1 The Digital Image

Problems: Transmission interference, compression artifacts, spilling, scratches, sensor noise, bad contrast and resolution, motion blur

Pixel: Discrete samples of an continuous image function.

Charge Coupled Device (CCD)

Has an array of photosites (a bucket of electrical charge) that charge proportional to the incident light intensity during exposure. ADC happens line by line.

Bloooming: oversaturation of finite capacity photosites causes the vertical channels to "flood" (bright vertical line)

Bleeding/Smearing: While shifting down, the pixels above get some photons on bright spot with electronic shutters.

Dark Current: CCDs produce thermally generated charge they give non-zero output even in darkness (fluctuates randomly) due to spontaneous generation of electrons due to heat → cooling. can be avoided by cooling, worse with age.

CMOS:

Same sensor elements as CCD, but each sensor has its own amplifier → faster readout, less power consumption, cheaper, more noise.

more noise, lower sensitivity

vs CCD cheaper, lower power, less sensitive, per pixel amplification random pixel access, no blooming, on chip integration

Sampling methods

Cartesian (grid), hexagonal, non-uniform

Quantization: Real valued function will get digital values (integers). A lossy process (original can not be reconstructed). Simple version: equally spaced $2^b = \#bits$ levels

Liner Interpolation:
$p(t) = p_0 + (t - t_0)\frac{p_1 - p_0}{t_1 - t_0}$ with $t \in [t_0, t_1]$

Bilinear Interpolation:

Resolution: Image resolution (cropping), geometric resolution (#pixels per area), radiometric resolution (#bits per pixel, color)

Image noise: commonly modeled by additive Gaussian noise: $I(x,y) = f(x,y) + c$, poisson noise (shot noise for low light, depends on signal & aperture time), multiplicative noise: $I = f + f \cdot c$, quantization errors, salt-and-pepper noise. SNR or peak SNR is used as an index of image quality $c \sim N(0, \sigma^2)$, $p(c) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(c-\mu)^2}{2\sigma^2}\right)$, SNR: $S = \frac{F}{\sigma}$ where $F = \frac{1}{XY}\sum_{x=1}^{X}\sum_{y=1}^{Y} f(x,y)$.

Color cameras

Prism need 3 sensors and good alignment
Filter mosaic coat □ directly on sensor
Wheel multiple filters in front of same sensor
New CMOS sensor layers that absorb color at different depths → better quality

2 Image Segmentation

Complete segmentation

Finite set of non-overlapping regions that cover the whole image $I = \bigcup_{i=1}^{n} R_i$ and $R_i \cap R_j = \emptyset$ $\forall i, j, i \neq j$

Thresholding: simple segmentation by comparing greylevel with a threshold to decide if in or out.

Chromakeying: when planning to segment, use special backgroundcolor. (Problems variations due to lighting, noise, halo around foreground due to aliasing mixed pixels due to motion blur(hard α-mask does not work)) $I_\alpha = |I - g| > T$

Receiver Operating Characteristic (ROC) analysis:

ROC curve characterizes performance of binary classifier Classification errors: False negative (FN), false positives (FP)

ROC curve plots TP fraction $\frac{TP}{TP+FN}$ vs FP fraction $\frac{FP}{FP+TN}$

Operating points: choose point with gradient

Pixel connectivity

also regions if x-connected

Connected component raster scanning: scanning row by row, if foreground & label if connected to other label, else give new label. (second pass to find equivalent labels)

Improve: when region found, follow border, then carry on (contour-based method)

Region growing

Start with seed point or region, add neighboring pixels that satisfy a criteria defining a region until we include no more pixels.

Seed region: by hand or automatically by conservative Thresholding

Inclusion criteria: greylevel thresholding, greylevel distribution model (include if $(I(x,y) - \mu^2)^2 < (n\sigma)^2$ and update μ and σ after each iteration) color or texture information

Snakes: active contour, a polygon and each point moves away from seed while criteria is met (can have smoothness constraint) Iteratively minimize enery function $E = E_{tension} + E_{stiffness} + E_{image}$

Background subtraction

simple: $I_\alpha = |I - I_{bg}| < T$ better: $I_\alpha = \sqrt{(I - I_{bg})^T \Sigma^{-1}(I - I_{bg})}$ where Σ is the background pixel appearance covariance matrix, computed seperately for each pixel. (Mahalanobis Distance)

Morphological operators

Logical transformations based on comparison of neighboring pixels. Inputs: Binary image, structuring element S.

Erode: $E = \{x : x + s \in I, \text{for every } s \in S\}$
delete FG pixels with 8-connected BG pixels

Dilate: $E = \{x : x - s, y \in I \text{ and } s \in S\}$
every BG pixels with 8-connected FG pixel make a FG pixel

Opening: $(I \ominus S) \oplus S$ **Closing:** $(I \oplus S) \ominus S$

Uses: smooth regions, remove noise and artifacts.

