Introduction to Computer Vision Final

Haochen Zhao, PKU YPC

Part 1: Temporal Analysis

1. Motion and Optical Flow

Definition: optical flow is the apparent motion of brightness patterns in the image
Goal: recover image motion at each pixel from optical flow
Key assumptions:

1. Brightness constancy
   1. **Brightness Constancy Equation**: $I(x,y,t-1) = I(x+u(x,y), y+v(x,y), t)$
2. Small motion
   1. **Taylor expansion**: $\nabla I \times [u,v]^T = 0$
   2. **The Aperture Problem (Ambiguity)**: If (u,v) satisfies the equation, so does $(u+u', v+v')$, if $\nabla I \times (u', v') = 0$
3. Spatial coherence
   1. Assume the pixel's neighbors have the same (u,v)
   2. If we use 5x5 window, then gives 25 equations per pixel

Lucas-Kanade Flow
$[\, I_x(p1) \quad I_y(p1) \; \ldots \quad \ldots \; I_x(p25) \quad I_y(p25) \,] [\, u \; v \,] = -[\, I_t(p1) \; \ldots \; I_t(p25) \,]$, $A_{25\times 2} d_{2\times 1} = b_{25\times 1}$
Overconstrained linear system: least squares solution:
$(A^T A)d = A^T b$ Eigenvalues of $A^T A$: λ_1, λ_2, Harris corner detector

FlowNet: Learning Optical Flow with CNN

2. RNN

2.1 Concepts

Process Sequential Data:

1. one2one: Vanilla NN
2. one2many: Image Captioning
3. many2one: action prediction
4. many2many: Video Captioning

Key Idea: internal state (hidden state) **Recurrent Formula**
RNN Hidden State Update: recurrence formula
$h_t = f_W(h_{t-1}, x_t)$
h: state, W: paras, f: func, x: input vector
RNN Output $y_t = f_{W_{hy}}(h_t)$ $f_{W_{hy}}$: another function with paras
notice: the same function and same set of parameters are used at every time step

2.2 Vanilla RNN

$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)\ y_t = W_{hy}h_t$

Char-level Language model:
input_layer -W_{xh}- hidden_layer -W_{hy}- output_layer -- target chars

Backpropagation through Time Run forward and backward through chunks of the sequence instead of whole sequence **Carry hidden states forward in time forever, but only backprop for some smaller number of steps** sequence length, longer term relationship will not be learned

Different Sampling Strategies

- Greedy sampling
- Weighted sampling
  - Exhaustive Search
  - Beam Search

RNN Tradeoffs

- Advantages
  - Can process any length input
  - (in theory) can use info from many steps back
  - Model size doesn't increase
  - Same weights, symmetry in input processing
- RNN Disadvantages
  - Recurrent computation is slow
  - (in practice) difficult to access info from many steps back

Applications

- Image Captioning
  - CNN + RNN
- Visual Question Answering (VQA)
  - CNN + LSTM
- Multilayer RNNs

Vanilla RNN Gradient Flow

Backpropagation from h_t to h_{t-1} multiplies by W_{hh}^T

- Non-linearity gradient always < 1, so gradient vanishing
- No non-linearity: Largest singular value > 1: exploding gradient; < 1: vanishing gradient

2.3 LSTM

Fix the vanishing gradient problem: separate memory
$[i\ f\ o\ g] = [\sigma\ \sigma\ \sigma\ tanh]\, W\, [h_{t-1}\ x_t]\ \ c_t = f@c_{t-1} + i@g\ \ h_t = o@tanh(c_t)$

Gates:

- i: Input gate
  - whether to write to cell
- f: Forget gate
  - whether to erase cell
- o: Output gate
  - how much to reveal cell
- g: Gate gate
  - how much to write to cell

LSTM Gradient Flow Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no W: Uninterrupted gradient flow

- Notice the gradient contains the f gate's vector of activations
  - allows better control of gradients values, using suitable parameter updates of the forget gate
- Notice that are added through the f,i,g,o gates
  - better balancing of gradient values

Summary

1. RNNs allow a lot of flexibility in architecture design
2. Vanilla RNNs are simple but don't work well
3. Common to use LSTM or GRU: additive interactions improve gradient flow
4. Backward flow of gradient in RNN
   1. Vanilla RNN: explode or vanish
   2. Exploding is controlled with gradient clipping
   3. Vanishing is controlled with additive interactions(LSTM)

3. Video Analysis

Video as data: 4D tensor: $T \times 3 \times H \times W$ Video Classification: Recognize **actions**

- Problem: Videos are big
- Solution: Train on short clips

3.1 Models (for short clips)

1. Single-Frame CNN
   1. Idea: normal 2D CNN to classify video frames independently
   2. Often a very strong baseline
2. Late Fusion with FC
   1. Intuition: Get high-level appearance of each frame and combine them
   2. Run 2D CNN on each frame, concate features and feed to MLP (Clip feature:$TDH'W'$)
3. Late Fusion with Pooling
   1. Run 2D CNN on each frame, pool features and feed to Linear
   2. Frame feature: $TDH'W'$ --Average Pool over space and time-- Clip feature: D
   3. Problem: Hard to compare low level motion between frames
4. Early Fusion
   1. Intuition: Compare frames with very first Conv layer, then normal 2D CNN (First 2D conv collapes all temporal info)

