10, -100, -100]
- $y_i = 0$

In practice, SVM and Softmax are usually comparable.

Summary

- **Image Classification** is a core computer vision task
- **Linear classifiers** learn one template per category to match with the input
- A **loss function** specifies your preference over different settings of weights
- **Cross-Entropy loss** maximizes probability of correct class
- **SVM Loss** wants correct score larger than other scores
- **Weight decay** is a widely used regularization method to alleviate overfitting

$$L(\boldsymbol{W}) = \lambda \|\boldsymbol{W}\|_2^2 + \sum_{i=1}^{n} -\log\left(\frac{\exp(s_{y_i})}{\sum_k \exp(s_k)}\right)$$

$$L(\boldsymbol{W}) = \lambda \|\boldsymbol{W}\|_2^2 + \sum_{i=1}^{n} \sum_j \max(0, s_j - s_{y_i} + 1)$$

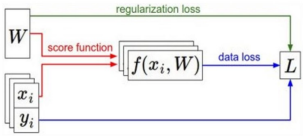
- How to find the $\boldsymbol{W}$ by minimizing these loss functions ?
- How to extend linear classifiers if not linearly separatable ?

14 Optimization

Train Linear Classifiers

- Linear classifiers: $s = f(x; W) = \hat{W}x + b = Wx$
  - Softmax Loss: $L(W) = \lambda\|W\|_2^2 + \sum_{i=1}^{n} -\log\left(\frac{\exp(s_{y_i})}{\sum_k \exp(s_k)}\right)$
  - SVM Loss: $L(W) = \lambda\|W\|_2^2 + \sum_{i=1}^{n}\sum_j \max(0, s_j - s_{y_i} + 1)$
- Given a set of training data $\{(x_i, y_i)\}$, our goal is to find the optimal W^* by optimizing the following the loss function (**How**?)

$$W^* = \arg\min L(W)$$

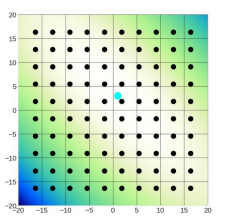


① Grid Search (遍历)

Optimization with Grid Search

```
best, bestScore = None, Inf
for d1 in range1:
    for d2 in range2:
        w = [d1, d2]
            if L(w) < bestScore:
                best, bestScore = w, L(w)
```

- Pros and cons
  - Simple: Only requires evaluating the models
  - High complexity in high-dimensional spaces



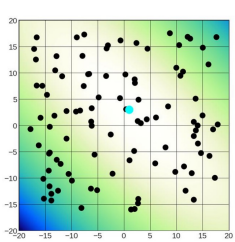
$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$

② Random Search (随机)

Optimization with Random Search

```
best, bestScore = None, Inf
for iter in range(numIters):
    w = random(N,1)  # sample
    score = L(w)     # evaluate
    if score < bestScore:
        best, bestScore = w, score
```

- Pros and cons
  - Simple: Only requires evaluating the models
  - Throwing darts at high-dimensional spaces might miss the optimal values



$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$

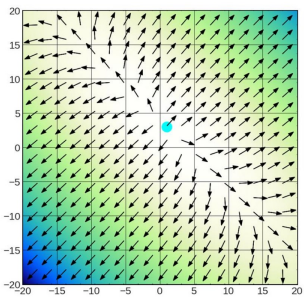
When to Use Grid/Random Search?

- The number of dimensions is small
- The space is bounded
- The objective function is impossible/hard to analyze
  - We need only evaluate the function!
- Random search is often more effective when combined with other strategies
  - E.g., Genetic Algorithm, Simulated Annealing
- Grid search makes it easy to systematically test something

③ Optimization with Gradient Decent 梯度下降

Optimization with Gradient Decent

- Gradients represent the direction and rate of the fastest increase of a function.
- Gradient Decent: Take repeated steps in the direction of negative gradient of a function at the current point to find the (local) minimum of the function.



Optimization with Gradient Decent

- Gradient Decent: at each step, move in the direction of negative gradient.

$$\arg\min_w L(w) \Rightarrow \nabla_w L(w) = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_N} \end{bmatrix} \Rightarrow w = w + \alpha \nabla_w L(w)$$

How to compute $\nabla_w L(w)$?

