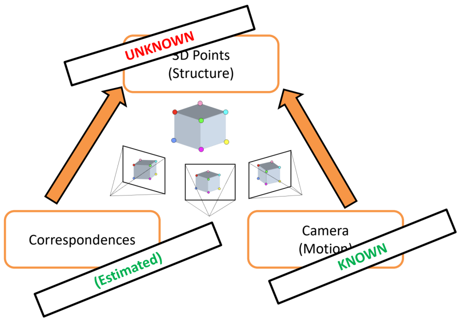
Final

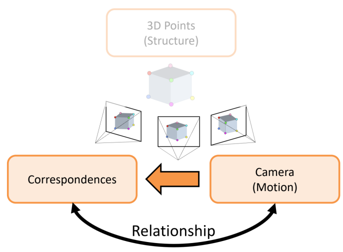
Lec 13 Epipolar Geometry + Calibration

Simple Stereo;
Corresp + Camera = Disparity = depth^{-1}

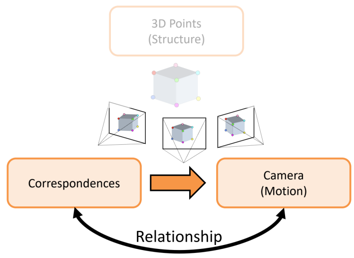


- The two cameras need not have parallel optical axes.
- Assume camera intrinsics are calibrated

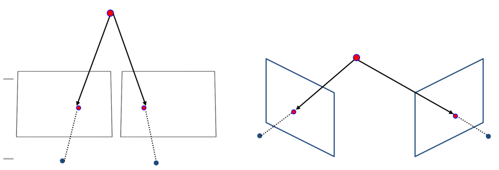
Camera helps Correspondence:
Epipolar Geometry



Correspondence gives camera:
Epipolar Geometry



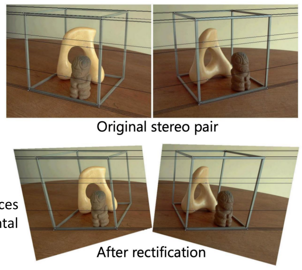
Camera (Motion) 与 Correspondence 关系串联:
Epipolar Geometry!



Same hammer:
Find the correspondences, then solve for structure

Given Camera: How to find Epipolar? Option 1: Homography: two image plane to one plane! ⇒ parallel

Option 1: Rectify via homography



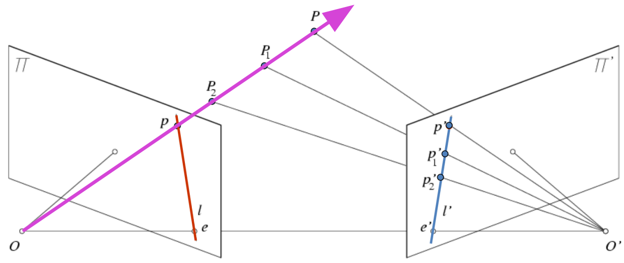
Then find correspondences on the horizontal scan line

General case, known camera, find depth:
Option 2

1. **Find correspondences**
2. Triangulate

Option2: Use math (Epipolar Geometry) to find epipolar

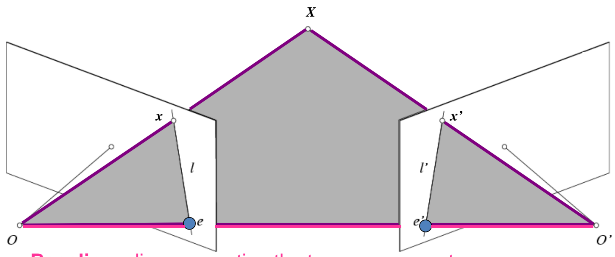
Epipolar constraint



- Potential matches for *p* have to lie on the corresponding epipolar line *l'*.
- Potential matches for *p'* have to lie on the corresponding epipolar line *l*.

Given O . O', P. then since Intrinsics are calibrated. so we can confirm O.O' equivalent position.
连线 OP. OP上每一点与O'连线, 与π'上的交点形成的连线便是 epipolar

Parts of Epipolar geometry



- **Baseline** – line connecting the two camera centers
- **Epipolar Plane** – plane containing baseline (1D family)
- **Epipoles**
  = intersections of baseline with image planes
  = projections of the other camera center
  = vanishing points of the baseline

因为 $\vec{op}$. $\vec{oo'}$ 形成了 Epipolar Plane!

△: 极点 Epipole: oo' (baseline) 与两个 Image Plane 的交点

如图可见:

Epipolars must pass through epipoles!

Epipoles *infinitely* far away, epipolar lines parallel

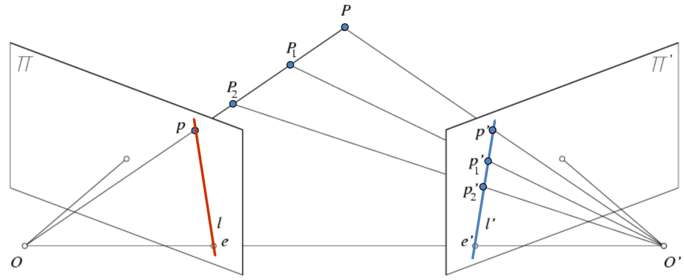
Ok so where were we?

- Setup: Calibrated Camera (both extrinsic & intrinsic)
- Goal: 3D reconstruction of corresponding points in the image
- We need to find correspondences!

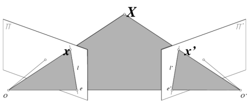
➔1D search along the epipolar line!

➔Need: Compute the epipolar line from camera

Ok so what exactly are l and l'?



Step 0: Factor out intrinsics



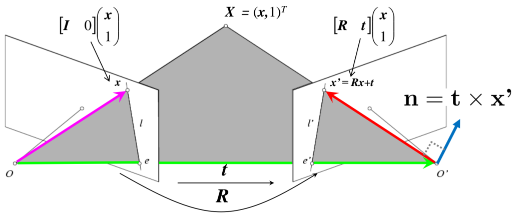
$$x = K[R\ \ t]X$$
$$K^{-1}x = [R\ \ t]X$$

- Let's factor out the effect of K (do everything in 3D)
- Make it into a ray with K^{-1} and use depth = 1
- This is called the *normalized* image coordinates. It may be thought of as a set of points with identity K

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

- Assume that the points are normalized from here on

Epipolar constraint: Calibrated case



The vectors x, t, and x' are coplanar

What can you say about their relationships, given $\mathbf{n} = \mathbf{t} \times \mathbf{x'}$?

$$\boxed{\mathbf{x'} \cdot (\mathbf{t} \times \mathbf{x'}) = 0}$$

$$\mathbf{x'} \cdot (\mathbf{t} \times (R\mathbf{x} + \mathbf{t})) = 0$$
$$\mathbf{x'} \cdot (\mathbf{t} \times R\mathbf{x} + \underbrace{\cancel{\mathbf{t} \times \mathbf{t}}}_{0})) = 0$$

$$\boxed{\mathbf{x'} \cdot (\mathbf{t} \times R\mathbf{x}) = 0}$$

Given K (intrinsic) and o'

(以 o 为参考系) 则:

$X_{\pi} = [I\ 0]X = x \quad X_{\pi'} = [R\ t]X = x'$

其中 R.t 是 o' 相对于 o 的 rotation

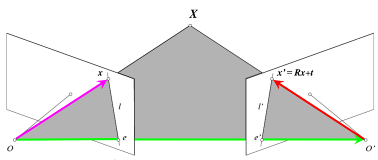
& translation $X_{\pi'}$

则: $t \times (\underline{Rx+t}) \perp x'$

$\therefore x' \cdot (t \times Rx) = 0$

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

$E = [t_\times]R$



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

Recall: $a \times b = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}\begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = [a_\times]b$

The vectors x, t, and x' are coplanar

$$\boxed{x'^T E x = 0}$$

E x **is the epipolar line associated with *x* (*l'* = *E x*)**

- Recall: a line is given by $ax + by + c = 0$ or

$$\mathbf{l}^T\mathbf{x} = 0 \quad \text{where} \quad \mathbf{l} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' \cdot [t \times (Rx)] = 0 \Rightarrow x'^T \underbrace{[t_\times]R}_{E} x = 0 \Rightarrow x'^T E x = 0$$

Essential Matrix
(Longuet-Higgins, 1981)

Recall: $a \times b = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}\begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = [a_\times]b$

The vectors x, t, and x' are coplanar

$$\boxed{x'^T E x = 0}$$

E x is the epipolar line associated with ***x*** (***l'*** = ***E x***)

$E^T x'$ is the epipolar line associated with x' ($l = E^T x'$)

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

E is singular (rank two)

E has five degrees of freedom

Recall, knowing the camera gives you the essential matrix (i.e. the plane per point)

So the DoF has to match up

Essential matrix: 3 x 3, 9 numbers, but rank2 means 2 columns fully define = 6 parameters -1 for scale = 5 DoF

Extrinsic Camera (R, T): 3 for rotation, 3 for translation, but -1 for scale = 5 DoF!

Some property of E.

If K is not the same ?

Epipolar constraint: Uncalibrated case

- Recall that we normalized the coordinates

$$x = K^{-1}\hat{x} \quad x' = K'^{-1}\hat{x}' \qquad \hat{x} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

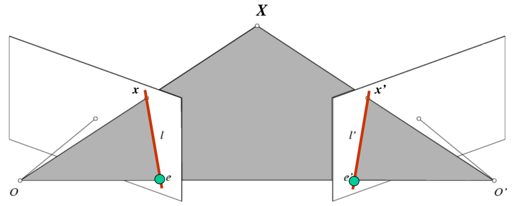
where $\hat{x}$ is the image coordinates

- But in the *uncalibrated* case, ***K*** and ***K'*** are unknown!
- We can write the epipolar constraint in terms of *unknown* normalized coordinates:

$$x'^T E x = 0$$
$$(K'^{-1}\hat{x}')^T E (K^{-1}\hat{x}) = 0$$
$$\hat{x}'^T \underbrace{K'^{-T} E (K^{-1}}\hat{x}) = 0$$
$$\hat{x}'^T F \hat{x} = 0$$

$$F = K'^{-T} E K^{-1}$$

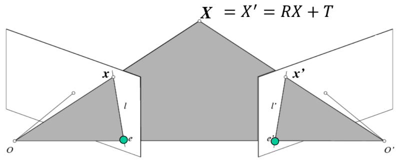
Fundamental Matrix
(Faugeras and Luong, 1992)



$x'^T E x = 0$ ⇒ $\hat{x}'^T F \hat{x} = 0$ with $F = K'^{-T} E K^{-1}$

- $\boldsymbol{F}\hat{x}$ is the epipolar line associated with $\hat{x}$ ($\boldsymbol{l}' = \boldsymbol{F}\hat{x}$)
- $\boldsymbol{F}^T\hat{x}'$ is the epipolar line associated with $\hat{x}'$ ($\boldsymbol{l} = \boldsymbol{F}^T\hat{x}'$)
- $\boldsymbol{F}\boldsymbol{e} = 0$ and $\boldsymbol{F}^T\boldsymbol{e}' = 0$
- $\boldsymbol{F}$ is singular (rank two)
- $\boldsymbol{F}$ has *seven* degrees of freedom

We know about the camera, K_1, K_2 and [R t]:



and found the corresponding points: $x \leftrightarrow x'$

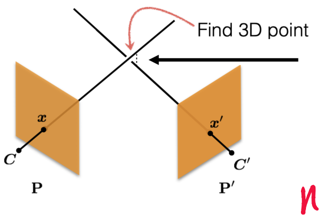
$$x = KX \qquad x' = K'X' = K'(RX + T)$$

only unknowns!