    2. Input: $T \times 3 \times H \times W$, reshape: $3T \times H \times W$, after first conv: $D \times H \times W$, then 2D CNN: class score
    3. Problem: one layer of temporal processing may not be enough
5. 3D CNN
    1. Intuition: Use 3D Conv and Pool to slowly fuse temporal info

Comparison:

- Late Fusion
    - Build slowly in space
    - All-at-once in time at end
- Early Fusion
    - Build slowly in space
    - All-at-once in time at start
    - **No temporal shift invariance!**
- 3D CNN
    - slow fusion
    - **Temporal shift invariance**

Application:

- C3D (The VGG of 3D CNNs)
    - 3D CNN: all 3x3x3 Conv, 2x2x2 Pooling
    - Used as video feature extractor
    - Problem: 3x3x3 Conv is expensive
- Two-Stream Fusion Networks
    - Use both motion and appearance
    - Spatial stream ConvNet
        - input: Single Frame $(3 \times H \times W)$
        - Conv + pool + full + softmax
    - Temporal stream ConvNet
        - input: multi-frame optical flow $([2 \times (T-1)] \times H \times W)$

3.2 Models (long-term structure)

Intuition: handle sequences: recurrent networks

- Extract features with CNN (2D or 3D)
- Process local fearures using recurrent networks (LSTM)
- Sometimes don't backprop to CNN to save memory, pretrain CNN as feature extractor

Recurrent Convolutional Network Entire network uses 2D feature maps: $C \times H \times W$ Each depends on 2 inputs:

- Same layer, previous timestep
- Prev layer, same time step

Use different weight at each layer, share weights across time

Comparison:

- RNN + CNN:
  - RNN: Infinite temporal extent
  - CNN: finite temporal extent
- RCN: Infinite temporal extent

Probelm: RNNs are slow for long sequences (can't be parallelized)

Part 2: Object Detection and Instance Segmentation

1. Object Detection

1.1 Single Object

Task

- localization + classification
- Output: 2D bounding box, 4 DoF

Treat localization as a regression problem Regression Loss: Error $(\Delta x, \Delta y, \Delta w, \Delta h)$

- L1 loss: $\Sigma|\Delta_i|$
  - robust, however not good at convergence
- L2 loss: $\Sigma\Delta_i^2$ **Not the same to L2 norm**
  - Not robust to a large error, good at convergence
- RMSE (Rooted mean squared loss): $\sqrt{\frac{1}{N}\Sigma\Delta_i^2}$
  - the gradient of sqrt is bad at zero
- Smooth L1 loss & Huber loss
  - quadratic when diff is small, linear when diff is big

1.2 Multiple Objects

Different images need different numbers of outputs

- Silding-Window based method
  - Idea: apply CNN to many different crops, classifies each crop as object or background
  - Problem: huge numbers of locations, scales, aspect ratios, expensive
- R-CNN
  - Idea: Region Proposals: Selective Search
  - Implementation
    - RoI from proposal method (~2k)
    - Warped image regions
    - Forward each region through ConvNet (pretrained)
    - Classify regions with SVMs
    - Predict corrections to RoI (dx, dy, dh, dw)
  - Problem
    - ~2k independant forward pass
    - cropped region doesn't contain sufficient info to refine bbox
- Fast R-CNN

  - Idea: Pass the image through ConvNet before cropping, crop Conv feature
  - Structure
    - Backbone network: AlexNet, VGG...
    - RoI proposal method, crop and resize
    - Per-Region Network
    - Object category: Linear + softmax
    - Box offset: Linear
  - Cropping features: RoI Pool
  - Problem: Runtime dominated by region proposals
- Faster R-CNN
  - Idea: Insert Region Proposal Network (RPN) to predict proposals from features
  - RPN
    - use K different anchor boxes of size and scale at each point
    - binary classification: whether contain object
    - For positive boxes, also predict a correction from anchor to gt-box (regress 4 numbers per pixel)
    - Sort boxes by their score, take top ~300
  - Training: jointly train with 4 losses
    1. RPN classify object / not object
    2. RPN regress box coordinates
    3. Final classification score
    4. Final box coordinates
  - Inference: two-stage detector
    - Stage 1: Run once per image
      - Backbone network
      - RPN
    - Stage 2: Run once per region
      - Crop features: RoI pool / align
      - Predict object class
      - Predict bbox offset

NMS: IoU threshold

How to Evaluate Detection?