How to choose α (learning rate)?

Iterate

Computing Gradients: Analytic Method

- Take the linear regression as an example, we can calculate the gradients

$$L(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \left(y_i - \boldsymbol{w}^T \boldsymbol{x}_i\right)^2$$

$$\nabla_w L(\boldsymbol{w}) = 2\lambda \boldsymbol{w} - \sum_{i=1}^{n} 2(y_i - \boldsymbol{w}^T \boldsymbol{x}_i)\boldsymbol{x}_i$$

- Numerical gradient: approximate, slow, easy to write
- Analytic gradient: exact, fast, error-prone
- In practice, always use the analytic gradient, implemented in a smart approach: backpropagation (explained in the next course)

Stochastic Gradient Descent: SGD

- Problem of Gradient Descent: the sum is expensive when N is large!

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

- **Solution**: Approximate the sum using a minibatch of examples, e.g. 32

$$\nabla_W L(W) = \frac{1}{B} \sum_{i=1}^{B} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

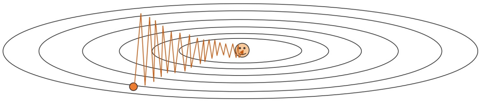
```
w = initialize_weights()
for t in range(num_steps):
  minibatch = sample_data(data, batch_size)
  dw = compute_gradient(loss_fn, minibatch, w)
  w -= learning_rate * dw
```

Hyperparameters:

- Number of steps?
- Batch size (shuffle)?
- Weight initialization?
- Learning rate / step size?

SGD的一些问题：①收敛速度在各方向上不一致（但lr对各方向相同）.

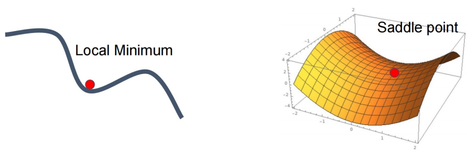
- What if loss changes quickly in one direction and slowly in another?
- The loss function has a high **condition number**: the ratio of the largest to the smallest singular value of the Hessian matrix is large.



- Slow progress along shallow dimension, jitter along steep direction.
- One could reduce the step size to prevent this issue, however, the convergence will be slow.

②可能找到 local minimization (not global).

- What if the loss function has a **local minimum** or **saddle point**?
- For local minima or saddle points, the gradients are zero; SGD gets stuck!



③使用minibatch会噪音较大

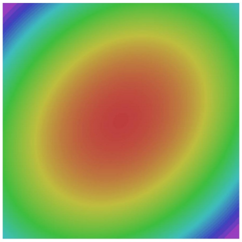
- The gradients come from minibatches so they can be noisy!

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N}\nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

⇩

$$\nabla_W L(W) = \frac{1}{B}\sum_{i=1}^{B}\nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

- **Batch size B ≪ total sample number N**



解决方法：①SGD with Momentum

SGD with Momentum

- Key idea: Build up the *velocity* for loss decent as a running mean of gradients

SGD

$$v_{t+1} = \nabla L(w_t)$$
$$w_{t+1} = w_t - \alpha v_{t+1}$$

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

SGD with Momentum

$$v_{t+1} = \rho v_t + \nabla L(w_t)$$
$$w_{t+1} = w_t - \alpha v_{t+1}$$

← 累积历史速度 v_t.

↓

容易跳出局部最小.