How many unknowns + how many equations do we have?

Solve by formulating Ax=0, see H&Z ch.12

但并非任何时候 Ox与 O'x' 两个 ray 会相交：

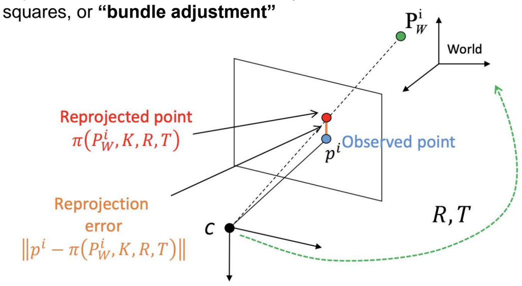


Ray's don't always intersect because of noise!!!

因此：⇒ Solve with non-linear least squares

Even if you do everything right, you will still be off because of noise, this is called the Reprojection Error

In practice with noise, want to directly minimize this with non-linear least squares, or **"bundle adjustment"**



Solve with non-linear least squares, iteratively

Summary: Two-view, known camera

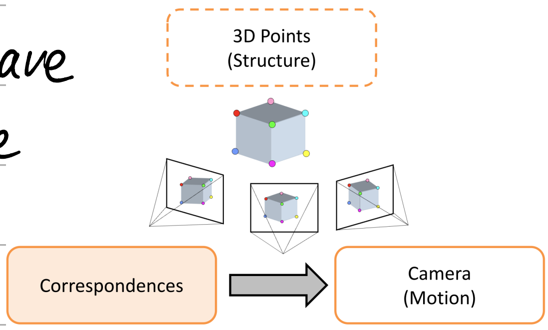
⇐ Summary

0. Assuming known camera intrinsics + extrinsics

1. Find correspondences:
   - Reduce this to 1D search with Epipolar Geometry!

2. Get depth:
   - If simple stereo, disparity (difference of corresponding points) is inversely proportional to depth
   - In the general case, triangulate.

Now consider if we have correspondence, can we estimate F?



$\mathbf{x} = (u, v, 1)^T, \quad \mathbf{x}' = (u', v', 1)$

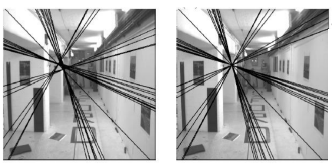
$$\begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0 \Rightarrow \begin{bmatrix} u'u & u'v & u' & v'u & v'v & v' & u & v & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

Solve homogeneous linear system using eight or more matches

$x'^T F x = 0$

⇐ Eight points cause if F' s.t. $x'^T F' x = 0$ then: $x'^T (cF') x = 0 \Rightarrow$ DOF: 8

Enforce rank-2 constraint (take SVD of $\boldsymbol{F}$ and throw out the smallest singular value)



$E = T_x R$

If we know E, we can recover t and R

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

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

The geometry of three views is described by a 3 x 3 x 3 tensor called the *trifocal tensor*

The geometry of four views is described by a 3 x 3 x 3 x 3 tensor called the *quadrifocal tensor*

After this it starts to get complicated…

Get the essential matrix with K (or some estimates of K) …

in practice you calibrate your cameras so you know K or have a very good estimate

$$E = K'^T F K.$$

Now if we have correspondence, how to calibrate?

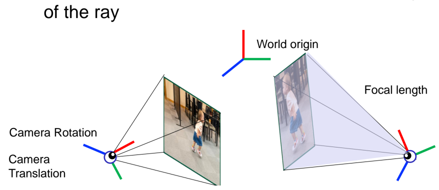
How to estimate the camera?

1. Estimate the fundamental/essential matrix!

2. Another method: Calibration

What are the camerea parameters?
- Extrinsics (R, T)
- Intrinsics (K)

How am I situated in the world + what is the shape of the ray



Approach two: Solve linear system!

$$\mathbf{x} = \mathbf{K}[\mathbf{R} \quad \mathbf{t}]\mathbf{X}$$

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

If we know the points in 3D we can estimate the camera!!

Can we factorize M back to K [R | T]?

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Yes.

Why? because K and R have a very special form:

intrinsic

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

QR decomposition

Practically, use camera calibration packages (there is a good one in OpenCV)

Solve for m's entries using linear least squares

Ax=0 form

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

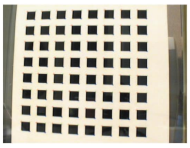
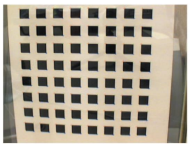
Similar to how you solved for homography!

Need at least 6 pairs of (3D coord, 2D image coord)

Inserting a 3D known object…

Also called "Tsai'scalibration" requires non-coplanar 3D points, is not very practical…

Modern day calibration uses a planar calibration target

Developed in 2000 by Zhang at Microsoft research

Doesn't plane give you homography?

Yes! If it's a plane, it's only a homography, so instead of recovering 3x4 matrix, you will recover 3x3 in Zhang's method

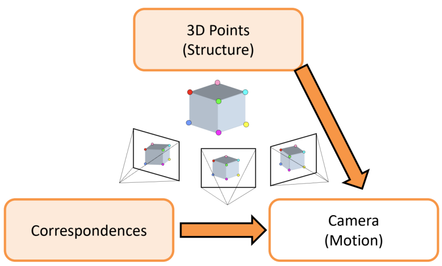
The 3x3 gives first two columns of R and T

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}$$

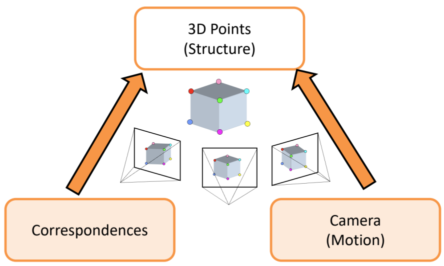
Moreover. if 3D points are in a plane, than it is :)

Lec 14: SfM (Structure from Motion)

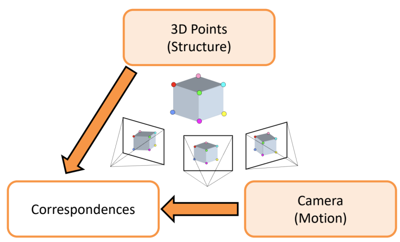
Camera Calibration; aka Perspective-n-Point



Stereo (w/2 cameras); aka Triangulation

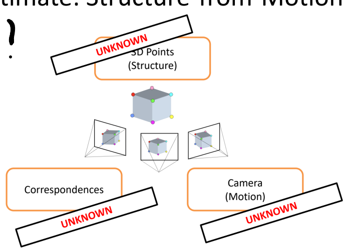


You can easily get correspondence via projection from 3D points + Camera



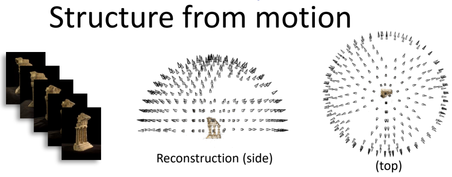
In the operations introduced before, they showed on to use triangle relationship. But what if none of these are known!

Ultimate: Structure-from-Motion



Start from nothing known (except maybe intrinsics), exploit the relationship to slowly get the right answer

Structure from motion



- Input: images with points in correspondence $p_{i,j} = (u_{i,j}, v_{i,j})$
- Output
  - structure: 3D location $\mathbf{x}_i$ for each point p_i
  - motion: camera parameters $\mathbf{R}_j$, $\mathbf{t}_j$ possibly $\mathbf{K}_j$
- Objective function: minimize *reprojection error*

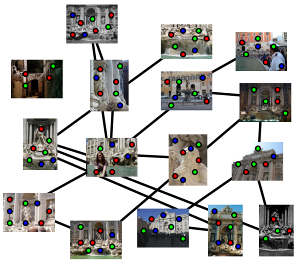
Given $P_{i,j}$*: 第i个点在第j个相机中的像素坐标

⇒ Structure & Motion

* Correspondence : Unknown !

Feature matching

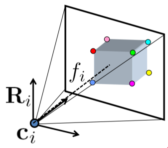
Match features between each pair of images



无对应关系？Recap on Stitching project
→ Feature extraction & Matching

Refine matching using RANSAC to estimate fundamental matrix between each pair

- Point: 3D position in space ($\mathbf{X}_j$)
- Camera (C_i):
  - A 3D position ($\mathbf{c}_i$)
  - A 3D orientation ($\mathbf{R}_i$)
  - Intrinsic parameters (**focal length**, aspect ratio, …)
  - 7 parameters (3+3+1) in total



- Minimize sum of squared reprojection errors:

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \underbrace{w_{ij}}_{\substack{\text{indicator variable:} \\ \text{is point } i \text{ visible in image } j\text{?}}} \cdot \Bigg\| \underbrace{\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)}_{\substack{\text{predicted} \\ \text{image location}}} - \underbrace{\begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix}}_{\substack{\text{observed} \\ \text{image location}}} \Bigg\|^2$$

Get optimized

$X_i.\ R_j.\ t_j$ in one optimization problem!