- precision:
  - measures "false positive rate"
  - true object detection / total number of objects predicted
- recall:
  - measures "false negative rate"
  - true object detections / total of real objects
- Evaluation metric: AP (Average Precision)
  - $AP = \frac{1}{11}\sum_{Recall_i} Precision(Recall_i)$
  - mAP is the mean of AP

Single-Stage Detectors: YOLO/SSD/RetinaNet

2. Instance Segmentation

Different Approaches for Instance Segmentation

- Top-down approach
    - object detection and then further find a binary mask inside bbox
- Bottom-up approach
    - grouping: group together similar data points and represents them with a token
    - classification

2.1 Mask R-CNN

Top-Down Approach: Mask R-CNN

- Idea: add a small mask network that operates on each RoI and predicts a 28x28 binary mask
- Structure
    - CNN + RPN
    - RoI Align
    - Conv

Problem with RoI Pool
Max-pool within each subregion:
Region features always the same size even if input regions are not --> Region features slightly misaligned **RoI Align** Sample at regular points in each subregion using **bilinear interpolation**

Masks

- Class-Specific vs. Class-Agnostic
    - default instantiation: class-specific masks
    - with class-agnostic masks is nearly as effective
- Multinomial vs. Independent

3. 3D Object Detection and Instance Segmentation

3.1 3D Object Detection

3D Object Detection:
3D oriented bounding box $(x, y, z, w, h, l, r, p, y)$ Simplified bbox: no roll & pitch (but has yaw) Much harder than 2D

A 2D bbox on an image is a frustum in the 3D space
Localize an object in 3D: it can be anywhere in the camera viewing frustum

3.2 From RGB-D

3D Object Detection from RGB-D: **Frustum PointNet**

- depth to point cloud
- 2D region (from CNN) to 3D frustum
- 3D box (from PointNet)

Pipeline of Frustum PointNet

- Frustun Proposal
- 3D Instance Segmentation
- Amodal 3D Box Estimation

Very expensive to perform sliding windows in 3D

3.3 From Monocular Camera

- Same idea as Faster RCNN, but proposals are in 3D
- 3D bounding box proposal, regress 3D box parameters + class score

VoteNet

- idea: ask the surface points to vote for object centers
- Deep Hough Voting: Pipeline
  - Input: point cloud (PointNet++)-->
  - Seeds (XYZ + feature) -->
  - Votes (XYZ + feature) -->
  - Vote clusters -->
  - Output: 3D bounding boxes

Part 3: Generative Model

1. Fundation

Objectives:

1. Learning $p_{model}(x)$ that approximates $p_{data}(x)$
2. **Sampling new x from $p_{model}()$**

Discrinative vs. Generative

- Discriminative
  - Y: labels, X: inputs
  - Learn $P(Y|X)$
- Generative
  - X is all the variables
  - $P(X)$ or $P(X, Y)$ (if labels are available)

Generative models

- Explicit density
  - Tractable density
    - **PixelRNN/CNN**
  - Approximate density
    - Variational
      - **VAE**
    - Markov Chain
      - Boltzmann Machine

- Implicit density
    - Direct
        - **GAN**
    - Markov Chain
        - GSN

2. PixelRNN/CNN

2.1 FVBN (Fully Visible Belief Network)

Explicit density model: $p(x) = p(x_1, x_2, \ldots, x_n)$
Likelihood of image x -> Joint likelihood of each pixel
Idea: use chain rule to decompose likelihood of an image
$p(x) = \Pi_{i=1}^{n} p(x_i | x1, \ldots, x_{i-1})$

2.2 PixelRNN and PixelCNN

Pros:

- Can explicitly compute likelihood $p(x)$
- Easy to optimize
- Good samples

Con:

- Sequential generation --> slow

3. VAE

3.1 Autoencoder

- Input data x --Encoder-->
- Features z --Decoder-->
- Reconstructed input data $\hat{x}$ --> L2 loss

can't generate new images from AE: don't know space of z

3.2 Variational Autoencoders

Intuition: x: image, z: latent factors to generate x: attributes, orientation, etc

- How to represent this model?
    - choose $p(z)$ to be simple, $p(x|z)$ be complex
- How to train?
    - Maximize likelihood of training data: $p_\theta(x) = \int p(z) p_\theta(x|z) dz$
    - $p(z)$: Standard normal distribution *N(0,I)*
    - $p_\theta(x|z)$
        - decoder neural network
        - assume this prob. distribution is also Gaussian
        - Only needs to predict $\mu_{x|z}, \Sigma_{x|z}$

    - Problem: Intractable:
        - $\int$, intractable to compute $p(x|z)$ for every z
        - Monto Carlo: Unbaised but high variance
- How to learn VAE:
    - Another way: $p_\theta(z|x) = \frac{p(z)p_\theta(x|z)}{p_\theta(x)}$, $p_\theta(x)$ is intractable
    - Probabilistic encoder: $q_\phi(z|x)$
        - Learn $q_\phi(z|x)$ to approximate $p_\theta(z|x)$
        - q: take input x, output $\mu_{z|x}, \Sigma_{z|x}$
    - How to learn
        - Build a loss (NLL loss): $L = -log(p_{\theta,\phi}(x))$
        - Minimize L with respect to θ, ϕ
        - But $log(p_{\theta,\phi}(x))$ is still intractable, need to approximate
            - **ELBO**: Evidence Lower BOund
                - E decoder + KL
- Why Called Variational: Variational inference
- Generating data
    - use decoder network and sample z from prior
    - Diagonal prior in z: independent latent variables
    - Different dims of z encode interpretable factors of variation