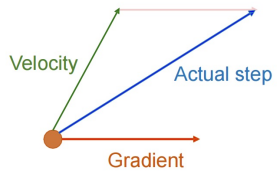
```
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

- ρ is often set to 0.9, and $v_{t+1} = h_t + \rho h_{t-1} + \rho^2 h_{t-2} + \cdots + \rho^t h_0$, $h_t = \nabla L(w_t)$

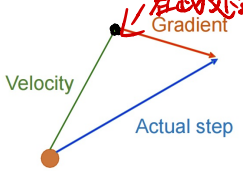
(2)

Nesterov Momentum

- "Look ahead" to the point where the velocity would take us; compute the gradient there and mix it with the velocity to get the actual update direction.



Momentum update



Nesterov Momentum

Nesterov Momentum

SGD with Momentum (TensorFlow)

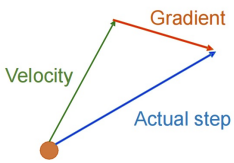
$$v_{t+1} = \rho v_t - \alpha \nabla L(w_t)$$
$$w_{t+1} = w_t + v_{t+1}$$

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla L(w_t + \rho v_t)$$
$$w_{t+1} = w_t + v_{t+1}$$

Change of variables: $\tilde{w}_t = w_t + \rho v_t$

$$v_{t+1} = \rho v_t - \alpha \nabla L(\tilde{w}_t)$$
$$\tilde{w}_{t+1} = \tilde{w}_t - \rho v_t + (1+\rho) v_{t+1}$$

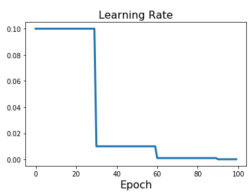


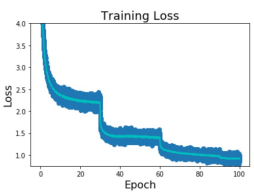
Annoying, usually we want to update in terms of w_t and $\nabla L(w_t)$.

```
for t in range(num_steps):
  dw = compute_gradient(w)
  old_v = v
  v = rho * v - learning_rate * dw
  w -= rho * old_v - (1 + rho) * v
```

Learning Rate Schedule

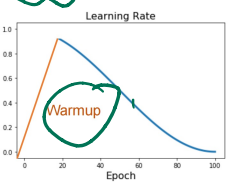
- **Idea**: Start with high learning rate, reduce it over time.
- **Stepwise Decay**: Reduce by some factor at fixed iterations

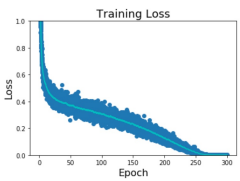




Learning Rate Schedule

- **Idea**: Start with high learning rate, reduce it over time.
- **Cosine Decay:** reduce the learning rate following $\alpha_t = \frac{1}{2}\alpha_0\left(1 + cos\left(\frac{t\pi}{T}\right)\right)$





Adaptive Learning Rate Methods.

AdaGrad

- Tuning the learning rates is an expensive process; Can we adaptively tune the learning rates, and even do so per parameter?
- AdaGrad adds element-wise scaling of the gradient based on the historical sum of squares in each dimension, which works well with sparse gradients.

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

- AdaGrad accumulates *grad_squared*, which keeps growing without bound.

RMSProp: "Leaky Adagrad"

AdaGrad

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

Leaky: $s_t = \gamma s_{t-1} + (1-\gamma) g_t^2 = \gamma[g_t^2 + (1-\gamma) g_{t-1}^2 + (1-\gamma)^2 g_{t-2}^2 + \cdots]$

Adam: RMSProp + Momentum

- Adam (Adaptive Momentum) combines the benefits of RMSProp and Momentum.

Adam

```
moment1 = 0
moment2 = 0
for t in range(1, num_steps + 1):  # Start at t = 1
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Momentum

RMSProp

- What happens at t=1? (Assume beta2 = 0.999)
- The moment2 will be extremely small, resulting in instability.

Adam: Bias Correction

⇒为何在迭代初期对梯度的估计要修正？

- Bias Correction: Normalize the accumulated moments by $1/(1-\beta^t)$.

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^2 = (1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\cdot g_i^2$$

$$\mathbb{E}[v_t] = \mathbb{E}\left[(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\cdot g_i^2\right] = \mathbb{E}[g_t^2]\cdot(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i} + \zeta = \mathbb{E}[g_t^2]\cdot(1-\beta_2^t) + \zeta$$

Adam优化器中的梯度归一化是通过计算每个参数的自适应学习率来进行的。这个过程涉及到两个关键的概念：一阶矩估计（即梯度的均值）和二阶矩估计（即梯度的未中心化的方差）。以下是具体的步骤：