- Minimizing this function is called *bundle adjustment*
  - Optimized using non-linear least squares, e.g. Levenberg-Marquardt

Challenges:

- Large number of parameters (1000's of cameras, millions of points)
- Very non-linear objective function

- Important tool: Bundle Adjustment [Triggs *et al.* '00]
  - Joint non-linear optimization of both cameras and points
  - Very powerful, elegant tool
- The bad news:
  - Starting from a random initialization is very likely to give the wrong answer
  - Difficult to initialize all the cameras at once

The good news:

- Structure from motion with two cameras is (relatively) easy
- Once we have an initial model, it's easy to add new cameras

Idea:

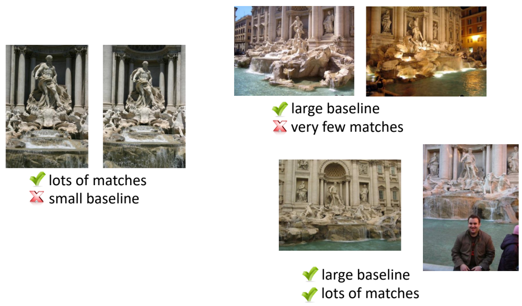
- Start with a small seed reconstruction, and grow

⇒ Incremental SfM:

Incremental SfM: Algorithm

1. Pick a strong initial pair of images
2. Initialize the model using two-frame SfM
3. While there are connected images remaining:
   a. Pick the image which sees the most existing 3D points
   b. Estimate the pose of that camera
   c. Triangulate any new points
   d. Run bundle adjustment

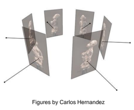
- We want a pair with many matches, but which has as large a baseline as possible



Strong:
Want many matches but want baseline as large as possible.

Multi-view Stereo (Lots of calibrated images)

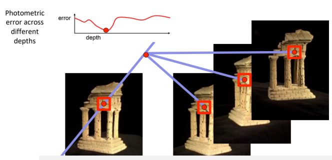
- Input: calibrated images from several viewpoints (known camera: intrinsics and extrinsics)
- Output: 3D Model



In general, conducted in a controlled environment with multi-camera setup that are all calibrated

The problem of SfM is that its output is sparse point cloud. With SfM's output of calibration information, can we form dense point cloud? ⇒ Multi-View stereo

Multi-view stereo: Basic idea



In this manner, solve for a depth map over the whole reference view

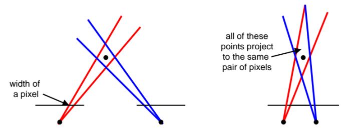
For an image pixel patch, consider ray through it with different depth, and see which depth most fit in other images best ⇒ Depth map

Multi-view stereo: advantages over 2 view

- Can match windows using more than 1 other image, giving a **stronger match signal**
- If you have lots of potential images, can **choose the best subset** of images to match per reference image
- Can reconstruct a depth map for each reference frame, and the merge into a **complete 3D model**

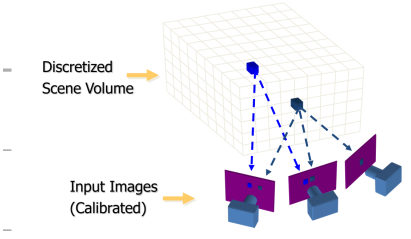
Source: Y. Furukawa

Choosing the baseline



- What's the optimal baseline?
  - Too small: large depth error
  - Too large: difficult search problem

Volumetric stereo



Goal: Assign RGB values to voxels in V *photo-consistent* with images

For 3D reconstruction, another approach: volumetric stereo

For every voxel, if projected on these cameras have minor error to gt, then this voxel remains, while others who don't satisfy this will be removed.

Lec 15 & 16 & 17: NeRF.

Problem Statement

⇐ What problem NeRF want to solve?

Input:
A set of calibrated Images

Output:
A 3D scene representation that renders novel views

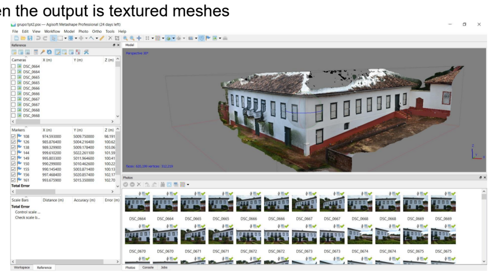
- Need to know the camera parameters: **extrinsic** (viewpoint) & **intrinsics** (focal length, distortion, etc)

Structure from Motion! (last lecture)

Input: set of images.
Output: extrinsics, intrinsics, 3D points, pixel correspondences

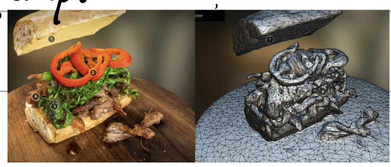
What was before NeRF? "Photogrammetry"

- Problem: Given calibrated cameras, recover highly detailed 3D **surface** model
- Often the output is textured meshes

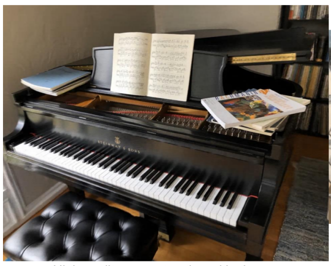


Before: how to image + calibration ⇒

⇐ Photogrammetry: Complicated!



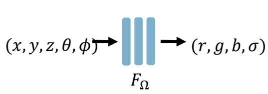
Advantage of NeRF



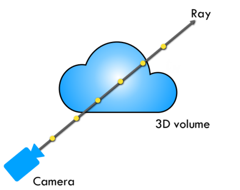
High quality reconstruction with view-dependent effects

Can represent non-opaque objects

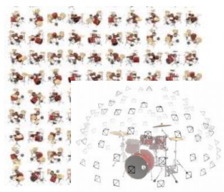
NeRF's Three Key Components



Neural Volumetric 3D Scene Representation



Differentiable Volumetric Rendering Function

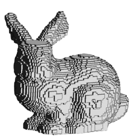


Optimization via Analysis-by-Synthesis

Original 3D representation ......

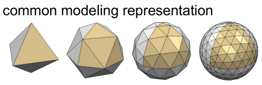
Volumetric representations

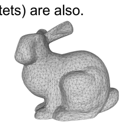
Model the *entire* space
Can be explicit (voxels) or implicit (NeRF)



Polygonal Meshes

- A mesh is a set of vertices with faces that defines the topology
- Mesh = {Vertices, Faces}
  - Vertices: N x 3
  - Faces: F x {3, 4, …} specifying the edges of a polygon
  - Triangle faces most common but tetrahedrons (tets) are also.
- **Surface** is explicitly modeled by the faces
- Most common modeling representation





Lightfield / Lumigraph

Levoy and Hanrahan, SIGGRAPH 1996
Gortler et al. SIGGRAPH 1996

光场/光栅图：有海量光线数据，通过计算和内插这些已记录的光线，来合成任意新视角的图像

- Previous approaches for modeling the Plenoptic Function
- Take a lot of pictures from many views
- Interpolate the rays to render a novel view

Stanford Gantry 128 cameras

Lytro camera

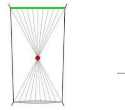
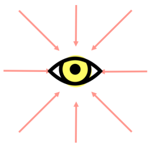
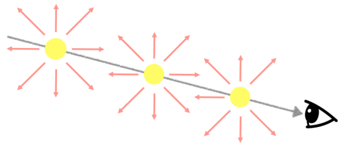


Figure from Marc Levoy

- These methods are called Image Based Rendering, because they literally interpolate the ray colors to make a new image
- i.e. no 3D information is recovered (you have to know the camera)



Plenoptic Function



NeRF

NeRF requires *integration* along the viewing ray to compute the Plenoptic Function

Bottom line: it models a 5D plenoptic function!

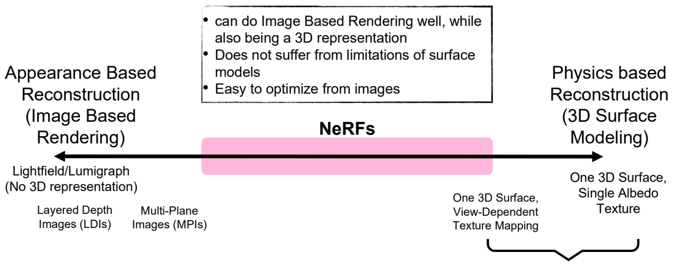
Density: Second key difference from lightfields, plenoptic function ✗

- Continuous probability density function (PDF) over "stuff"
- Connected to opacity: high density == very opaque, solid

$$\underbrace{(x,y,z}_{\text{Spatial location}},\underbrace{\theta,\phi)}_{\text{Viewing direction}} \rightarrow F_\Omega \rightarrow (\underbrace{r,g,b}_{\text{Output color}},\underbrace{\sigma}_{\text{Output density}})$$

Two difference between lightfield (NeRF) and plenoptic function.

Where NeRF stands



☆ "Analysis-by-Synthesis"

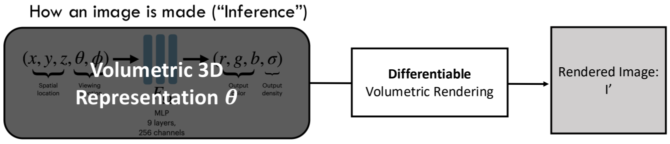
- Search for a world state from which you can explain many observations through synthesis
- In English: "If you understand (analyze) something, you can create (synthesize) it"
- (For NeRF): "If you really know what a scene looks like, you can render it from any view"
- (For Chemistry): "If you know how a molecule is structured, you can synthesize it from other molecules"
- Commonly used paradigm across CV!

* Core Function NeRF want to learn: For a point (x, y, z), if look through it via direction (θ, φ), what's its observered rgb value and its volumn density?

Training Strategy: Find a way to generate rgb for one ray with rgb & σ info as pred. *

"Neural Radiance Fields"

How an image is made ("Inference")

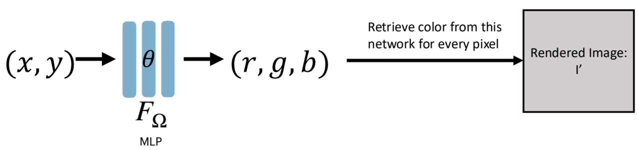


"Training" Objective (aka Analysis-by-Synthesis):

$$\min_{\theta} \| \text{Rendered Image: } I' - \text{Observed Image: } I \|_2$$

Toy setting: 2D. no φ & φ.

Let's simplify, do this in 2D:

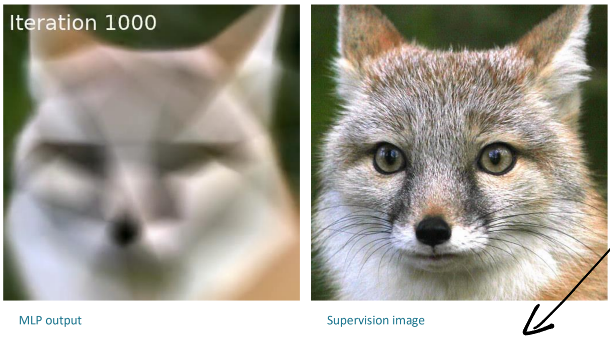


How to get MLPs to represent higher frequency functions?