3.3 Summary

Probabilistic spin to traditional AEs: allows generating data Defines an intractable density: derive & optimize a (variational) lower bound

- Pros
    - Principled approach to generative models
    - Interpretable latent space
    - Allows inference of $q(z|x)$, can be useful feature representation for other tasks
- Cons
    - Maximizes lower bound of likelihood: OK but not good as PixelRNN/CNN
    - Samples blurrier and lower quality compared to SOTA
- Active areas of research
    - More flexible approximations
    - Learning disentangled representations

4. GAN

GANs: not modeling any explicit density function

- Problem: want to sample from complex, high-dim training distribution, no direct way
- Solution: Sample from a simple distribution like random noise, learn transformation to training distribution
- Objective: Generated images should look "real"
- Solution
    - Use a discriminator network to tell whether the generate images is within data distribution or not
- Structure

- Discriminator network: try to distinguish between real and fake images
- Generator network: try to fool discriminator by generating real-looking images
- Training GAN: 2-player games
  - Train jointly in minimax game
  - $min_{\theta_g} max_{\theta_d}$
  - Discriminator (θ_d) wants to maximize objective such that $D(x)\ 1$, $D(G(z))$~0
  - Training: alternate between:
    - Gradient ascent on discriminator
    - Gradient ascent on generator, different objective
- Architecture for stable Deep Conv GANs
  - replace pooling with strided Convs(discriminator) and fractional-strided Convs(generator)
  - use BN in both d & g
  - remove FC hidden layers
  - use ReLU in generator except for output, which uses Tanh
  - use LeakyReLU in discriminator
- Evaluation Metric
  - Manual inspection
  - Quantitative measures
    - Nearest neighbor
    - User study
    - Mode drop and mode collapse
    - FID
      - Embed a set of generated samples into a feature space given by a specific layer of InceptionNet(or any CNN)
      - Lower FID means smaller distance between synthetic and real data distributions
- Interpretavle Vector Math

Summary

Don't work with explicit density function Take game-theoretic approach

- Pros
  - Beautiful, SOTA
- Cons
  - Trickier / more unstable to train
  - Can't solve inference queries such as $p(x), p(z|x)$
- Active areas of research
  - Better loss, more stable training
  - Conditional GANs, GANs for all kinds of application

VAE

- Blurry
- Full coverage of the data
- Support approximate inference

GAN

- More reakistic

- Only penalize fakem, may suffer mode collapse
- Can't infer prob.

Part 4: 3D Reconstruction

1. SfM

1.1 2D-3D basics

Camera center - Pixel position - 3D point Camera Pose from 2 Views

1.2 Structure from motion

SfM Pipeline

- Unstructured Images --Assoc.-->
- Scene Graph --SfM-->
- Sparse Model --MVS-->
- Dense Model

Data Association

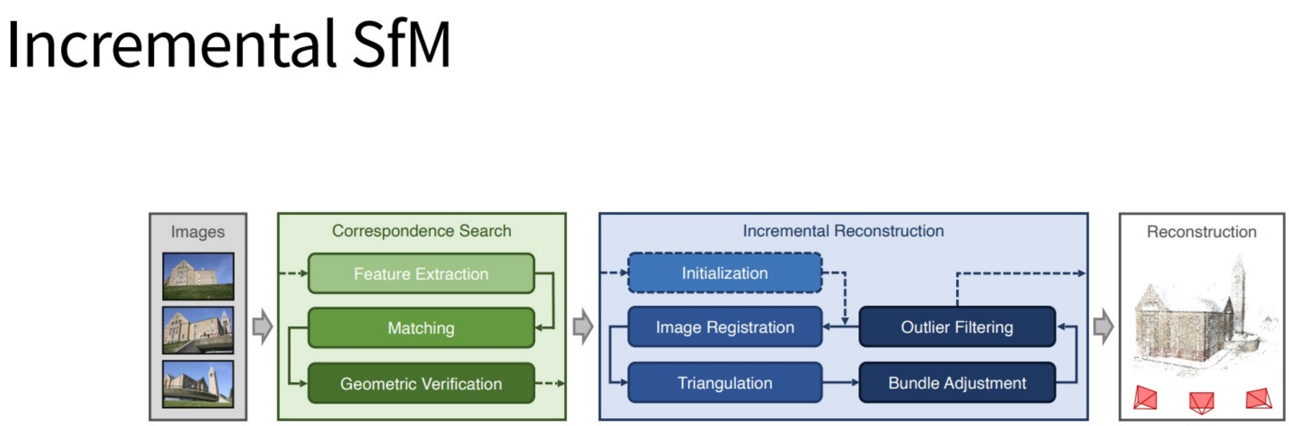
- identified pairs of overlapping images
- geometrically verified inlier matches (and feature descriptors optionally)
- related camera poses (if known calibration)