1. **计算梯度的一阶矩（m）和二阶矩（v）的估计**：
   - 一阶矩估计（m）是对梯度g的指数移动平均（Exponential Moving Average, EMA）：
     $$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1)\cdot g_t$$
     其中，m_t是t时刻的一阶矩估计，m_{t-1}是上一时刻的估计，g_t是t时刻的梯度，β_1是矩估计的衰减率（通常接近1，例如0.9）。
   - 二阶矩估计（v）是对梯度平方的指数移动平均：
     $$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^2$$
     其中，v_t是t时刻的二阶矩估计，v_{t-1}是上一时刻的估计，g_t^2是t时刻梯度的平方，β_2是另一个衰减率（通常也接近1，例如0.999）。
2. **偏差校正**：
   - 由于m和v是在迭代开始时初始化为0的，所以在迭代初期，它们会被低估。为了解决这个问题，需要对m和v进行偏差校正：
     $$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
     $$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
     其中，$\hat{m}_t$和$\hat{v}_t$分别是偏差校正后的一阶矩和二阶矩的估计。
3. **更新参数**：
   - 最后，使用这些经过偏差校正的矩估计来更新参数：
     $$\theta_{t+1} = \theta_t - \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$
     其中，θ_t是t时刻的参数，ϵ是一个很小的常数，以避免除以零（通常设为10^{-8}）。

Adam: Full Method

- Adam (Adaptive Momentum) combines the benefits of RMSProp and Momentum.

```
moment1 = 0
moment2 = 0
for t in range(1, num_steps + 1):  # Start at t = 1
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  moment1_unbias = moment1 / (1 - beta1 ** t)
  moment2_unbias = moment2 / (1 - beta2 ** t)
  w -= learning_rate * moment1_unbias / (moment2_unbias.sqrt() + 1e-7)
```

L2 Regularization vs Weight Decay

- However, for a long period of time, the performance of models trained with SGD with momentum is better than Adam on the testing set.
- Due to the gradient normalization in Adam, Weight Decay is "canceled"

Optimization Algorithm

$$L(w) = L_{data}(w) + L_{reg}(w)$$
$$g_t = \nabla L(w_t)$$
$$s_t = optimizer(g_t)$$
$$w_{t+1} = w_t - \alpha s_t$$

正则化项

L2 Regularization vs. Weight Decay

$$L(w) = L_{data}(w) + \lambda |w|^2$$
$$g_t = \nabla L(w_t) = \nabla L_{data}(w_t) + 2\lambda w_t$$
$$s_t = optimizer(g_t) + 2\lambda w_t$$
$$w_{t+1} = w_t - \alpha s_t$$

$\frac{g_t}{|G_t|+w_t}$ 当$|G_t|$很大时，w_t的影响很小，这个weight Decay就没效果了，overfitting

AdamW Should be Your Default Optimizer

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

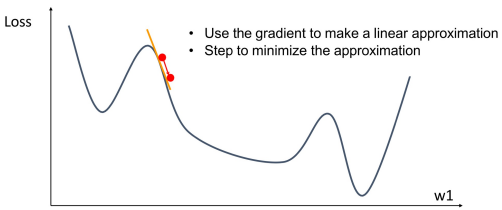
given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t+1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow$ SelectBatch$(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda\boldsymbol{\theta}_{t-1}$ ⇒ weight decay
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1-\beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1-\beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow$ SetScheduleMultiplier(t) ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda\boldsymbol{\theta}_{t-1}\right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

归一化(梯度修正)

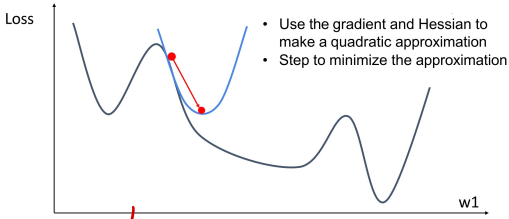
Summary of Optimization Algorithms

Algorithm	Tracks the first moments	Tracks the second moments	Leaky second moments	Bias correction	Weight decay
SGD	✗	✗	✗	✗	✓
SGD+Momentum	✓	✗	✗	✗	✓
Nesterov	✓	✗	✗	✗	✓
AdaGrad	✗	✓	✗	✗	✗
RMSProp	✗	✓	✓	✗	✗
Adam	✓	✓	✓	✓	✗
AdamW	✓	✓	✓	✓	✓