3 Image Filtering

Modify the pixels of an image based on some function of the local neighborhood of the pixels- If sum greater 1 get brighter, if smaller darker.

shift invariant: Doing the same thing, applying the same function over all pixels (in the formula below if K does not depend on x, y)

linear: linear combination of neighbors can be written as:
$\sum_{(i,j) \in \underbrace{\mathbb{N}(x,y)}_{\text{neighborhood}}} K(x,y,i,j) \underbrace{I}_{\text{Input}}(x+i, y+j)$

Filter at edges: clip filter (black), wrap around, copy edge, reflect across edge, vary filter near edge

Correlation

$I'(x,y) = \sum_{(i,j) \in \mathbb{N}(x,y)} K(i,j)I(x+i, y+j)$

$I' = K \circ I$ e.g. template matching: search for best match by minimizing mean squared error or maximizing area correlation. (remove mean (from filter, from image) to avoid bias)

Convolution

$I' = K * I, I'(x,y) = \sum_{(i,j) \in \mathbb{N}(i,j)} K(i,j)I(x-i, y-j)$
if $K(i,j) = K(-i, -j) \implies$
$correlation = convolution$
$convoution = correlation +$ filter rotated 180°

Continuous: $(f * g)(t)$
$= \int_{-\infty}^{\infty} f(\tilde{t})g(t - \tilde{t})d\tilde{t}$
$= \int_{-\infty}^{\infty} f(t - \tilde{t})g(\tilde{t})dt$

Kernels

separable: if a kernel can be written as a product of two simpler filters → computationally faster (filter $P \times Q$, image $N \times M : (P + Q) * NM$ instead of $PQNM$)

Separable filters can be written as $K(m,n) = f(m)g(n)$. For a rectangular neighborhood with size $(2M + 1) \times (2N + 1), I'(m,n) =$

$f * (\boxed{g * I(N(m, n))})$

$\boxed{I'(m,n)} = \sum_{j=-N}^{N} g(j)I(m, n-j)$

$\boxed{I'(m, n)} = \sum_{j=-N}^{N} f(i)I''(m-i, n)$

Box filter: all same values normalized to sum = 1

Gaussian Kernel: $K(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ is separable, e.q. $\sigma = 1$

Gaussian Smoothing Kernel Top-5
- Rotationally symmetric
- has single lobe (Neighbor's influence decreases monotonically)
- Still one lobe in frequency domain ,(No corruption from high frequencies)
- Simple relationship to σ
- Easy to implement efficiently

High Pass Filter: high pass filter detects edges
High Pass Filter Laplacian Operator

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Low Pass Filter: blurs (detects "smooth" regions) Gaussian Filter is a low pass filter

Conversion: Subtracting one from central element of low-pass filter gives a high-pass filter with inverted sign, because. $(f - \delta) * a = f * a - \delta * a = f * a - a = -(a - (f * a))$ Normalize the low-pass kernel and then subtract one from central element. Normalize low-pass filter, then subtract the kernel from central element matrix. To get the high pass filter, you do not need to normalize.

Band pass filter: ⬜ do LPF and HPF with cutoffs $f_{LP} < f_{HP}$ $f =$ cut of frequencies, cannot coincide
Filter image with high-pass and low-pass filter to get band pass filter. Only works when you have an overlap in frequencies. If no overlap: $I *_{convolution} (\delta - f_{LP-f_{HP}}) \rightarrow$ gap between is band filter.

Band reject filter: ◼ do LPF and HPF with cutoffs $f_{LP} > f_{HP}$

Image sharpening: increases high frequency components to enhance edges: $I' = I + \alpha|K * I|$ K: high-pass filter, α: scalar $\in [0, 1]$ds

4 Features

Desirable properties: shift, rotation, scale, brightness invariant

Edge Detection

How to tell if there is an edge? Local maxima of the first derivative and the zero crossing of the second derivative.

Edge detection filters:
Sobel:
$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Prewitt:
$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Roberts:
$K_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, K_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$
Gradient Magnitude:
$M(x,y) = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$
Gradient Angle:
$\alpha(x,y) = \tan^{-1}(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x})$

Laplacian operator

detect discontinuities by considering second derivative $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ or $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ are discrete

space approximations. Is isotropic(rotationally invariant), zero crossings make edge locations. Sensitive to fine details and noise ($\rightarrow$ smoothing before applying). blur image first (LoG)

Laplacian of Gaussian (LoG): convolve gaussian blurring and laplacian operator in LoG operator
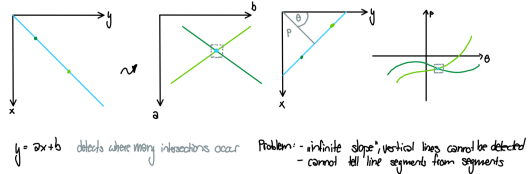
(cheaper) $LoG(x,y) = -\frac{1}{\pi\sigma^4}(1 - \frac{x^2+y^2}{2\sigma^2})e^{-\frac{x^2+y^2}{2\sigma^2}}$

Canny Edge Detector: 5 Steps

1. smooth image with a Gaussian filter
2. compute gradient magnitude and angle using Sobel/Prewitt/...
3. apply non-maximum suppression to gradient magnitude image (Quantize edge normal to one of four directions: horizontal, +45°, vertical, -45°. If $M(x,y)$) smaller than either of its neighbors in edge normal direction suppress, else keep
4. Double thresholding for intensity to detect strong and weak edge pixels
5. Reject weak edge pixels not connected to strong edge pixels

Hough Transform

Fitting a straight line to a set of edge pixels

y= ax+b   detects where many intersections occur   Problem: - infinite 'slope', vertical lines cannot be detected - cannot tell line segments from segments