Structure from Motion: 3 paradigms

1. Incremental
2. Global
3. Hierarchical

Incremental SfM

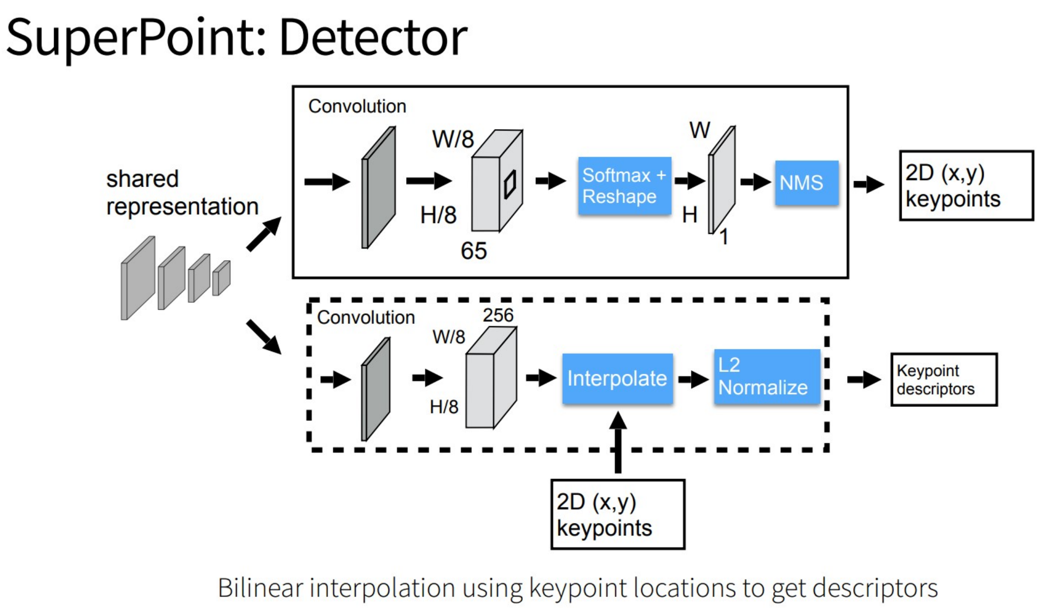
- Initialization
    - Choose 2 non-panoramic views
    - Triangulate inlier correspondences
    - Bundle adjustment
        - Non-linear refinement of structure and motion
        - Minimize reprojection error
    - Absolute camera registration
        - Find 2D-3D correspondences
        - Solve Perspective-n-Point problem
    - Outlier filtering



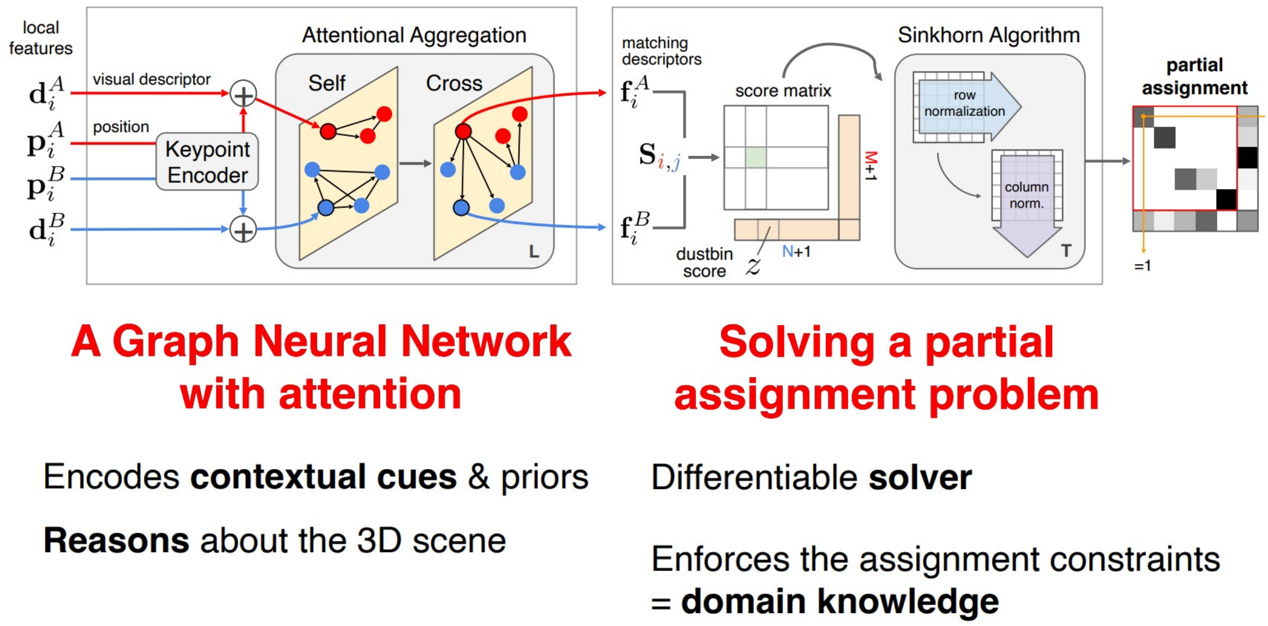
1.3 Learning-based structure from motion

Improve the robustness/precision via data-driven learning

1. Improving features and keypoints for matching
    1. SuperPoint: A Learned Detector and Descriptor Key Challenges:
        1. How to get training data?
        2. How to maintain precision/detection at high resolution



2. Improving the matching process via global reasoning SuperGlue: context aggragation + matching + filtering



Slide credits: Paul-Edouard Sarlin

2. MVS

2.1 Introduction to Multi-View Stereo

Idea: Reconstruct the dense 3D shape from a set of images and camera parameters Why MVS Given the Development of Depth Sensors?