Challenge observed: 1)

Optimize with "Training" Objective (aka Analysis-by-Synthesis):

$$\frac{\partial L}{\partial \theta} = \frac{\partial (rgb - rgb')}{\partial \theta} \qquad \min_{\theta} \| \text{Rendered Image: } I' - \text{Observed Image: } I \|_2$$

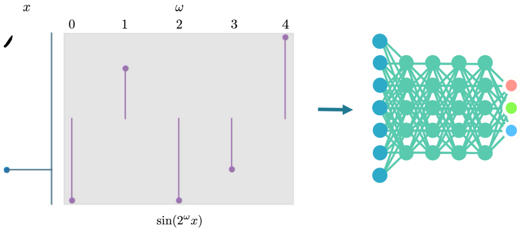


Why so ?

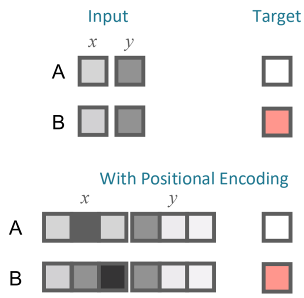
From example, we can see that: with slight diff in x, rgb can differ greatly! But if with PE, then 'slight diff in x' ⇒ distinct PE input

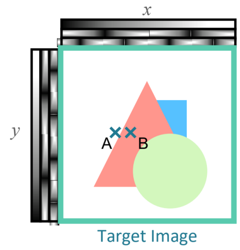
So it can respond high-freq change in rgb !

Positional Encoding



Fourier Features $\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \cdots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p))$





NeRF Network Architecture

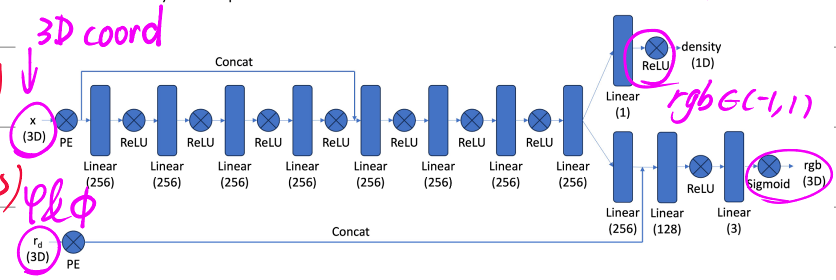
Now: the architecture:

① Prediction of σ doesn't depend on direction ϕ & φ ! (Intuitively Yes)

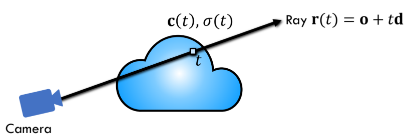
② 'Coord & Direction' inputs both undergo PE.

③ ReLU or Sigmoid ? Check desired output range!

Next section you will implement this:



Volumetric formulation for NeRF



at a point on the ray $\mathbf{r}(t)$, we can query color $\mathbf{c}(t)$ and density $\sigma(t)$

How to integrate all the info along the ray to get a color per ray?

Summary

for a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

differentiable w.r.t. $\mathbf{c}, \sigma$

$$\mathbf{c} \approx \boxed{\sum_{i=1}^{n} w_i \mathbf{c}_i = \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i}$$

weights colors

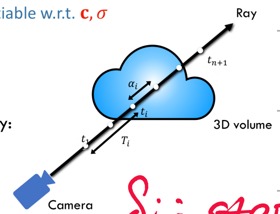
How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

δ_i: stepsize

How much light is contributed by ray segment i:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$



δ_i: stepsize.

Remember, expected color is equal to

$$\int T(t)\sigma(t)\mathbf{c}(t)dt \approx \sum_i T_i \alpha_i \mathbf{c}_i = \sum_i w_i \mathbf{c}_i$$

$T(t)\sigma(t)$ and $T_i\alpha_i$ are "rendering weights" — probability distribution along the ray (continuous and discrete, respectively)

You can also render entities other than color in 3D, for example it's depth, or any other N-D vector $\boldsymbol{v}_i$

Volume rendered "feature" $= \sum_i w_i \boldsymbol{v}_i$

*Finally, how to pred rgb of one ray (one pixel) with rgb & σ of points along the way?

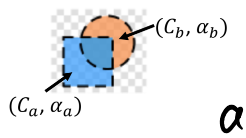
T_i: The prob that ray first hit c_i

σ_i: logit ⇒ $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$

The formula of Volumetric Rendering actually originate from alpha blending:

Alpha Blending

for two image case, A and B, both partially transparent:



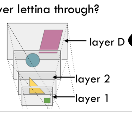
$$I = C_a \alpha_a + C_b \alpha_b (1 - \alpha_a)$$

How much light is the previous layer letting through?

⇐ $I = C_a \alpha_a + C_b (1 - \alpha_a) \alpha_b$

how much light left to pass through.

General D layer case: $I = \sum_{i=1}^{D} C_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$



$$I = \sum_{i=1}^{D} C_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

When transfered to Volumetric Rendering: for differentiable : $\sigma_i \to \alpha_i$:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$

So: $C = \sum_{i=1}^{n} C_i \alpha_i \prod_{j=1}^{i-1} (1-\alpha_j)$

Summary

for a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

differentiable w.r.t. $\mathbf{c}, \sigma$

$$\mathbf{c} \approx \boxed{\sum_{i=1}^{n} w_i \mathbf{c}_i = \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i}$$

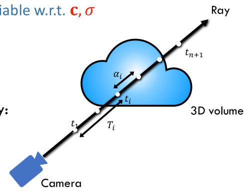
weights colors

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1-\alpha_j)$$

How much light is contributed by ray segment i:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$



Hierarchical Sampling vs. Acceleration Structures

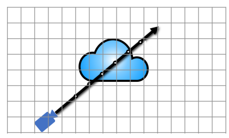
Hierarchical Sampling

Iteratively use samples from NeRF to more efficiently sample visible scene content

Acceleration Structures

Distill/cache properties of NeRF into a structure that helps generate samples: e.g. Occupancy Grids

Straightforward compute —> storage tradeoff



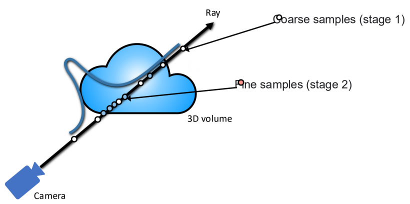
Now: Some bells & whistles:

How to sample reasonable? Intuitively, most of the space can be vacuum!

Key Idea: sample points proportionally to expected effect on final rendering

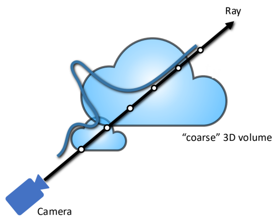
$$C \approx \sum_{i=1}^{N} \boxed{T_i \alpha_i} c_i$$

treat weights as probability distribution for new samples



What about aliasing during coarse sampling?

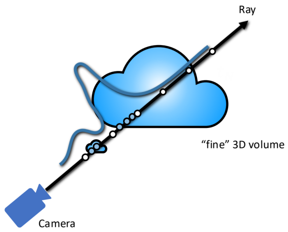
Solution: train two NeRFs! —> lower resolution for first "coarse" level



Solution: Hierarchical Search

What about aliasing during coarse sampling?

Solution: train two NeRFs! —> higher resolution for second "fine" level

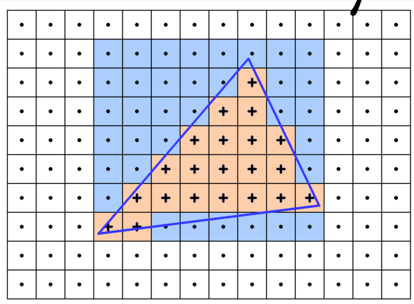


Coarse → Fine, and use two NeRFs!

Rasterization = conversion of primitives to pixels (details in CS184)

Rasterizat

Blending (+ Sl



Last topic: Gaussian Splatting

Preliminary: About Rasterization (光栅化): Instead of casting rays, we Map Object to pixels.

Differentiable Gaussian Rendering

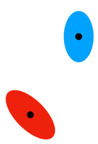
What is the representation of a 3D Gaussian?

How to project to 2D and rasterize?

How to model/aggregate appearance?

Position $\mathbf{p}$

$$\mathcal{G}_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T \mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}$$

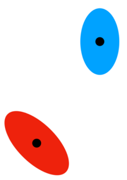


$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

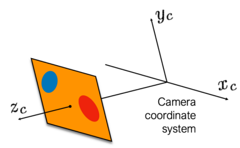
Factorize as scale and rotation: $\mathbf{V} = RSS^TR^T$

$R \in SO(3)$

Each Gaussian also has an opacity and view-dependent color (via SH coefficients): $\alpha, \mathbf{c}$



$\mathbf{p}', R', S$



$\pi(\mathbf{x}) = \mathbf{u}$

$$z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

π: Projection function for mapping 3D points to pixels

2D mean: $\mu_{2D} = \pi(\mu_{3D})$

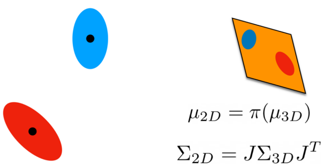
2D covariance:

$$J = \frac{\partial \pi}{\partial \mathbf{x}}(\mu_{3D})$$

$$\Sigma_{2D} = J\Sigma_{3D}J^T$$

Q: What is the image-space projection of a 3D Gaussian?

A: Can approximate as a 2D Gaussian!



$\mu_{2D} = \pi(\mu_{3D})$

$\Sigma_{2D} = J\Sigma_{3D}J^T$

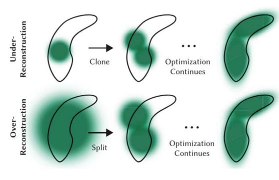
1. Sort Gaussians from closest to furthest from the camera

2. For each pixel $\mathbf{u}$, compute opacity for each gaussian $\mathcal{G}_k$:

$$\bar{\alpha}_k = \alpha_k \frac{e^{-(\mathbf{u}-\mu_{2D}^k)^T(\Sigma_{2D}^k)^{-1}(\mathbf{u}-\mu_{2D}^k)}}{2\pi|\Sigma_{2D}^k|^{0.5}}$$

+ Lots of efficient GPU optimization strategies

Initialize with sparse point cloud from SfM



Split/clone Gaussians based on heuristics

A glance at Gaussian Splatting.