Alternative parameterization:
$x\cos(\theta) + y\sin(\theta) = p$
$(x-a)^2 + (y-a)^2 = r^2$ **For circles:** if r known: calculate circles with radius r around edge pixels $\rightarrow$ intersection (local maxima) of circles gives center.
Where lots of them meet is the center of a circle. else: use 3D hough transform with parameters (x_0, y_0, r)

Corner Detection
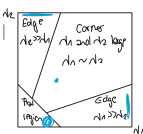
Edges are only well localized in one direction $\rightarrow$ detect corners.
Desirable properties: Acute localization, invariance against shift, rotation, scale, brightness

change, robust against noise, high repeatability
Linear approximation for small $\Delta x\Delta y$: (Taylor) $f(x + \Delta x, y + \Delta y) \approx f(x,y) + f_x(x,y)\Delta x + f_y(x,y)\Delta y$

Local displacement sensitivity (Harris corners)

$S(\Delta x, \Delta y) = (\Delta x\Delta y)\left(\sum_{x,y\in\text{window}}\begin{bmatrix} f_x^2 & f_xf_y \\ f_xf_y & f_y^2 \end{bmatrix}\right)\begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$
$\approx$ SSD. Find points where $\min\Delta^T M\Delta$ is large for $||\Delta|| = 1$ i. e. maximize the eigenvalues of M
Harris cornerness: Measure of cornerness
$C(c,y) = \det(M) - k * trace(M)^2 = \lambda_1\lambda_2 + k(\lambda_1 + \lambda_2)$
Robustness of Harris corner detector: Invariant to brightness offset, invariant to shift and rotation but not to scaling! $\lambda_1 >> \lambda_2 \rightarrow$ edge, λ_1 and λ_2 large $\rightarrow$ corner, else $\rightarrow$ flat region.

not scale invariant:

Overcome issues: look for strong DoG response or consider local maxima in position and scale space, Gaussian weighing.

Lowe's SIFT features

Look for strong responses of difference of Gaussians (DoG) filter, only look at local maxima in both position and scale.

DoG: $DoG(x,y) = \frac{1}{k} * e^{-\frac{x^2+y^2}{(k\sigma)^2}} - e^{-\frac{x^2+y^2}{\sigma^2}}$ e.g. $k = \sqrt{2}$
Orientation: create histogram of local gradient directions computed at selected scale, assign canonical orientation at peak of smoothed histogram. Get a SIFT descriptor (threshold image gradients are sampled over 16×16 array of locations in scale space) and do matching with these. Invariant to scale, rotation, illumination and viewpoint.

5 Fourier Transformation

Aliasing: Happens when undersampling e.g. taking every second pixel, else characteristic errors appear: typically small phenomena look bigger, fast phenomena look slower. (e.g. wagon wheels backwards in movies, checkerboards misrepresented)

Fourier Transform

$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$
$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}$
Represent function on a new basis with basis elements $e^{i2\pi(ux+vy)} = \cos(2\pi(ux + vy)) + i\sin(2\pi(ux + vy))$
$F(u) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi fx}dx$, **2D:** $F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi i(ux,vy)}dxdy$
For images: transformed image $\rightarrow F = U * f \leftarrow$ vectorized image, U: Fourier matrix
For discrete: $F(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)e^{-2\pi i(\frac{ux}{N}, \frac{v}{M})}$
1D-periodic function: $f(t) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{i2\pi nt}{T}}$,
$c_n = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)e^{\frac{-i2\pi nt}{T}}dt$

Properties of Fourier transform

Linearity: $F(ax(t) + by(t)) = aX(t) + bY(t)$
Time Shift: $F(x(t \pm t_0)) = X(t)e^{\pm i2\pi ft_0}$
Frequency Shift: $F(e^{i2\pi f_0 t}x(t)) = X(f - f_0)$
Scaling: $F(x(at)) = \frac{1}{|a|}X\left(\frac{f}{a}\right)$
Convolution, Filtering: The Fourier transfomr of the convolution of two functions is the product of their Fourier transform:
$F \cdot G = U(f **g)$
Convolution, Sampling: The Fourier transfomr of the product of two functions is the convolution of the Fourier transform.
$F **G = U(f \cdot g)$
Duality: $F(X(t)) \longleftrightarrow x(-f)$
Sampling:
A sampling function $s(t)$ which is an impulse train with period T and its Fourier transform $S(f)$:
$S(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$
$S(f) = \frac{1}{T}\sum_{n=-\infty}^{\infty} X(f - \frac{n}{T})$ where $\delta(*)$ Dir.-delt. f.
A continuous signal can be sampled by multiplying with $s(t)$: $x_s(t) = x(t)s(t)$ To compute the Fourier Transform of $x_s(t)$, we can use the convolution theorem:
$F(x_s(t)) = X(t) * S(t) = \frac{1}{T}\sum_{n=-\infty}^{\infty} \delta(f - \frac{n}{T}) * X(t) = \frac{1}{T}\sum_{n=-\infty}^{\infty} X(f - \frac{n}{T})$
Sampling in 2D: $sample_{2D}(f(x,y)) = \sum_{i=\infty}^{\infty}\sum_{j=\infty}^{\infty} f(x,y) * \delta(x - i, x - j) = f(x,y)\sum_{i=\infty}^{\infty}\sum_{j=\infty}^{\infty} \delta(x-i, x-j)$
DFT: The 2D DFT of an image $I(x,y)$ is given by: $F(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} I(x,y) \cdot e^{-j2\pi(\frac{ux}{N} + \frac{vy}{N})}$
$F(f(x,y))(\sum_{i=\infty}^{\infty}\sum_{j=\infty}^{\infty} \delta(x-i, x-j))F(f(x,y)) * F(\sum_{i=\infty}^{\infty}\sum_{j=\infty}^{\infty} \delta(x-i, x-j)) = \sum_{i=\infty}^{\infty}\sum_{j=\infty}^{\infty} F(u - $