- Measurement of depth sensors is either sparse or with limited range
- Texture and lighting info is missing
- Nice association between image and 3D

2.2 Classic MVS

Reconstruction from Silhouettes

- Approach
  - Back-project each silhouette
  - Intersect back-projected volumes
- Pros
  - Easy to implement, fast
  - Accelerated with Octress
- Cons
  - Requires identification of silhouettes
  - Not photo-consistent
  - No concavities

Multi-View Stereo vs. Two-View Stereo

- Different points on the object's surface will be more clearly visible in some subset of cameras
  - Could have high-rres closeups of some regions
  - Some surfaces are foreshortened from certain views
  - Some points may be occluded entirely in certain views

- More measurements per point can reduce error

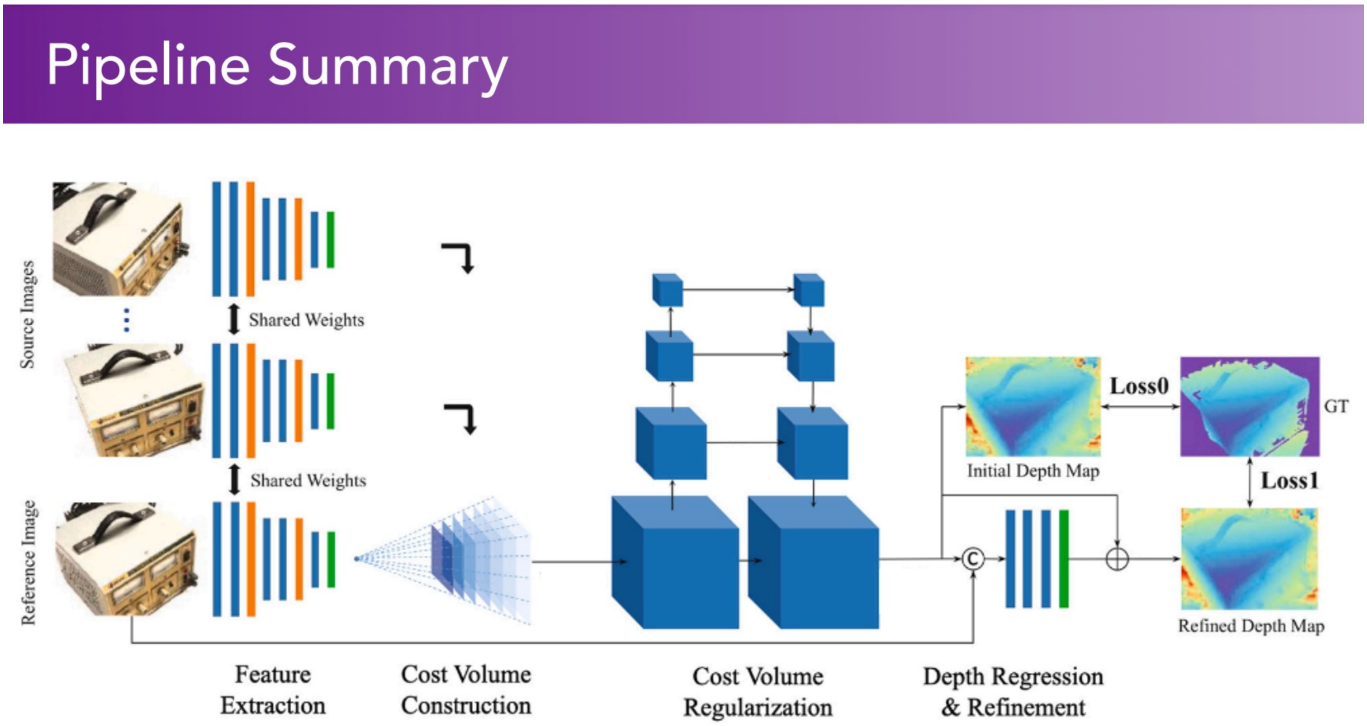
Limitations of classical MVS

- Textureless Area
- Reflection/Transparency
- Repetitive patterns

Learning-Based MVS

- Learned feature: more robust matching
- Shape prior: more complete reconstruction

2.3 Learning-based MVS: a first pipeline



- Issues
  - Quality and speed tradeoff
  - Flying points when there is abrupt depth change
- Possible solution
  - Depth estimation following a coarse to fine strategy
  - Stronger loss function regularizing flying points

2.4 Learning-based MVS: Improvements

- Adaptive Space Sampling
  - Coarse-to-fine Sampling
    - Analyze per-pixel confidence intervals
    - Narrow down the sampling range based on uncertainty
- Depth-Normal Consistency
  - How to Improve Surface Smoothness?
    - Key observation: Surface smoothness is reflected by surface normal