Finally, 'holy grail': Dynamic Scene & Using prior knowledge

Lec 18: Texture

Instance (实例) v.s. Category (门类)

Recognition: Instance vs. Category

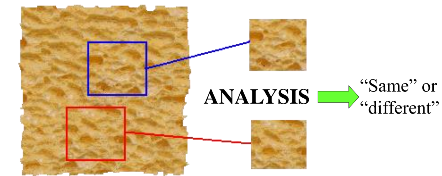
- Instance recognition:
  - "Find me this particular chair again"
  - Often simple template matching works OK
    - Even better with many small templates, a.k.a. feature descriptors
- Category recognition:
  - "find me all chairs"
  - Templates don't work. Why?
  - Focus on things that might be invariant across the category
  - Relates to concept of "texture"

- Texture depicts spatially repeating patterns ⇒ Texture (纹理) ⇐ Def
- Many natural phenomena are textures

E.g. For two places on the right: Should identify as 'same'

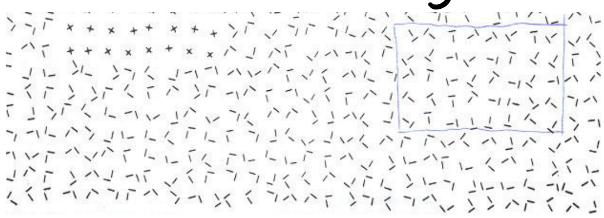
radishes rocks yogurt

Texture Analysis



True (infinite) texture

Compare textures and decide if they're made of the same "stuff".



Human vision is sensitive to the difference of some types of elements and appears to be "numb" on other types of differences.

Human vision operates in two distinct modes:

1. **Preattentive vision**
   parallel, instantaneous (~100--200ms), without scrutiny, independent of the number of patterns, covering a large visual field.
2. **Attentive vision**
   serial search by focal attention in 50ms steps limited to small aperture.

★ Julesz Conjecture:

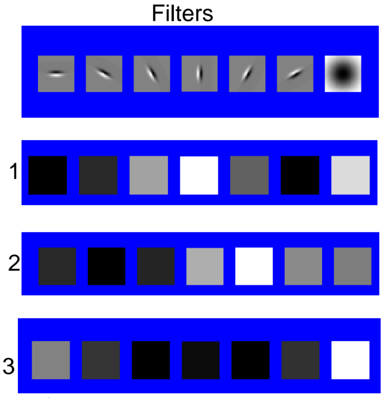
*Textures cannot be spontaneously discriminated if they have the **same first-order and second-order statistics** of texture features (textons) and differ only in their third-order or higher-order statistics.*

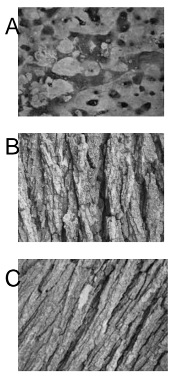
Two questions of texture modeling

- What are the texture features (textons)? 纹理基元
  - Pixels
  - Pixel patches
  - Outputs of V1-like filters
  - Clusters of patches / filter outputs
  - CNN features
  - Etc.
- How do we aggregate statistics
  - Various types of histograms
  - Implicit or explicit

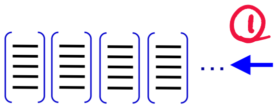
We can view texton as filter*. Conv will indicate how image respond to this texton

Filters





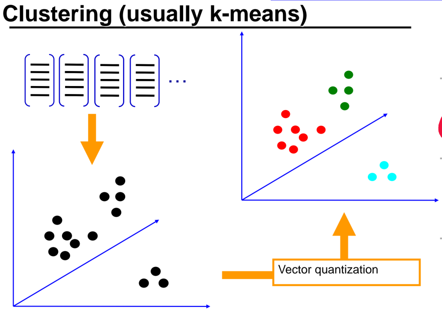
Patch Features



①

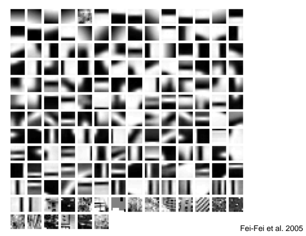
Scene Classification △

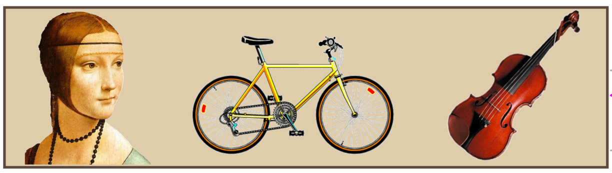
Clustering (usually k-means)



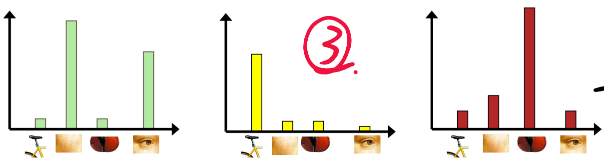
②

Clustered Image Patches ("Bag of Visual Words")





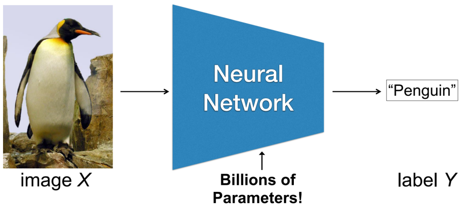
* Brain Structure has similarities with this

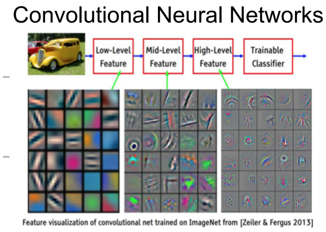


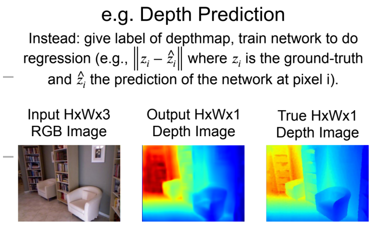
③

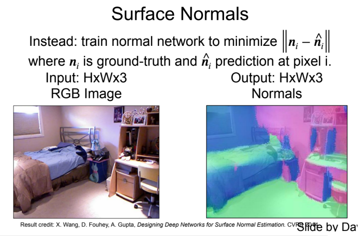
△: Method: Patch Feature① → K-means to form '视觉字典'② → 图像表示为视觉单词频率直方图③

Lec19: Image to image translation







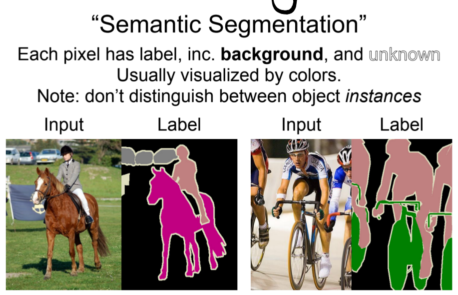


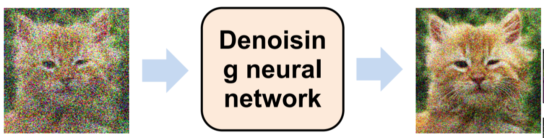
Neural Network 现广泛应用于CV，其中一类任务，task中：

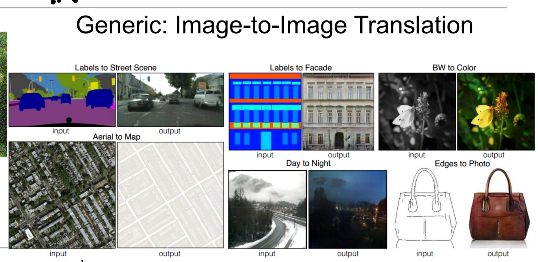
$R^{H\times W\times 3/1} \rightarrow NN \rightarrow R^{H\times W\times ?}$, i.e., 给每个pixel分配一个feature.

Generic Task: Image to image translation

如 Depth Prediction / Surface Normal Task / Denoise / Semantic Segmentation / ……，分别给 R^1, R^3, R^3, $R^{\text{num of class}}$





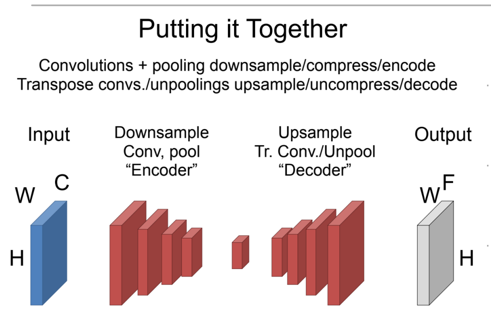


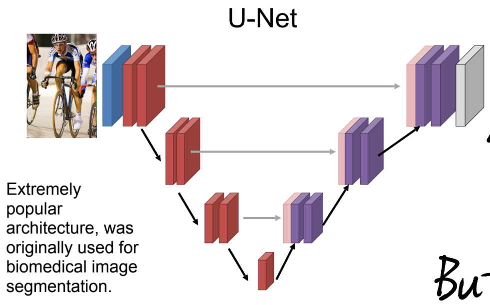
设计网络时，使用卷积为主要操作。但感受野随深度↑才↑，而深度↑ compute↑

We need to:
1. Have large receptive fields to figure out what we're looking at
2. Not waste a ton of time or memory while doing so

These two objectives are in total conflict

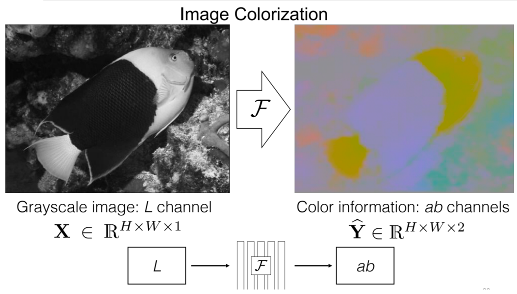
⇒ Conflict! How to solve it?

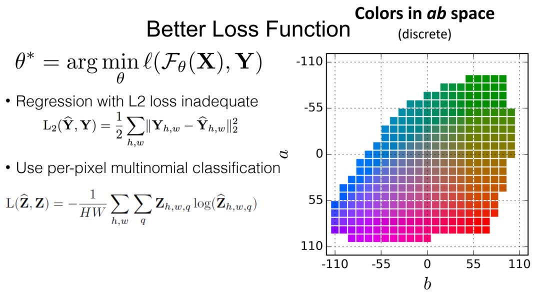




Better way: Novel Architecture: Unet!

But, e.g., in image colorization.

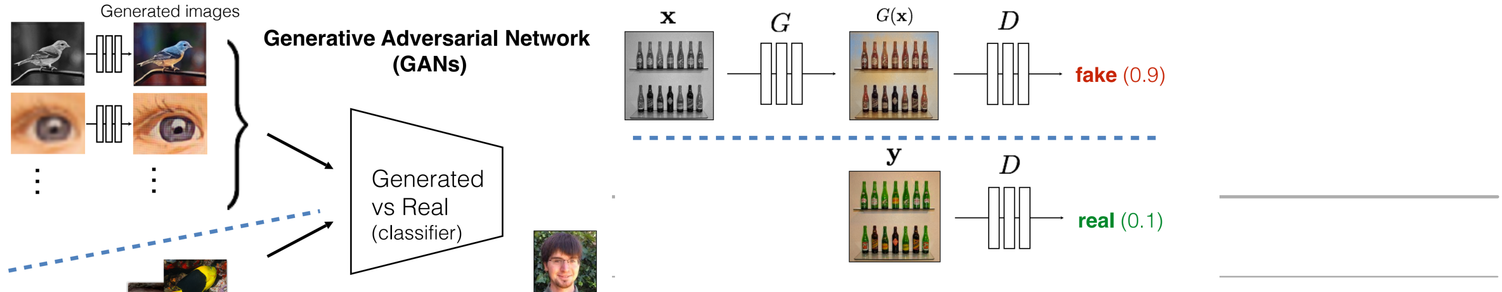




Loss design is critical.