$i, v - j)$

Dirac Delta Function:
$\delta(K - k) = \int_{-\infty}^{\infty} e^{2\pi i(K-k)x} dx$

$\int_{-\infty}^{\infty} \delta(t)\, dt = 1$ and $\delta(t) = \begin{cases} 0 & \text{for } x \neq 0 \\ und. & \text{for } x = 0 \end{cases}$

Sifting Property:
$\int_{-\infty}^{\infty} f(t) \cdot \delta(x - a)\, dx = f(a)$

Dirac Comb:
$\text{III}_T(x) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$,
sampling = product with this
TODO!

Box Filter: $\text{rect}(t) = \begin{cases} 1, & \text{if } |t| \leq \frac{1}{2} \\ 0, & \text{otherwise.} \end{cases}$

Triangle Filter: $\text{tri}(t) = \begin{cases} 1 - |t|, & \text{if } |t| \leq 1 \\ 0, & \text{otherwise.} \end{cases}$

Fourier transform of important functions

Nyquist Sampling theorem

The sampling frequency must be at least twice the highest frequency $w_s \geq 2w$ If not the case: band limit before with low-pass filter. Perfect reconstruction: $\text{sinc}(x) =$

$\frac{\sin(\pi x)}{\pi x}$

Why should this hold? Function $f(t)$, sampling function $S_{\Delta t}(t)$ with sampling frequency w_s. Fourier transform of the sampled function can be derived as

$$\tilde{F}(u) = F(f(t) \cdot S_{\Delta t}(t))$$
$$= F(u) * S_{\Delta t}(w)$$
$$= \int_{-\infty}^{\infty} F(\tilde{t}) S_{\Delta t}(w - \tilde{t})\, d\tilde{t}$$
$$= \int_{-\infty}^{\infty} F(\tilde{t}) \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \delta(w - \tilde{t} - \frac{n}{\Delta T})\, d\tilde{t}$$
$$= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F(w - nw_s).$$

If we want to reconstruct the signal $f(t)$ from F and $S_{\Delta t}$, $F(w)$ cannot overlap with its neighbors $F(w - w_s)$ and $F(w + w_s)$. Thus, w_s should be larger than w_n. Highest frequency of $f(t)$.

Image restoration problem: $f(x) \rightarrow h(x) \rightarrow g(x) \rightarrow \tilde{h}(x) \rightarrow f(x)$

The "inverse" kernel $\tilde{h}(x)$ should compensate $h(x)$. May be determined by: $F(\tilde{h})(u,v) \cdot F(h(u,v)) = 1$
Problems: Convolution with kernel k may cancel out some frequencies & noise amplification.

Avoid: Regularization: $F(\tilde{h})(u,v) = \frac{F(h)}{|F(h)|^2 + \varepsilon}$ avoid singularities

6 Unitary Transforms

Vectorization: interpret image as vector row-by-row: $I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$

linear image processing: can be written as $\vec{g} = H\vec{f}$

Image collection (IC): $F = [f_1, f_2 \ldots f_n]$

Autocorrelation matrix $Rff = \frac{F \cdot F^H}{N}$ its Eigenvector with largest Eigenvalue is direction of largest variance among pictures.

Unitary transform: for transform A iff $A^H = A^{-1}$ if real-valued $\rightarrow$ orthonormal every unitary transform is a rotation + sign flip, length conserved

Energy conservation: $||\vec{C}||^2 = \vec{C}^H C = \vec{f}^H A^H A f = ||\vec{f}||^2$

Karhunen-Loeve Transform Same as PCA.
Order by decreasing eigenvalues

Energy concentration property: no other unitary transform packs as much energy in the first J coefficients (for arbitrary J) and mean squared approx-

imation error by choosing only first J coefficients is minimized.

Optimal energy concentration of KLT consider truncated coefficient vector $\vec{b} = I_J\vec{c}$ (I_J: identity matrix with first J columns) Energy in first J coefficients for an arbitrary transform A:
$E = Tr(R_{bb}) = Tr(I_J R_{cc} I_J) = Tr(I_J A R_{ff} A^H I_J) = \sum_{k=0} J = 1 a_k^T R_{ff} a_k^*$ where a_k^T is $k - th$ row of A. Lagrangian cost function to enforce unit-length basis vectors: $L = E + \sum_{k=0}^{J-1} \lambda_k (1 - a_k^T a_k^*) = \sum_{k=0}^{J-1} a_k^T R_{ff} a_k^* + \sum_{k=0}^{J-1} \lambda_k (1 - a_k^T a_k^*)$
Differentiating L with respect to a_j: $R_{ff} a_j^* = \lambda_j a_j^*$ $\forall j < J$ necessary condition

