Image Models, Interpreting Images

Saturday, January 13, 2024 5:38 PM

Given a point on a surface: the intensity is the total of the incoming lights, material properties, and object shape

$$L_o(\omega_o) = \int L_i(\omega_i) f(\omega_o, \omega_i)(\omega_i \cdot n) d\omega_i$$

Where:
wo = outgoing light direction
wi = incoming light direction
f = BRDF reflection model representing fraction of reflected energy, diffuse + specular

Other models:

Di-electrics
Subsurface scattering
Fluorescence

Interpreting Images:
- Can't interpret images pixel by pixel
- Interpret images through local differences in intensity

Photometric Stereo

Thursday, January 18, 2024 12:48 PM

Photometric Stereo:

Idea: Control the light sources to recover the surface shape

Note that a surface given by $z(x,y), n = t_1 \times t_2 = \left(1,0,\frac{\partial z}{\partial x}\right) \times \left(0,1,\frac{\partial z}{\partial y}\right) = \left(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, 1\right)$

Lambertian model: $I^k(x,y) = \rho(x,y) \cdot n \cdot s^k$ for ρ the surface property, and s the direction to the light

Therefore if $b(x,y) = \rho(x,y) \cdot n$, then $I^k = b(x,y) \cdot s^k$

Given multiple s^k we want to solve for $b(x,y)$

Define $e(x,y) = \begin{pmatrix} I^1(x,y) \\ \vdots \\ I^K(x,y) \end{pmatrix}_{K\times 1}$, $S = \begin{pmatrix} s^1 \\ \vdots \\ s^K \end{pmatrix}_{K\times 3}$

For $K = 3$: $b(x,y) = S^{-1}e(x,y)$

In general: $b(x,y) = (S^T S)^{-1} S^T e(x,y)$

And $n = \frac{b(x,y)}{|b(x,y)|}$, $\rho = |b|$

Recovering Surface from Normals:

Integrate $\frac{\partial z}{\partial x}$ along row 0 to get $z(x,0)$

Integrate $\frac{\partial z}{\partial y}$ along each column starting with value from first row

$z(x,y) = z(x_0,y_0) + \int_{(x_0,y_0)}^{(x,y)} (p\,dx + q\,dy)$ for deritave esitmates p, q

May have errors because of noisy estimate

Instead, could minimize cost function: $\iint (z_x - p)^2 + (z_y - q)^2 dxdy$ for z_x, z_y derivatives of the best fit surface

Cameras and Projection, Projective Geometry

Thursday, January 25, 2024 7:23 PM

Camera Coordinate System:

- Origin at optical center
- Image plane in front of origin
- Camera looks down -z axis

Projection matrix: convert 3D rays to points on image plane

$\mathrm{P}_E = [\mathrm{R} \mid -\mathrm{RC}]$

$\mathrm{R} = \begin{pmatrix} \mathrm{u} \\ \mathrm{v} \\ \mathrm{w} \end{pmatrix}$ for transformation from world coordinates to camera coordinates with orthonormal basis u,v,w

RC the displacement of the camera from world origin

Intrinsic matrix: intrinsic properties of the camera

$$K = \begin{pmatrix} -d_x & s & c_x \\ 0 & -d_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Aspect ratio: $\alpha = \frac{d_x}{d_y}$

Skew: s

Principle point: $\left(c_x, c_y\right)$

$P = K * P_E$

Projective Geometry: all coordinates in homogenous

- $ax + by + c = 0 \rightarrow (a, b, c)$
- Point on a line: $l \cdot x = 0$
- Point of two intersecting lines: $x = l_1 \times l_2$
- Line going through two points: $l = x_1 \times x_2$

- Ideal points $x_\infty = (a, b, 0)$
  Line at infinity: $l_\infty = (0, 0, 1)$
- Intersection of two parallel lines is an ideal point

Features Detection

Saturday, January 27, 2024 2:21 PM

Feature detection: find parts of the image that are unusual (unique)

Edge detection: find edges in the images

- Derivative: calculate gradient of image

  $$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

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

  Edges where magnitude of gradient is large

- Convolution: convolve image with feature filter

  Horizontal edge: $K = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$, vertical edge: $K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

  Alternatively, smooth image using gausian filter, then compute numerical derivative:

Corner Detection:

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

Or calculate eigenvalues

- Smooth image to reduce noise
- Compute x and y derivatives
- Construct C in windows around each pixel
- Find eigenvalues and if both are large, then it is a corner

Matching

Monday, February 5, 2024 6:42 PM

Matching features: want to match features from image A to B assuming they are related

- Useful for 3D reconstruction by reconstruction 3D point from two images

Match square windows around each interest point: simple similarity based

- For each interest point in A, find the best matching point in B using a distance metric

- Sum Squared Distances: $\sum_{x,y}\left(A(x,y)-B(x,y)\right)^2$
    - May give good scores to bad (ambiguous) matches
    - Calculate SSD(f1, f2) / SSD(f1, f2') to determine the best match

- Normalized Cross Correlation: $\sum_{x,y}\frac{(A(x,y)-\mu_A)(B(x,y)-\mu_B)}{\sigma_A\sigma_B}$

- Use a threshold value to filter out extremely bad matches
    - Maximize TP: increase threshold
    - Minimize FP: lower threshold

SOFT:

- Can handle changes in viewpoint, illumination
- Fast and efficient, run in real time
- Lots of code available

Learning correspondence:

- Siamese network to detect patch similarity
    - Train two convnets on patches and train to detect patching/reject mismatching patches

3D Reconstruction from Matching

Monday, February 5, 2024 7:05 PM

Given two images taken from two cameras with a matching point

- Measurements X_L, X_R
- Unknowns: X_0, Z_0
- Baseline: d distance between images
- Focal length: f

$X_0 = \frac{dX_L}{X_L - X_R}$, $Z_0 = \frac{df}{X_L - X_R}$

Note: $t \times p = [t]_\times \cdot p = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \cdot p$

Epipolar geometry:

Given two cameras: $[I\ 0]$, $[R\ t]$

Essential matrix: $E = [t]_\times R$

Fundamental Matrix: perform SVD on a list of points to fulfil the linear equation $x^T F x = 0$

8-point algorithm

Given n point correspondences, set up a system of equations:

$$\begin{bmatrix} u_1u_1' & v_1u_1' & u_1' & u_1v_1' & v_1v_1' & v_1' & u_1 & v_1 & 1 \\ u_2u_2' & v_2u_2' & u_2' & u_2v_2' & v_2v_2' & v_2' & u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_n' & v_nu_n' & u_n' & u_nv_n' & v_nv_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

Assume $F = U\Sigma V^T$, then $\Sigma' = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and the solution $F' = U\Sigma' V^T$

Motion from correspondences

- Use 8-point algorithm to estimate F
- Get E from F:

$$\mathbf{F} = \mathbf{K}_2^{-\top}\mathbf{E}\mathbf{K}_1^{-1}$$
$$\mathbf{E} = \mathbf{K}_2^{\top}\mathbf{F}\mathbf{K}_1$$

- Decompose E into skew-symmetric and rotation matrices:

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$$

For K_1 and K_2 the intrinsic camera matrices for each camera

Two-View Reconstruction

Monday, February 12, 2024 9:22 PM

Idea:

- Detect features in each view
- Match features across two views
- Use fundamental matrix to eliminate outliers
- Estimate camera rotation and translation across views
- Back project rays from camera centers to triangulate 3D point

Eliminating Outliers: RANSAC

- Randomly sample s points
- Fit a model to sample s
- Count the number of inliers that approximately fit the model
- Repeat N times
- Choose the model with largest set of inliers

For fundamental matrix:

- For N times
  - Pick 8 correspondences
  - Estimate F
  - Count number of inliers with $x_1^T F x_2$ close to 0
- Pick the F with the largest number of inliers

Adapt number of iterations based on proportion of outliers

$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}$ for p estimated probability of the correct solution, ϵ proportion of outliers

Recognition, Retrieval, Precision & Recall

Wednesday, February 21, 2024 4:43 PM

Statistical viewpoint: want to find the probability of a class given the image

$$\frac{P(class\,|image)}{P(not\ class|image)} = \frac{P(image\,|class)}{P(image\,|not\ class} * \frac{P(class)}{P(not\ class)}$$

Challenges:

- Representation: how to represent an object category
- Learning: how to form the classifier
- Inference: How can the classifier be used on new data

- Scale, viewpoint, lighting, occlusion issues

Bag of Words: frequency of words from a defined dictionary

- Extract features of image (SIFT, regular grid)
- For images, sample patches of image around features and add to dictionary
- Learn representations of objects using visual vocabulary (clustering)
- Represent images using frequencies of "visual words"
- Use distances between representations for classification (nearest neighbors, similarity)
  - Cosine similarity: $cossim(d, q) = \frac{d \cdot q}{|d||q|}$

Retrieval: Want to find similar images given a query

Precision = Number relevant / Number returned
Recall = Number relevant / Number total relevant in dataset

Gradient Descent (again):

Given some cost function C:

$w = w - \eta \nabla C$

Neural Networks

Thursday, February 29, 2024 4:45 PM

End to end training: train model to learn feature representations

Hand crafted: use feature representations made by hand

Used neural networks to train feature representations

Optimization: gradient descent

- Update rule: $w_k = w_k - \eta \frac{\partial C}{\partial w_k}$
- Use chain rule to back propagate derivatives: given $y = g(x)$, $z = f(x)$ $\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$
- Momentum: $v_{t+1} = \rho v_t + \nabla f(w_t)$, $w_{t+1} = w_t + \alpha v_t$

Convolutional NNs: use learnable filters to process image

- Use a smaller kernel and convolve input image
- Kernel size: $k \times k \times in\_channels$, out_channels number of kernels
- Apply non linearity (RELU, etc)
- Pooling: Take window and compute max, average, etc
  $\mathrm{a}_{\mathrm{rc}} = \mathrm{x}_{\mathrm{region}} \cdot \theta$

- Transfer learning: can train CNN as feature extractor, then learn a new FCN for each task

Regularization:

- Apply complexity penalty to cost function: $C = C_0 + \lambda * dist(w)$ for regularization parameter
- L1: $dist(w) = \frac{1}{n} \sum |w|$, $w = w - \eta \nabla C - \frac{\eta \lambda}{n} sign(w)$
- L2: $dist(w) = \frac{1}{2n} \sum w^2$, $w = w - \eta \nabla C - \frac{\eta \lambda}{n}$