- Summary
  - Deep volumetric stereo can lead to more robust matching and more complete reconstruction
  - But volume-based methods are NOT computationally efficient, since the 3D target scene is sparse
  - Adaptive sampling can improve computation efficiency and reconstrction quality
  - Normal prediction is easier than depth, and can help improve depth accuracy and smoothness

3. NeRF

Neural Radiance Field Key Idea of NeRF

- Use an implicit function to replace the volume representation
- Track light emission along different directions

3.1 Implicit Functions

SDF: Signed Distance Field

- Pros
  - Compared to point clouds: clearly defines the (iso-)surface
  - Compared to meshes: can continuously adapt to arbitrary topology
  - continues in 3D
  - Can give analytic normals, can be applied with boolean operations
- Cons:
  - well-defined for only watertight meshes (with interior and exterior)
  - need extra steps to visualize
  - Not all complex shapes can be efficiently/accurately represented with simple primitives

DeepSDF: Efficiently representing complex shapes by learning their SDF

- Idea: Learn a continuous representation of 3D implicit surfaces

Visualization of implicit functions is done by extracting iso-surfaces

1. Running inference for multiple queries in input space
2. Rendering the result by combining the queries

3.2 Volume Rendering with Ray Marching

General Idea

- The appearance of the surface will be observed at views along the camera ray
- If we have a **light transport model** from the surface along the ray to the pixels, we will know the pixel color

A single point: $I_1 = \alpha_1(\frac{c_1}{\sigma_1})$

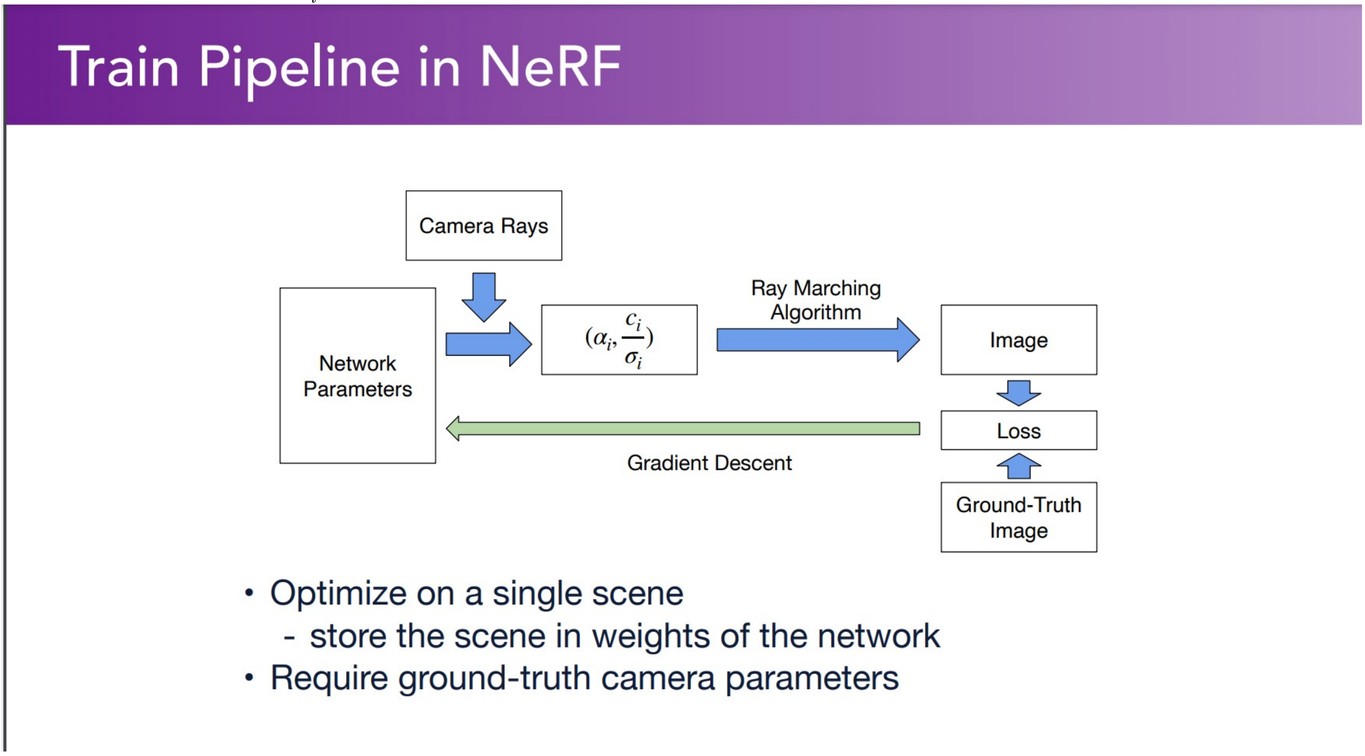
- I_1: light intensity after point 1
- c_1: predicted emission radiance at point 1
- α_1: opacity of point 1

Multipoints: $I_k = \alpha_k(\frac{c_k}{\sigma_k}) + (1-\alpha_k)I_{k-1}$

$T_i = \Pi_{j=i+1}^{n}(1-\alpha_j) = exp(-\sum_{j=i+1}^{n}\sigma_j\delta_j)$ $I = \sum_i T_i\alpha_i(\frac{c_i}{\sigma_i})$, final radiance of the ray