Simple recognition

SSD between images, best match wins very expensive, since need to correlate with every image

Principle Component analysis PCA

Linear dimension reduction method
Optimization goal:
$\operatorname{argmin}_{||w||_2=1, z} \sum_{i=1}^n ||x_i - z_i w||_2^2$
The optimal solution is given by
$z_i = w^\top x_i$.
Substituting gives us:
$\hat{w} = \operatorname{argmax}_{||w||_2=1} w^\top \Sigma w$
Where $\Sigma = \frac{1}{n} \sum_{i=1}^n x_i x_i^\top$ is the empirical covariance. Closed form solution given by the principal eigenvector of Σ, i.e. $w = v_1$ for $\lambda_1 \geq \cdots \geq \lambda_d \geq 0$:
$\Sigma = \sum_{i=1}^d \lambda_i v_i v_i^\top$
For $k > 1$ we have to change the normalization to $W^\top W = I$ then we just take the first k principal eigenvectors so that $W = [v_1, \ldots, v_k]$.

Steps:
- Center image
- Normalize data and subtract mean necessary to ensure first principal component describes direction of maximum variance. Otherwise, first principal component would correspond to mean
- Get Eigenvectors and values from covariance matrix or do SVD
- Sort Eigenvalues and vectors in descending order
- Get j largest components
- Construct projection matrix from selected j Eigenvectors (U_j)
- Transform dataset by multiplying with projection matrix

PCA through SVD
- The first k col of V where $X = USV^\top$.
- first principal component eigenvector of data covariance matrix with largest eigenvalue
- covariance matrix is symmetric $\rightarrow$ all principal components are mutually orthogonal

Kernel PCA
$\Sigma = \frac{1}{n} \sum_{i=1}^n x_i x_i^\top = X^\top X \Rightarrow$ kernel trick:
$\hat{\alpha} = \operatorname{argmax}_\alpha \frac{\alpha^\top K^\top K \alpha}{\alpha^\top K \alpha}$
Closed form solution:
$\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}} v_i$ $K = \sum_{i=1}^n \lambda_i v_i v_i^\top, \lambda_1 \geq \cdots \geq 0$
A point x is projected as: $z_i = \sum_{j=1}^n \alpha_j^{(i)} k(x_j, x)$

Uses of PCA: lossy compression by keeping only the most important k components.
- take the original image I
- apply PCA on the original image, if you do not have a PCA already.
- Compress the image by projecting the image into the PCA subspace. $(I - \mu)U_k$ where U_k is the matrix of the k Principal components.
- apply the inverse PCA transformation from point 2. on the compressed data to get the reconstructed image. $I \cdot U_k^T + \mu$

Calculate units of PCA
Assume dataset of 1000 images, with size 50×50
1. dataset mean = $50 \times 50 = 2500$
2. Truncated eigenmatrix $2500 \times K$
3. Compressed images $1000 \times K$
4. $I_K = (I - \bar{I})\Phi$
5. $\hat{I} = I_K \Phi^T + \bar{I}$

PCA is just a linear transformation from one coordinate system to another, which can easily be "undone" in a lossless manner by reversing the transformation. The dimensionality reduction aspect comes when you start dropping the last principal components, which are the dimensions which capture the least variance.
Face recognition eigenfaces and face detection.

Eigenspace matching

Do PCA with mean subtraction and get closest rank-k approximation of database images (eignfaces)

For a new query: normalize, subtract mean (of database) project to subspace then do similarity matching with eigenfaces.

6.1 Fischerfaces:

Find directions where ratio between / within individual variance is maximized. Linearly project to basis where dimension with good signal: noise ratio is maximized.

$$W_{\text{opt}} = \underset{W}{\arg\max} \frac{\det(WR_BW^H)}{\det(WR_WW^H)}, R_b = \sum R_B \sum_{i=1} cN_i(\vec{\mu}_i - \vec{\mu})(\vec{\mu}_i - \vec{\mu})^H, R_W = \sum_{i=1}^c \sum_{\Gamma_l \in Class}(\Gamma_l - \mu_i)(\Gamma_l - \mu_i)^H$$

Fischer linear discriminant analysis (LDA): maximize between class scatter, while minimizing within less scatter

JPEG Compression

Divide image into 8×8 block:

Discrete cosine transform (DCT): uses only real values and is easier to compute than a Fourier transform.

DC: First coefficient (general intensity)

ZigZag:

Quantization Table: Divide by this value, round to nearest integer, lossy

7 Pyramids and Wavelets

Scale-space representations

From an original signal $f(x)$ generate a parametric family of signals $f^+(x)$ where fine-scale information is successively suppressed e.g. successive smoothing or image pyramids (smooth & downsample)

Applications: Search for correspondence (look at coarse scale, then refine with finer scale) edge tracking coarse to fine estimation control of detail and computational cost (e.g. textures)

Example: CMU face detection: need different scales for template to match.