First-Order Optimization: All Previous Methods



Second-Order Optimization: The Newton's Method



Second-Order Optimization: The Newton's Method

- The second-order Taylor expansion:

$$L(w) \approx L(w_0) + (w - w_0)^\intercal \nabla_w L(w_0) + \frac{1}{2}(w - w_0)^\intercal \mathbf{H}_w L(w_0)(w - w_0)$$

- Solve for the critical point: Newton parameter update (direction and step size)

$$w^* = w_0 - \mathbf{H}_w L(w_0)^{-1} \nabla_w L(w_0)$$

- The Newton's method is generally not suitable for optimizing neural networks
  - The Hessian has O(N^2) elements, and the matrix inverting takes O(N^3)
  - N = (Tens or Hundreds of) Millions

Second-Order Optimization: L-BFGS

- The bottleneck is the computation of the inverse Hessian matrix $O(N^3)$.

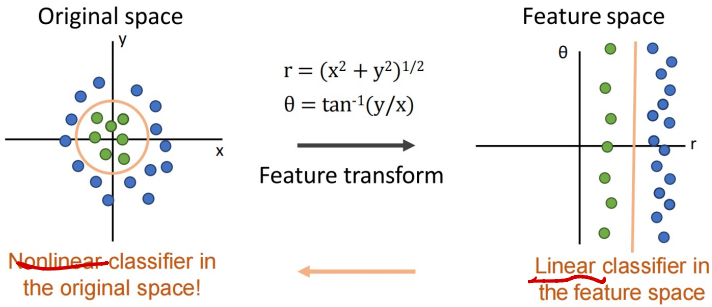
$$w^* = w_0 - \mathbf{H}_w L(w_0)^{-1} \nabla_w L(w_0)$$

- Quasi-Newton methods approximate the (inverse) Hessian with gradients over time $O(N^2)$ and do not explicitly compute the inverse Hessian.
- BGFS is one popular Quasi-Newton method (L-BFGS: limited memory BFGS)
- BGFS usually works very well in full batch, deterministic mode
  - For a single, deterministic f(x) then L-BFGS will probably work very nicely
- BGFS does not transfer very well to mini-batch setting.
  - Adapting second-order methods to large-scale, stochastic setting is an active area of research.

15 Neural Networks.

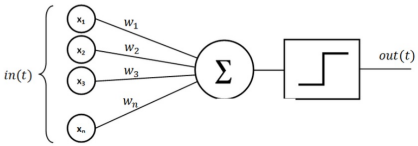
The Ability of Linear Classifiers is Limited

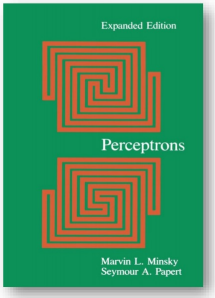
- **Idea #1**: Extract high-dimensional features, then apply linear classifiers.



Single-Layer Perceptron

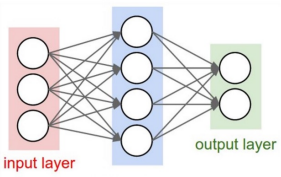
- Minsky and Papert published Perceptrons and showed that single-layer perceptrons (SLPs) cannot solve some very simple problems (XOR).
- SLPs are essentially linear classifiers.
- Contributed to the so-called AI winter.





Multilayer Perceptrons

- Input image: $x \in \mathbb{R}^D$; Category scores: $s \in \mathbb{R}^C$
- Linear Classifier: $s = Wx$, $W \in \mathbb{R}^{C \times D}$
- **2-layer Neural Networks** (MLPs):
  $s = W_2\sigma(W_1 x)$, $W_1 \in \mathbb{R}^{H \times D}$, $W_2 \in \mathbb{R}^{C \times H}$
- where H is the number of hidden neurons, and σ is a non-linear activation function.
- The activation function is indispensable (why), and should be differentiable (why).