Question: Any universal loss?

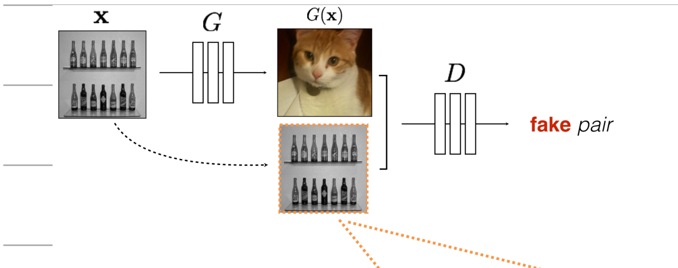
有的。Core idea: teacher-student. 'teacher' component as scaffolding.



$$\arg\max_{D} \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \boxed{\log D(G(\mathbf{x}))}\ +\ \boxed{\log(1 - D(\mathbf{y}))}\ ]$$

G tries to synthesize fake images that ***fool*** the ***best*** **D**:

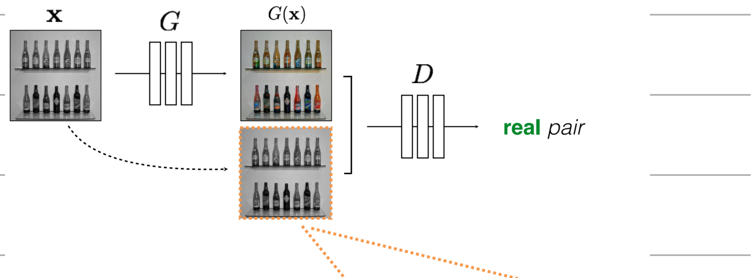
$$\arg\boxed{\min_{G}}\boxed{\max_{D}} \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(G(\mathbf{x}))\ +\ \log(1 - D(\mathbf{y}))\ ]$$



$$\arg\min_{G}\max_{D} \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(\boxed{\mathbf{x}}, G(\mathbf{x})) + \log(1 - D(\boxed{\mathbf{x}}, \mathbf{y}))\ ]$$

G's perspective: **D** is a loss function.

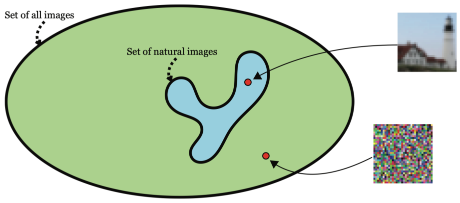
Rather than being hand-designed, it is *learned*.



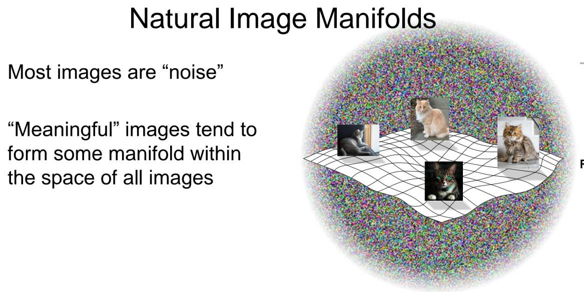
$$\arg\min_{G}\max_{D} \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(\boxed{\mathbf{x}}, G(\mathbf{x})) + \log(1 - D(\boxed{\mathbf{x}}, \mathbf{y}))\ ]$$

Also, we need 'pair'. Why? 这种机制称为 Conditional GAN.
判别器不仅要看输出图像是否逼真，还要看Y是否与输入X匹配

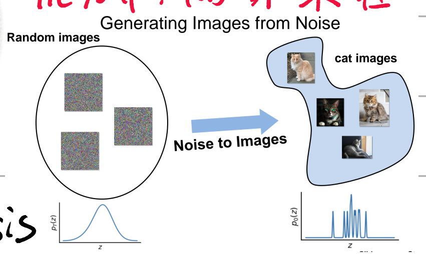
Lec 20: Generative Models of Images

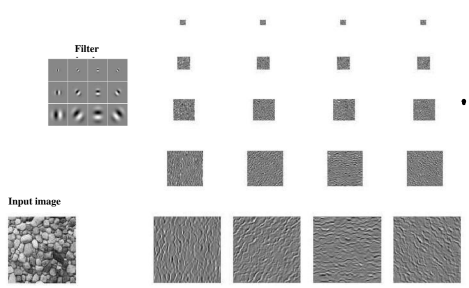


Image中 pixel 组合与 value 取值所造成的 permutations 繁多, 但其中很小一部分图片, 我们才认为是'有意义的'图像.



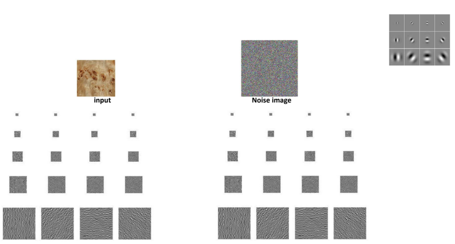
认为有意义图像在巨大图像空间中并非随机分布，而聚集在一个特定的低维'流形'上



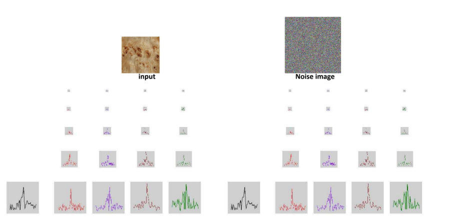


从 Parametric Texture Synthesis 开始：最早用多尺度滤波器(金字塔)来合成

Step 1: Convolve with filterbank



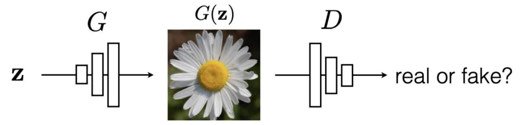
Step 2: match per--channel histograms



Step 3: collapse pyramid and repeat!

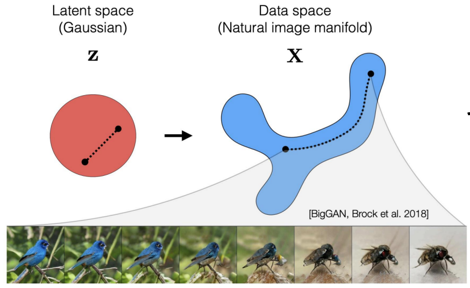
GANs as generative models

- **G** tries to synthesize fake images that ***fool*** the ***best*** **D**
- **D** tries to identify the fakes



$$\arg\min_{G}\max_{D}\ \mathbb{E}_{\mathbf{z},\mathbf{x}}[\ \log D(G(\mathbf{z}))\ +\ \log(1-D(\mathbf{x}))\ ]$$

GANs can "walk" on the manifold

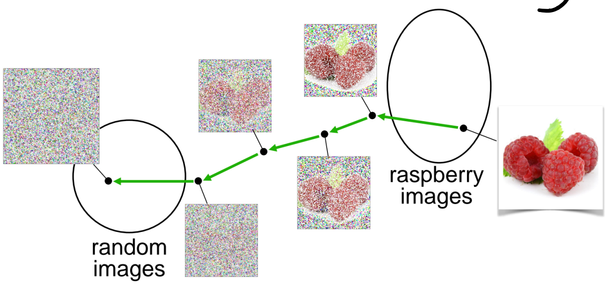


之后有CNN-based方式，甚至可实现style transfer类任务。

之后GAN盛行。

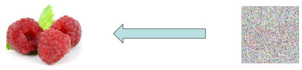
Diffusion: (Recently Popular)

noise→image: hard; image→noice: easy.

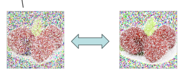


Key insight

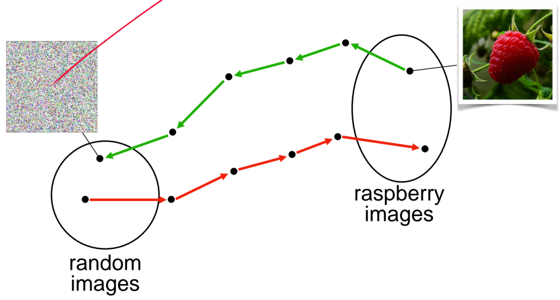
- Globally, creation is much harder than destruction



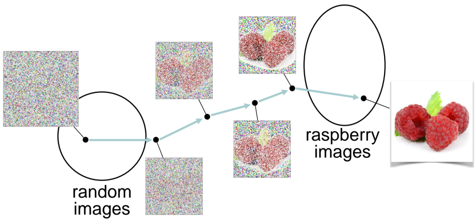
- But locally, they are almost reversable!

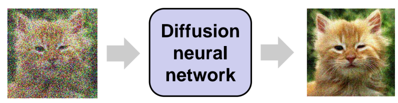


但局部它们加噪过程又可逆！用NN预测噪声，然后去噪



Denoising diffusion neural network





This network can be a U-Net or other suitable image-to-image network

Curious property of Diffusion

+We are training the model to reconstruct the training set
  - +But it fails!
  - +Instead, it generate novel images
  - +Which is what makes it great

+Perhaps it models images as textures
  - +Keeping important correlations and throwing away the rest
  - +But we don't know the "model space" of these textures

Denoising: train a U-net, learn to 一步步去噪。虽然训练目标是"重建"训练数据(去噪)，但模型最终学会的并非死记硬背。

Diffusion可与LLM结合，w/ 文字控制生成过程。

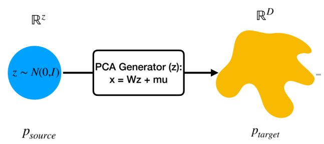
Lec21: Flow Matching

Generative Story

通常生成范式: latent space $\xrightarrow{\text{sample}}$ Images

- Any Generative Model has a process of sampling an image
- For ex, here's the generative story for PCA in its probabilistic interpretation:

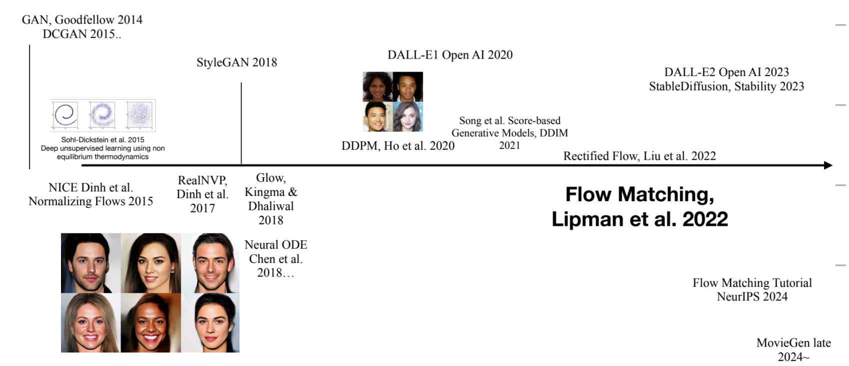
1. Sample from a Gaussian Distribution
   $z \sim N(0, I)$
2. Project to Images (W = Eigenvectors, Mu = avg datapoint)
   $x = Wz + \mu$



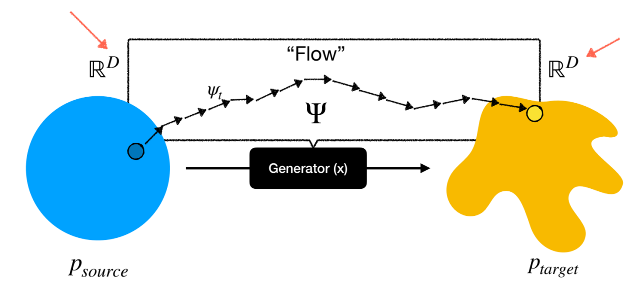
- GANs really opened up the possibility of image generation
- But people didn't like it for many reasons
  - Severe mode collapse
  - Unstable training mechanics
- Flow/Diffusion is a reactionary movement against GANs, next natural evolution

"No GAN Movement"

History



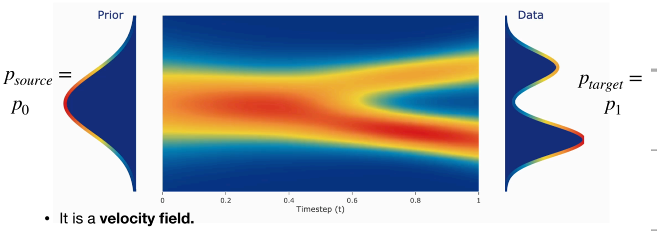
Flow based Generative Models



1. Latent space dim is same as the target!
2. Takes T steps to go from src to tgt

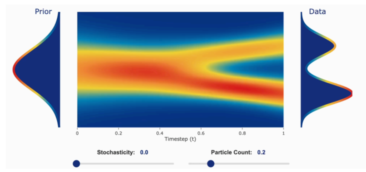
但 Flow base model 不追求 sample directly 而是 flow.

What is Flow?



- It is a **velocity field.**
- It's like a river with some currents, every point defines how fast you move (velocity)
- You ride this river to go from one distribution to next

Riding the river = Integration



Simplest "Euler Integration":
$x_{t+\Delta t} = x_t + v_\theta(x_t, t)\Delta t$

- Riding this rive means you add little bits of velocity defined at each location
- This is called "Integration", also called solving the Ordinary Differential Equation (ODE) with initial state x_0, through some differential parametrized by a network: $\frac{dx}{dt} = v_\theta(x, t)$
- You can add stochasticity when riding it, then it becomes SDE (more next lecture)

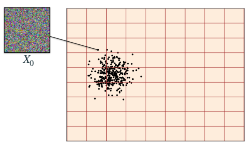
希望学习一个速度场 $v_\theta(x,t)$，通过在这个速度场中"漂流"积分，我们可以将噪声分布（源）平滑地变换为复杂图像分布。

The flow can be though about learning a warping function

like, gaussian noice

$X_t = \psi_t(X_0)$, $t \in [0,1]$

Warping   Source $X_0 \sim p$

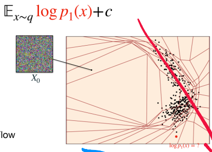


Initial approach trained flow with Maximum Likelihood

$D_{KL}(q \,\|\, p_1) = -\mathbb{E}_{x\sim q} \log p_1(x) + c$

$X_t = \psi_t(X_0)$, $t \in [0,1]$

Warping   Source $X_0 \sim p$



- Normalizing Flow, Continuous Normalizing Flow
- Chaining ψ_t needs to satisfy the continuity equation!!!!
- This requires ODE integration DURING training with invertible neural networks

早期传统 Normalizing Flow（归一化流）用最大似然训练 让P_1分布尽可能贴合真实分布q

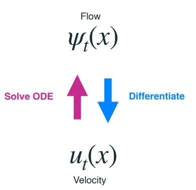
训练、极其缓慢

Previous Normalizing Flow works

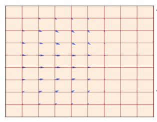
Caveat

- Tries to directly deal with this continuity equation constraint
- Very slow to train (need to integrate while training)
- Other constraints like invertibility of ψ_t
- Nice idea with promising results but limited capability + not practical to train

Instead, model Flow with Velocity



$\frac{d}{dt}\psi_t(x) = u_t(\psi_t(x))$



- **Pros**: velocities are ***linear***
- **Cons**: simulate to sample

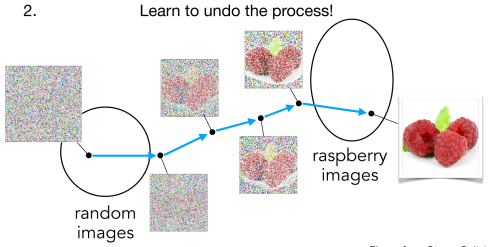
不学位置映射 $\psi_t(x)$，而学速度场 $u_t(x)$：不问"下一刻我在哪"

而问"我现在往哪个方向走，且速度多少"

In Flow approach:

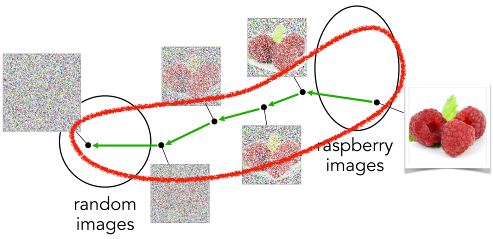
Training

1. Take real data, corrupt it to left distribution somehow
2. Learn to undo the process!



$$$ question, how to pick the intermediate path?

How to generate this Green path?



How to construct x_t

TLDR: Sample noise, add it, then reconstruct the data

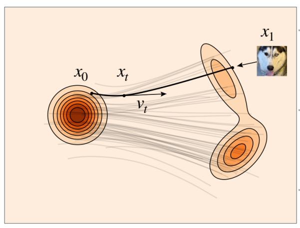
Flow matching says you can **pick any combination**, as long as it starts from a sample in the source (e.g. gaussian) and ends with a sample in the target distribution (image)

$$x_t = \alpha_t x_0 + \sigma_t x_1$$

$x_0 \sim p_0(x)$ $x_1 \sim p_1(x)$

Flow training

- For each data x_1
  - Sample some noise x_0
  - Combine it however you want to get x_t
  - Now learn to predict the velocity at x_t
    - What is the velocity? Depends on how you got x_t



What is the velocity supervision?

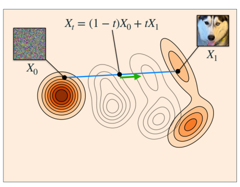
$x_t = \alpha_t x_0 + \sigma_t x_1$

$$x_t = (1-t)x_0 + t x_1$$

$$\frac{dx_t}{dt} = -x_0 + x_1$$

$$= x_1 - x_0$$

$\mathbb{E}_{t,x_0,x_1} \left\| u_t^\theta(X_t) - (X_1 - X_0) \right\|^2$



*conditioned on a single sample

理想速度就是 $x_1 - x_0$. 因此

loss: $\mathbb{E}_{t,x_0,x_1} \| u_t^\theta(X_t) - (X_1 - X_0) \|^2$

Inside a Training Loop

Flow Matching

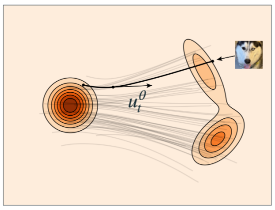
```
x = next(dataset)
t = torch.rand(1) # Sample timestep (0,1)
noise = torch.randn_like(x) # Sample noise
x_t = (1-t) * noise + (t) * x # Get noisy x_t

flow_pred = model(x_t, t) # Predict noise in x_t
flow_gt = x - noise # ground truth flow (w/ linear sched)
loss = F.mse_loss(flow_pred, flow_gt) # Update model
loss.backward()
optimizer.step()
```

During inference:

- Just take a small step in the velocity
- Use any ODE Solver, i.e. integration you like, like Euler integration:

$$x_{t+\Delta t} = x_t + \Delta t \cdot \frac{dx}{dt}\Big|_{x_t,t}$$



Sample from $X_0 \sim p$

Training: Model parameterization

- You can make your network output undo the noise in many different ways, predicting x, v, noise, or flow

$$v_t = \alpha_t x_1 - \sigma_t x_0 \qquad u_t = x_1 - x_0 = \epsilon - x_0$$
$$u_t = \dot{x}_t - x_0$$

- These are all equivalent because of the linear relationship with x_t. You can derive all of these as long as you know one of them

$$x_t = \alpha_t x_0 + \sigma_t x_1$$

- For example

$$\mathbb{E}[(\hat{\mathbf{x}}_0 - \mathbf{x}_0)^2] = \mathbb{E}\left[\left(\frac{\mathbf{x}_t - \sigma(t)\hat{\varepsilon}}{\alpha(t)} - \frac{\mathbf{x}_t - \sigma(t)\varepsilon}{\alpha(t)}\right)^2\right] = \mathbb{E}\left[\frac{\sigma(t)^2}{\alpha(t)^2}\left(\hat{\varepsilon} - \varepsilon\right)^2\right].$$

Training: Flow Matching vs. Diffusion

Algorithm 1: Flow Matching training.

Input : dataset q, noise p
Initialize v^θ
while *not converged* **do**
- $t \sim \mathcal{U}([0,1])$ ▷ sample time
- $x_1 \sim q(x_1)$ ▷ sample data
- $x_0 \sim p(x_0)$ ▷ sample noise
- $x_t = \Psi_t(x_0|x_1)$ ▷ conditional flow
- Gradient step with $\nabla_\theta \|v_t^\theta(x_t) - \dot{x}_t\|^2$

Output: v^θ

Algorithm 2: Diffusion training.