Gaussian Pyramid: Image pyramid with Gaussian for smoothing

Laplacian Pyramid: Preserve difference between upsampled Gaussian pyramid level and Gaussian pyramid level. Like a band-pass filter - each

level represents spatial frequencies that are largely unrepresented at other layers Compression. **Haar transform:** has two major sub-operations:

1. scaling captures info at different frequencies
2. translation captures info at different locations

8 Optical Flow

Apparent motion of brightness patterns use extracted feature points and commpute their velocity vectors projection of 3D velocity vectors on I

Problem: cannot distingish motion from changing lighting! also estimate observed projected motion field normal flow not always well defined

Key assumptions:

Brightness constancy: Projection of the same point looks the same in every frame.

Small motion: Points do not move far

Spatial coherence: Points move like their neighbors

Brightness constancy constraint:
$I(x + \frac{dx}{dt}\delta t, y + \frac{dy}{dt}\delta t, t + \delta t) = I(x,y,t)$ $I = Intensity$

Small motion $\rightarrow$ can linearize with Taylor expansion:

$\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} \approx 0$ or shorthand $I_x \cdot u + I_y \cdot v + I_t \approx 0$ move $I - t$ on one side, vectorize unknowns. For LK, sum up over a window of pixes

Aperture problem: The aperture problem refers to the fact that when flow is computed for a point that lies along a linear feature, it is not possible to determine the exact location of the corresponding point in the second image. Thus, it is only possible to determine the flow that is normal to the linear feature. 1 equation, 2 unknowns cannot determine exact location, take normal flow.
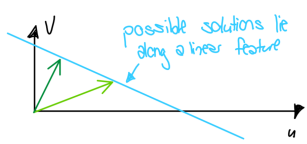
Add additional smoothness constraint:

$e_s = \int\int(u_x^2 + u_y^2 + v_x^2 + v_y^2)dxdy$ close $\approx$ parallel

Besides OF constraint:

$e_c = \int\int(I_xu + I_yv + I_t)^2dxdy$ Minimize $e_s + \lambda e_c$

Lukas-Kanade

Assume spatial coherence: same displacement for neighbourhood ($N \times M$ window) $\rightarrow$ linear

least squares problem:

$$\begin{bmatrix} I_x(x_1, y_1) & I_y(x_1, y_1) \\ \cdots & \cdots \\ I_x(x_{NM}, y_{NM}) & I_y(x_{NM}, y_{NM}) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} I_t(x_1, y_1) \\ \cdots \\ I_t(x_{NM}, y_{NM}) \end{bmatrix} \implies \begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_yI_x & \sum I_yI_y \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum I_xI_t \\ \sum I_yI_t \end{bmatrix}$$

When solvable? A^TA invertible, eigenvalues λ_1, λ_2 large, $\frac{\lambda_1}{\lambda_2}$ small

Errors: motion is large(r than a pixel)

$\rightarrow$ iterative refinement and coarse-to-fine estimation.

A point does not move like its neighbors

$\rightarrow$ motion segmentation.

Brightness constancy does not hold:

$\rightarrow$ exhaustive neighborhood search with normalized corrolation.

The matrix is singular, meaning all gradient vectors point in the same direction.

$\rightarrow$ No unique solution. **KLT feature tracker:** to find patches where LSE well-behaved $\rightarrow$ LK-flow

Iterative refinement: Estimate velocity, warp using estimate, refine,...

Coarse-to-Fine Estimation: Image Pyramid. Start small, compute OF, rescale, take larger and initialize with last estimate

Applications: Image stabilization (get flow between two frames and warp image using same OF for all pixels s.st. OF close to 0) frame interpolation, video compression, object tracking, motion segmentation

Affine motion: $I_x(a_1 + a_2x + a_3y) + I_y(a_4 + a_5x + a_6y) + I_t \approx 0$

SSD tracking: For large displacements: match template against each pixel in small area around, match measure can be (normalized) correlation or SSD choose max. as match (sub-pixel also possible)

Bayesan Optical Flow: Some low-level motion illusions can be explained by adding an underlying model to LK-tracking e.g. brightness constancy with noise.

9 Video Compression

Interlaced video format: 2 temporally shifted half images

$\rightarrow$ increase frequency, decrease spatial resolution $\rightarrow$ not progressive

Lossy video compression: take advantage of redundancy spatial correlation between pixels, temporal correlation between frames

$\rightarrow$ basically drop perceptually unimportant details

with optical flow: Encode optical flow based on previous frame can cause blocking artifacts (if OF of 2 pixels point to same coordinate, there will be a hole somewhere), does not work well for lots of movement, fast movement and scene changes.

If temporal redundancy fails $\rightarrow$ use motion-compensated prediction

Types of coded frames:

I-Frame: Intra-coded frame, coded independently of all others

P-Frame: Predictively coded frame, based on previously coded frame

B-Frame: Bi-directionally coded frame, based on previous & future

Block-Matching Motion Estimation:

Is a type of temporal redundancy reduction

Motion Estimation Algorithm ME

1. Partition frame into blocks (e.g. 16×16 pixels)
2. For each block, find the best matching block in reference frame

Metrics for best match: sum of differences or squared sum of diff.

Candidate blocks: All blocks in e.g. 32×32 pixel area

Search strategies: Full search, partial (fast) search

Motion Compensation Algorithm MC Use the best matching of reference frame as prediction of blocks in current frame

$\rightarrow$ gives motion vectors & MC prediciotn error or residual (encode with conventionl image coder)

Motion Vector: relative horizontal & vertical offsets of a given block from one frame to another

Not limited to integer-pixel offsets, can use half-pixel ME to capture sub-pixel motion.