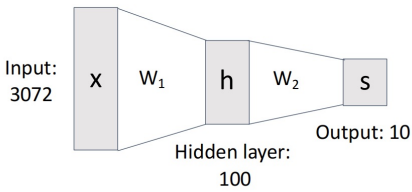
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Neural Networks: Visual Viewpoint

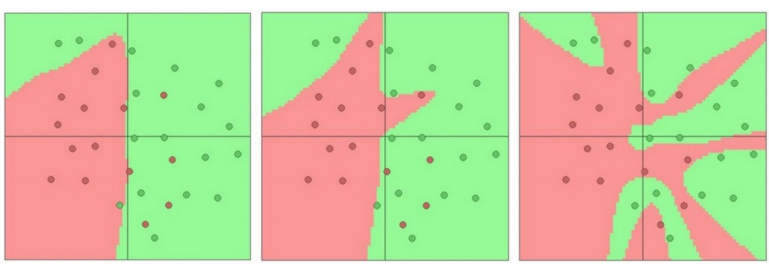
2-layer Neural Network

The first layer is bank of templates; The second layer recombines templates



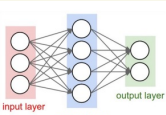
Neural Networks: Geometric Viewpoint

- Neural networks allow nonlinear decision boundaries!
- MLPs: More hidden units → More capacity

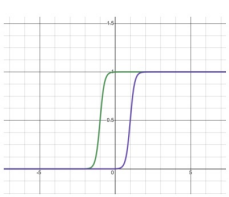


3 hidden units | 6 hidden units | 20 hidden units

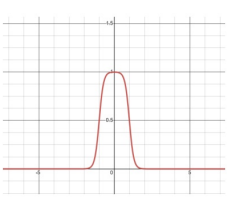
Universal Approximation of MLPs



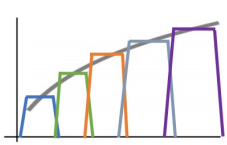
- MLPs with at least one hidden layer can approximate any function with arbitrary precision (Proof by construction: Approximating a function $f: \mathbb{R} \to \mathbb{R}$)



Sigmoid activation functions



Build a "basis function" using two hidden units

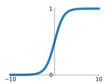


Use the "basis functions" to approximate arbitrary functions

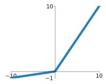
Activation Functions

- Widely-used activation functions (All of them can be used to form "basis functions")

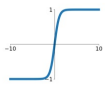
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



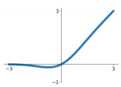
Leaky ReLU
$\max(0.1x, x)$



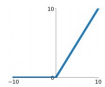
tanh
$\tanh(x)$



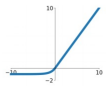
GELU
$\approx x\sigma(1.702x)$



ReLU
$\max(0, x)$

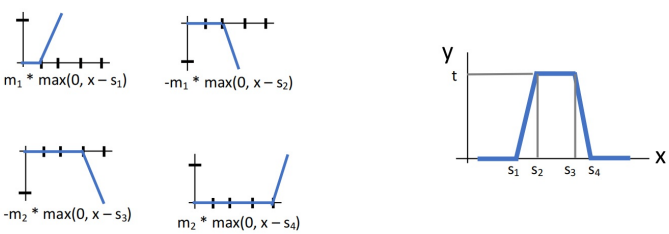


ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



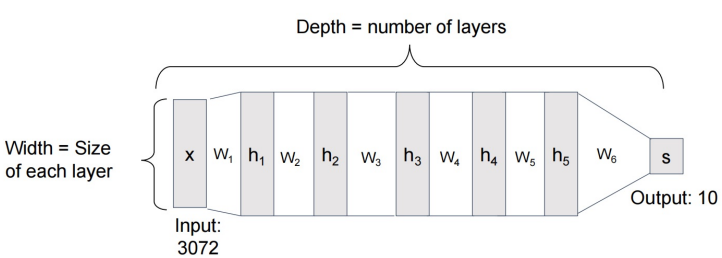
Activation Functions

- Build "basis functions" with a ReLU activation functions
- ReLU can be your default activation function (other reasons?)