3.3 Learning NeRF

What NeRF learns: $(\alpha_i, \frac{c_i}{\sigma_i}) = F_\Theta(x, y, z, \theta, \phi)$ Pixel loss: Comparing I with gt pixel value



Part 5: Embodied AI

1. Embodied AI

Classic Disembodied AI: "intelligence as computation" Embodiment: "intelligence requires a body"

Visual Motor Coordination/Integration

How Humans Learn: Perception-Action Loop

- Perceive, forms hypotheses, and then take action to examine

Research Questions

- Classic AI
  - Thinking, reasoning, abstract problem solving
- Embodied AI
  - Movement, physical interaction with the real world

Trends in CV Field

- Moving to Interaction?Robotic tasks
- Encourage active/interactive perception

Now: Industrial Robots, Autonomous Driving; Future: Home Robots Related Work: RT-1 Limitations

- Limited task diversity, still mainly pick and place
- No 3D Vision
  - Lack of geometry
  - Lack of 3D scene modeling
- Demonstrations are costly, scalability is questionable

Goal: a scalable 3D-aware home robot

- Keys
  1. better scalability via leveraging synthetic data and simulation
  2. leverage 3D signal to improve performance and generalizability
- Mobile manipulation
  - Grasping
  - Functional manipulation
  - Navigation

2. Object Grasping

2.1 Tasks

Grasping: restraining an object's motion in a desired way by applying forces anf torques at a set of contacts Grasping Synthesis: a high-dim search or optimization problem to **find gripper poses** or joint configs

Terms

- Grasp Pose: the position and orientation of a hand
  - 4-DoF grasp: 3D position + 1D orientation (top-down)
  - 6-DoF grasp: 3D position + 3D orientation
- Closure
  - Force Closure
    - the positive span of the wrench cones is the entire wrench space
    - good minimum requirement for a robot hand
  - Form closure
    - the rigid body is fully immobilized by a set of rigid stationary fixtures
    - usually too strict, requiring too many contacts
  - Simply: form closure -> force closure -> successful grasp

Dataset Real Dataset: GraspNet-1Billion: with grasping annotation Grasp pose annotation pipeline:

- The grasp point is firstly sampled from Point Cloud
- Then the grasp view, the in-plane retation and the gripper depth are sampled and evaluated, 6D pose of each object
- Collusion detection os alsoo conducted to avoid the collision between grasps and background or other object

Steps:

1. Grasp poses are sampled and annotated for each single object

2. For each scene, we project these grasps to the corresponding object

2.2 Representation

2.2.1 Voxel Grids

Volumetric Grasping: similar to a pixel over a 3D grid instead of a 2D image Network: **VGN** (Volumetric Grasping Network) Explicit geometry; limited by the volume resolution

TSDF: Truncated Signed Distance Function

VGN Detection Network

- Input: TSDF volume fused from multi-view depth maps
- Output: 6-channel volume grid, with each voxel containing quality(1d), orientation(quaternion,4d), width(1d)
- Follows a 3D FCN encoder-decoder architecture
- Post processing: 3D Gaussian smoother, NMS

2.2.2 Point Cloud

Backbone: PointNet, PointNet++ Network: GraspNet-baseline Explicit geometry; sensor noise, depth error

Problem: transparent and specular objects

Goal: Generalizable Material-Agnostic Object Grasping

Approaches:

1. First restoring depth, then grasping
2. Multiview RGB-based method (depth-free)

2.3.2 GraspNeRF

- Sparse inputs
- Generalize to novel scene
- Real-time speed
- 6-DoF grasping
- End-to-end differentiable

3. Object Manipulation

Introduction: Manipulation Tasks: Grasping, Ungrasping, Object rearrangement, In-hand manipulation, Tool use

GAPartNet: Generalizable, actionable DexGraspNet: dexterous grasp dataset

4. Locomotion and Navigation

Navigation: drive the agent to find the target Challenges:

1. Novel scene

2. Noisy indoor positioning
3. Poor scene understanding

Goals: Point Goal, Image Goal, Object Goal

Navagation Methods

- Classical Modeular Navigation
    - Good generalizability
    - Satisfying performance
    - Hard to implement
- End-to-end RL
    - Easy to implement
    - Satisfying performance
    - Requires extensive training time
    - Poor generalizability

4.1 Classical Modular Navigation

1. Mapping
    1. 3D representation
        1. Explicit: voxel, mesh, pc, octomap
        2. Implicit: TSDF, voxel hashing
    2. Mapping Methods
        1. Panoptic fusion
        2. Kimera: Real-Time Metric-Semantic Localization and Mapping
        3. 3D Dynamic Scene Graphs
2. Planning
    1. Rule-based local planner
        1. Search-based Methods
            1. DFS & BFS
            2. A* & variations
        2. Sample-based Methods
            1. RRT & varaiations
            2. Fast Marching
    2. Learning-based non-local planner
        1. Learning-based Methods
            1. Imitation Learning
3. Execution

4.2 End-to-end RL

Hardware Framework: Robot End - Computer End - Cloud server