Input : dataset q, noise p
Initialize s^θ
while *not converged* **do**
- $t \sim \mathcal{U}([0,1])$ ▷ sample time
- $x_1 \sim q(x_1)$ ▷ sample data
- $x_t = p_t(x_t|x_1)$ ▷ sample conditional prob
- Gradient step with $\nabla_\theta \|s_t^\theta(x_t) - \nabla_{x_t} \log p_t(x_t|x_1)\|^2$

Output: v^θ

$p_t(x_t|x_1)$ general
$p(x_0)$ is general

$p_t(x_t|x_1)$ closed-form from of SDE $dx_t = f_t dt + g_t dw$
- *Variance Exploding*: $p_t(x|x_1) = \mathcal{N}(x|x_1, \sigma_{1-t}^2 I)$
- *Variance Preserving*: $p_t(x|x_1) = \mathcal{N}(x|\alpha_{1-t}x_1, (1-\alpha_{1-t}^2)I)$, $\alpha_t = e^{-\frac{1}{2}T(t)}$

$p(x_0)$ is Gaussian
$p_0(\cdot|x_1) \approx p$

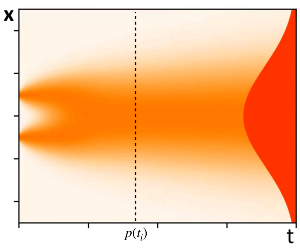
Algorithm: Flow Matching: 采样一个 noise x_0 与 data x_1，强制中间状态 x_t 在两点连线上，而训练目标：$v_\theta(x_t) \to x_1 - x_0$

Diffusion: 采样 x_1，依 $p_t(x_t|x_1)$ 随机加噪得 x_t

训练目标：预测噪声，即 $\nabla \log p_t$

Lec22: Diffusion Sampling

Revisit diffusion models with a 1D example



t: Miika Aittala

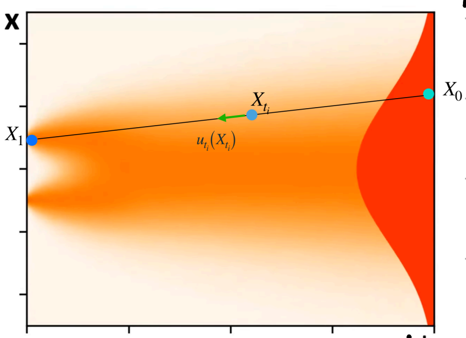
Recall in 1D example: how to: noise → distribution?

We need to know the 'speed vector', i.e., 'flow', of $X_{t_i} \Rightarrow u_{t_i}(X_{t_i})$. We may use a model to represent this: $u_{t_i}^\theta(X_{t_i})$

So objective: $\mathbb{E}_{t,X_0,X_1}\|u_{t_i}^\theta(X_{t_i}) - u_{t_i}(X_{t_i})\|_2$

Training Objective:

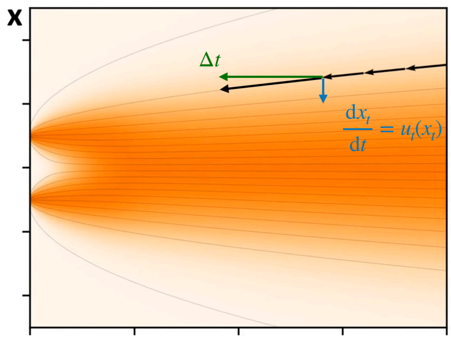
$$\mathbb{E}_{t,X_0,X_1}\left\|u_{t_i}^\theta(X_{t_i}) - u_{t_i}(X_{t_i})\right\|^2$$



Solving the flow ODE with discretization

Euler step:

$$x_{t+\Delta t} = x_t + \Delta t \cdot \frac{dx}{dt}\Big|_{x_t,t}$$



With this model who can give $\frac{dx}{dt}\Big|_{x_t,t}$

We can flow in a way:

$$x_{t+\Delta t} = x_t + \Delta t \cdot \frac{dx}{dt}\Big|_{x_t,t}$$

Truncation error: fails to approximate ideal trajectory by finite steps.

1. Naive solution: sampling with more steps.
2. Time steps are long at high noise levels and short at low noise levels
3. Higher-order ODE solver

This is not a perfect approach

⇐ 离散化 can be problematic

Model may give $\frac{dx}{dt}\Big|_{x_t,t}$ ⇒

which is a little bit inaccurate

Model fails to approximate the marginal flow.

1. Stochastic sampler (SDE) injects fresh noise throughout the evolution in addition to reducing the noise.

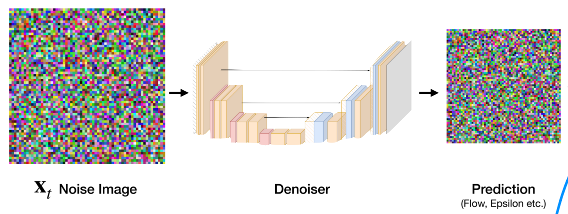
该 sol 很巧妙!

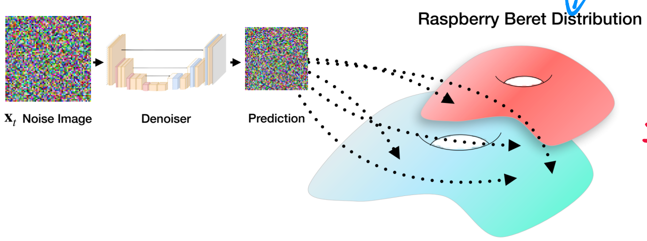
"给模型一些迂回的余地!" x_{t+1} $\hat{x}_t$ x_t

Diffusion!

A specific implementation of 'flow' idea

Diffusion Guidance
Motivation





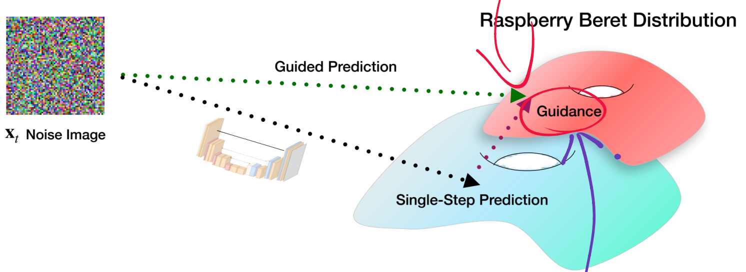
Unguided Diffusion Isn't Too Useful!

Natural Image Distribution

Without condition injection and we want conditioned generated image, 生成 image distribute over entire natural image space!

⇒ We want to inject a guidance ......

Diffusion Guidance
Push Toward a Conditional Mode



Natural Image Distribution

How? Two approaches.

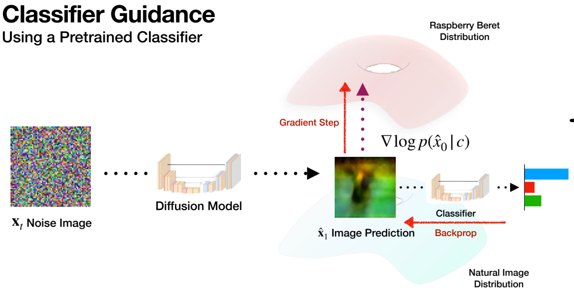
Original	Current
Classifier Guidance	**Classifier-Free Guidance**
Guide with a pretrained classifier.	Guide a diffusion model with itself.

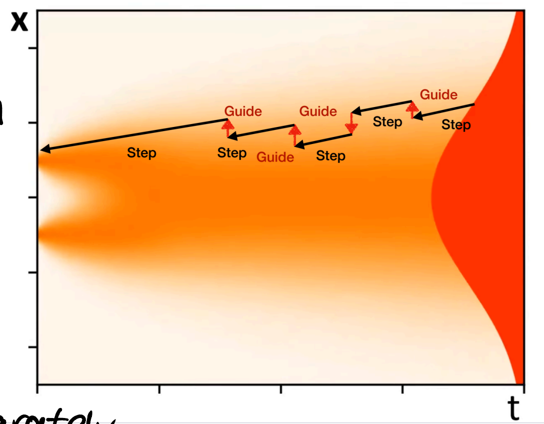
How do we do this?

Classifier Guidance

For a generated step image, use a classifier to generate guidance vector.

Classifier Guidance
Using a Pretrained Classifier

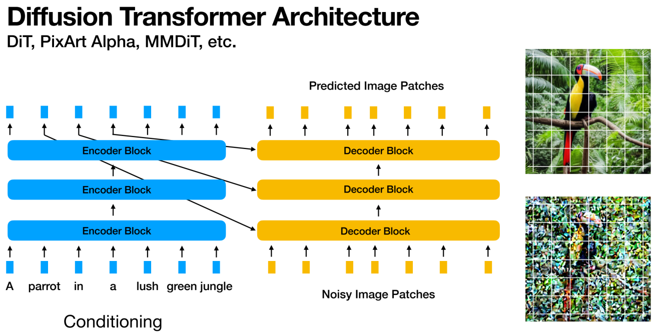




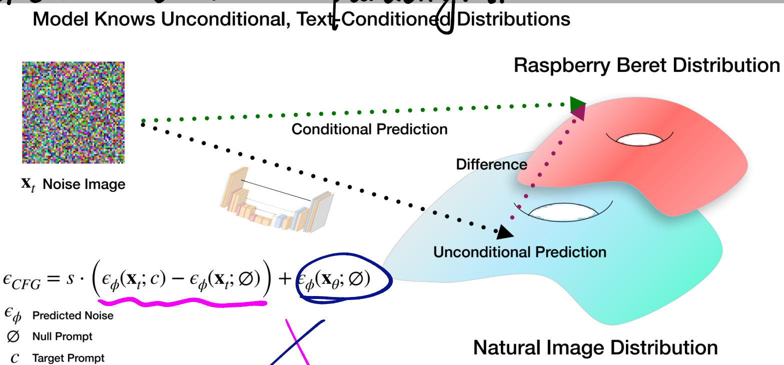
But this approach is not favored: ① Images in the middle: OOD for classifier, so guidance vector may not be reliable!

② Train these two models jointly or separately are both trouble-some! (Hard to train)

Classifier-Free Guidance.



Approach 1: With Transformer's encoder-decoder paradigm.



$$\epsilon_{CFG} = s \cdot \left(\epsilon_\phi(x_t; c) - \epsilon_\phi(x_t; \varnothing)\right) + \epsilon_\phi(x_\theta; \varnothing)$$

ϵ_ϕ Predicted Noise
$\varnothing$ Null Prompt
c Target Prompt
s CFG Scale

Approach 2: Use UNet

$\epsilon_\phi(\cdot\,;\cdot)$ image. condition

At one step, generate both: $\epsilon_\phi(x_t; c)$ & $\epsilon_\phi(x_t; \varnothing)$

Their gap can be a guidance!

$\epsilon_\phi(x_t; c)$

guidance

$\epsilon_\phi(x_t; \varnothing)$.

And amplify $\epsilon_\phi(x_t; c) - \epsilon_\phi(x_t; \varnothing)$

让生成图片强烈地被推向 guidance 方向.