Deep Neural Networks

- **#1 Limitation of MLPs**: Universal Approximation of MLPs → Significantly Large number of hidden neurons (Width) → Deep Neural Networks



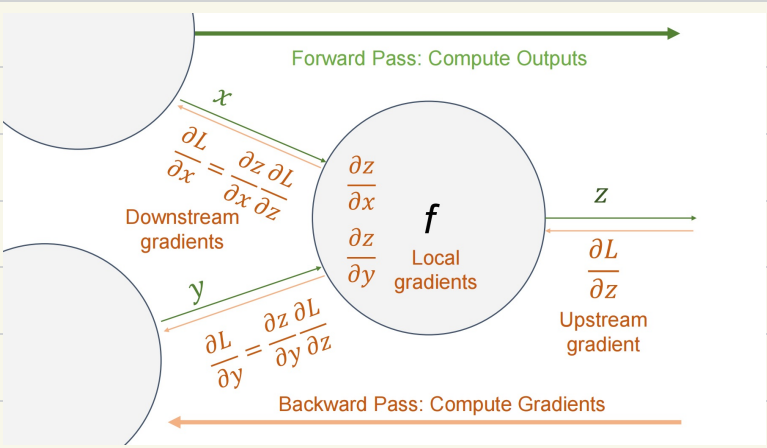
Summary

- From linear classifiers to multilayer percetions (MLPs)
- Intuitive explanations of Universal Approximation of MLPs
- Activation functions of MLPs (ReLU, Sigmoid)
- Geometric and visual viewpoint of MLPs
- From MLPs to Convolutional Neural networks

1b Back Propagation

Problem: How to Compute Gradients?

- Compute gradients to train the neural network with SGD / AdamW.
- **Example**: A 2-layer neural network: $s = W_2 \max(0, W_1 x + b_1) + b_2$
- Per-element data loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
- L2 regularization: $R(W) = \sum_k W_k^2$
- Total loss: $L(W_1, W_2, b_1, b_2) = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$.
- **Goal**: compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}$.
- **Compute gradients manually**: error prone and tedious; not feasible for very complex models!



Back Propagation

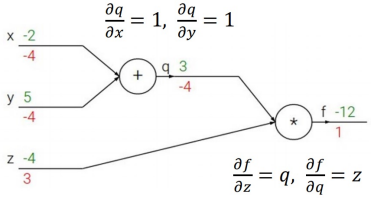
- Example: $f(x, y, z) = (x + y) \cdot z$
- Forward pass: Compute outputs

  $x = -2,\ y = 5,\ z = -4$

  $q = x + y,\ f = q \cdot z$
- Backward pass: Compute derivatives

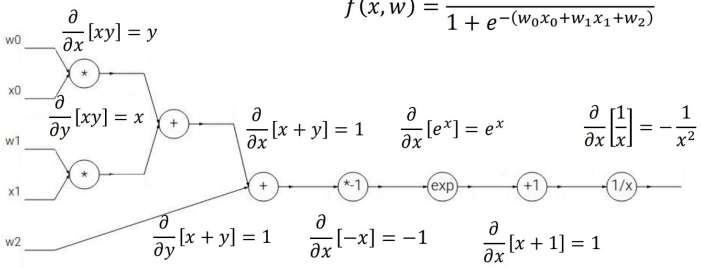
  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
- Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = -4, \quad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = -4, \quad \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = 3$$



Back Propagation

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

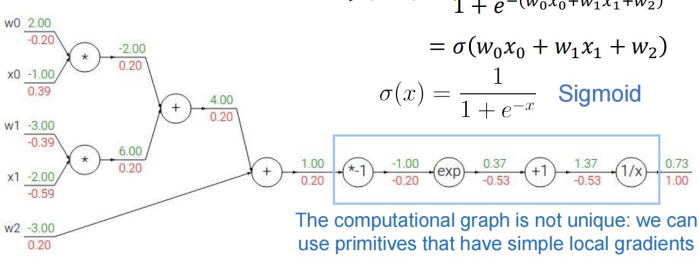


Back Propagation

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$= \sigma(w_0 x_0 + w_1 x_1 + w_2)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \text{Sigmoid}$$