Half-pixel ME (coarse-fine) algorithm:

1. Coarse step: find best integer move
2. Fine step: refine by spatial interpolation and best-matching

Advantages and disadvanages

+ good, robust performance, one MV per block $\rightarrow$ useful for compression, simple periodic structure (GoP)

- assumes translational motion (fails for complex motion)

$\rightarrow$ codes these frames/blocks without prediction produces blocking artifacts

MPEG-GoP IBBPBBPBBI dependencies be-

tween frames

Scalable Video Coding:

Decompose video into multiple layers of prioritized importance: e.g.

temporal scalability: Include B-frames or not

spatial scalability: Base resolution + upsampling difference

SNR scalability: Base with coarse quatizer + fine quantizer

Benefits: Adapting to different bandwidths, facilitates error resiliency by identifying more and less important bits.

10 CNN

$\varphi(W * v^{(l)})$ For each channel there is a separate filter.

Convolution

$C = channel \ F = filterSize \ inputSize = I$
$padding = P \ stride = S$

Output size $1 = \frac{I+2P-K}{S} + 1$

Output dimension $= l \times l \times m$

Inputs $= W * H * D * C * N$

Trainable parameters $= F * F * C * \#filters$

Neural Networks, d.o.i

w are the weights and $\varphi : \mathbb{R} \mapsto \mathbb{R}$ is a nonlinear **activation function**: $\phi(x, w) = \varphi(w^\top x)$

ReLU: $\max(0, z)$, **Tanh:** $\frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$

Sigmoid: $\frac{1}{1+\exp(-z)}$

Universal Approximation Theorem: We can approximate any arbitrary smooth target function, with 1+ layer with sufficient width.

Forward Propagation

Input: $v^{(0)} = [x; 1]$ Output: $f = W^{(L)} v^{(L-1)}$ Hidden: $z^{(l)} = W^{(l)} v^{(l-1)}, v^{(l)} = [\varphi(z^{(l)}); 1]$

Backpropagation

Non-convex optimization problem:

$$\left(\nabla_{W^{(L)}} \ell\right)^T = \frac{\partial \ell}{\partial W^{(L)}} = \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial W^{(L)}}$$

$$\left(\nabla_{W^{(L-1)}} \ell\right)^T = \frac{\partial \ell}{\partial W^{(L-1)}} = \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial W^{(L-1)}}$$

$$\left(\nabla_{W^{(L-2)}} \ell\right)^T = \frac{\partial \ell}{\partial W^{(L-2)}} = \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial z^{(L-2)}} \frac{\partial z^{(L-2)}}{\partial W^{(L-2)}}$$

Only compute **the gradient**. Rand. init. weights by distr. assumption for φ. ($2/n_{in}$ for ReLu and $1/n_{in}$ or $1/(n_{in} + n_{out})$ for Tanh)

Gradient Descent, i.o.i

Converges only for convex case. $\mathcal{O}(n * k * d)$

$$w^{t+1} = w^t - \eta_t \cdot \nabla \ell(w^t)$$

For linear regression:

$$||w^t - w^*||_2 \leq ||I - \eta X^\top X||_{op}^t ||w^0 - w^*||_2$$

$\rho = ||I - \eta X^\top X||_{op}^t$ conv. speed for const. η. Opt. fixed $\eta = \frac{2}{\lambda_{min} + \lambda_{max}}$ and max. $\eta \leq \frac{2}{\lambda_{max}}$.

Momentum: $w^{t+1} = w^t + \gamma \Delta w^{t-1} - \eta_t \nabla \ell(w^t)$

Learning rate η_t guarantees convergence if $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$

11 Graphics Pipeline

1. Modelling Transform (Object to World Space)
2. Viewing Transform (World to Camera Space)
3. Primitive Processing (Output primitives from transformed vertices)
4. 3D-Clipping (Remove primitives outside the frustum)
5. Screen-Space Projection (Project from 3D to 2D screen space)
6. Scan Conversion (Discretize continuous primitives)
7. Lighting, Shading, Texturing
8. Occlusion Handling (Update Color using Z-buffer)
9. Display

Programmer's View:
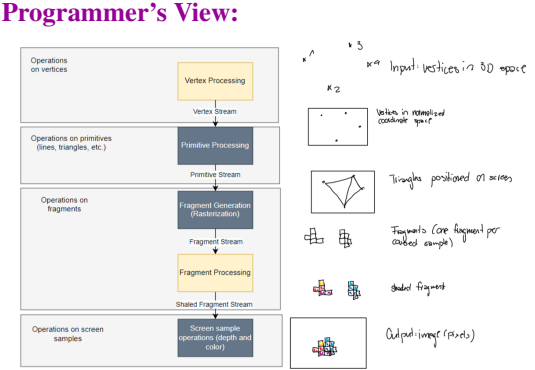
Vertex Processing: Per-vertex operations e.g Transforms and Lighting flow control. This is done with the Vertex Shader. Input: uniforms and per-vertex attributes. Output: Varying per vertex
Fragment Processing: Per-fragment operations

e.g. Shading and Texturing Blending. This is done with the Fragment Shader. Input: Uniform and varying per-fragment attributes. Output: Per-fragment color

Inputs/Outputs:

- Uniforms: (V/F) global constant inputs e.g. light position, texture map etc.
- Varying: (V/F) value passed from vertex to fragment shader by being interpolated across primitives first. e.g interp. pixel color

12 Colors and Light

CIE Experiment: subject is shown two stimuli at the same time, one with the pure spectral color, the other a linear combination of the three primaries (RGB). Subject can control how much primaries were dimmed and asked to match the second stimulus to the first. $\rightarrow$ find how humans perceive color. Can also add red light to reference if impossible to match $\rightarrow$ negative red values.

xyY color space: x,y control chormacity, Y is luminance.