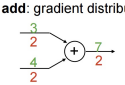


The computational graph is not unique: we can use primitives that have simple local gradients

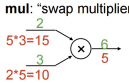
$$\frac{\partial}{\partial x}[\sigma(x)] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}}\right)\left(\frac{1}{1 + e^{-x}}\right) = (1 - \sigma(x))\sigma(x)$$

Common Patterns in Gradient Flow

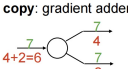
add: gradient distributor



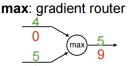
mul: "swap multiplier"



copy: gradient adder



max: gradient router



Back Propagation in Mathematics Viewpoint

- Given $\{(x, y)\}$, we can define a neural network recursively:

$$x_{n+1} = f_n(x_n; W_n), \quad x_0 = x$$

where f_n is the function for the n^{th} network layer, W_n is its parameters.

- The loss function for one sample can be defined as $L = \mathcal{L}(x_N, y)$.
- **Goal**: compute $\frac{\partial L}{\partial W_n}$ $(n = 0, \dots, N)$.
- **Back Propagation**: An application of the chain rule in calculus.
- First of all, compute $\frac{\partial L}{\partial x_N}$, then compute gradients recursively:

$$\frac{\partial L}{\partial x_n} = \frac{\partial L}{\partial x_{n+1}} \cdot \frac{\partial x_{n+1}}{\partial x_n} = \frac{\partial L}{\partial x_{n+1}} \cdot \frac{\partial f_n}{\partial x_n} \qquad \frac{\partial L}{\partial W_n} = \frac{\partial L}{\partial x_{n+1}} \cdot \frac{\partial x_{n+1}}{\partial W_n} = \frac{\partial L}{\partial x_{n+1}} \cdot \frac{\partial f_n}{\partial W_n}$$

Summary

- **Computational graphs** are a useful data structure for organizing computation in neural networks
- **Back Propagation** lets us compute gradients in a computational graph
- **Flat Back Propagation** has a backward pass that "mirrors" the forward pass
- **Auto Differentiation** composes pairs of forward and backward functions and reverse the computation graphs to compute the gradient automatically
- **Back Propagation** extends to vector and tensor-valued functions

复合函数的偏导与链式法则：

复合函数偏导：

以二元函数 $z = f(u, v)$ 为例，z 是 u，v 的函数，但若 u 和 v 又都是 x 和 y 的函数，则 z 最终是 x 和 y 的函数，即

$$z(x, y) = f[u(x, y), v(x, y)].$$

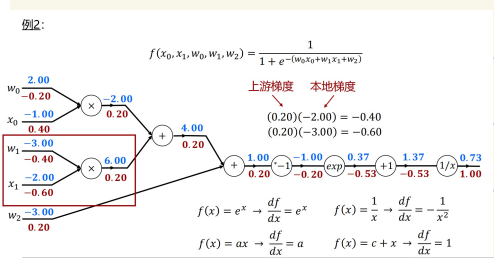
由 $z = f(u, v)$ 得

$$dz = \frac{\partial f}{\partial u} du + \frac{\partial f}{\partial v} dv.$$

又由于 $du = \frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy$，$dv = \frac{\partial v}{\partial x} dx + \frac{\partial v}{\partial y} dv$，代入上式得

$$dz = \frac{\partial f}{\partial u}\left(\frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy\right) + \frac{\partial f}{\partial v}\left(\frac{\partial v}{\partial x} dx + \frac{\partial v}{\partial y} dy\right)$$

例2：



$$z(x, y) = f[u(x, y), v(x, y)]$$

$$dz = \frac{\partial f}{\partial u}\left(\frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy\right) + \frac{\partial f}{\partial v}\left(\frac{\partial v}{\partial x} dx + \frac{\partial v}{\partial y} dy\right)$$
$$= \left(\frac{\partial f}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial x}\right) dx + \left(\frac{\partial f}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial y}\right) dy$$

这就是 z 关于 x 和 y 的全微分关系，根据偏导数的定义

$$\frac{\partial z}{\partial x} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial x}, \qquad \frac{\partial z}{\partial y} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial y}.$$

这也叫偏导的**链式法则**。