White point: $(x,y) = (0.3, 0.3)$

Purple line: Line connecting 380 and 770mm of non-spectral colors.

Isoline: a line with about constant distance to the border

Inverse color: Intersection of line drawn by the color and white point with the opposite border. **xyY to XYZ:** $X = x\frac{Y}{y}$ $Z = \frac{Y}{y} - x\frac{Y}{y} - Y$

XYZ to xyY: $x = \frac{X}{X+Y+Z}$ $y = \frac{Y}{X+Y+Z}$

RGB: Same color space as XYZ. Can be transformed with matrix multiplication. Additive color model, good for combining colored lights. Used in monitors/displays.

CMY: Inverse of RGB. Subtractive color model. Used in passive color systems (printers).

RGB to CMY: $\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$

YIQ: Luminance Y, In-phase I (orange-blue), Quadrature Q (purple-green) components. Advantages for natural and skin colors. Used in NTSC US-color TV.

HSV: Hue: base color, Saturation: purity of color,

Value: brightness. Intuitive for interactive color picking. Used by designers.

13 Transformations

Linear functions: $f(ax + by) = af(x) + bf(y)$
Homogeneous Coordinates: Raise dimensionality by 1 and set its coordinate to 1.

$$\begin{pmatrix} x & y \end{pmatrix}^T \leftrightarrow \begin{pmatrix} xw & yw & w \end{pmatrix}^T w \in \mathbb{R} \setminus \{0\}$$

This allows non-linear transformations to still be denoted as matrices.

Translation: $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$ **Scale:** $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Rotations: Not commutative. $R^{-1} = R^T$.

3D-rotate(x): $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3D-rotate(y): $\begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3D-rotate(z): $\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

To rotate around arbitrary axes, see Quaternions.

Shear: $\begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_y & 1 & 0 & 0 \\ sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rigid Transformation: Transformation that preserves vector length. (Only rotation & translation)
Change Coordinate Systems:

$p' = \begin{bmatrix} r_1 & r_2 & r_3 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} p$ where r_1, r_2, r_3 are the old axes in the new system and t is the translation from new origin to old origin.

Transform normals with:
$p' = Mp \Rightarrow n' = (M^{-1})^T n$

Quaternions

Rotations and translations efficiently.

$$z = a + bi + cj + dk$$
$$\begin{pmatrix} u & v & w \end{pmatrix}^T \leftrightarrow 0 + ui + vj + wk$$

Properties: $i^2 = j^2 = k^2 = -1$ $ijk = -1$
$ij = k$ $ki = j$ $jk = i$

$ji = -k \quad ik = -j \quad kj = -i$

Vector form: $z = s + \mathbf{v}$ $\mathbf{v}$ is a vector, s is a scalar

Product: $(s_1 + v_1) \cdot (s_2 + v_2) = s_1 s_2 - v_1 \cdot v_2 + s_1 v_2 + s_2 v_1 + v_1 \times v_2$

Conjugate: $\overline{(s_1 + v_1)} = s_1 - v_1, \quad z\bar{z} = \|z\|^2$

Inverse: $z^{-1} = \frac{\bar{z}}{\|z\|^2}, \quad 1 = z^{-1}z = zz^{-1}$

Rotation: Vector $a = (x,y,z)^T$, rotate around u
1. $(x,y,z)^T \rightarrow$ Quaternion $p = 0 + xi + yj + zk$
2. Compute $q = cos(\frac{\theta}{2}) + sin(\frac{\theta}{2})\frac{u}{\|u\|}$ and $q^{-1} = \bar{q}$
3. $p' = qpq^{-1}$

14 Shading and Lighting

Flux: $\Phi(A)[\frac{J}{s} = W]$ total energy/photons passing through space A per time unit.

Radiosity: $B(x) = \frac{d\Phi(A)}{dA(x)}[\frac{W}{m^2}]$ Flux per unit area leaving surface

Irradiance: $E(x) = \frac{d\Phi(A)}{dA(x)}[\frac{W}{m^2}]$ Flux per unit area arriving at surface

(Rad.) Intensity: $I(\vec{\omega})[\frac{W}{sr}]$ Flux per solid angle emanating from point source

Radiance $L(x,\vec{\omega}) = \frac{d^2\Phi(A)}{cos\theta dA(x)d\vec{\omega}}[\frac{W}{m^2sr}]$ Intensity per unit area

BRDF

Bidirectional Reflectance Distribution Function encodes behavior of light that bounces off a surface, given incoming direction ω_i, how much gets reflected in outgoing direction ω_o.

TODO: Add equations

Types of reflections:

diffuse   specular   glossy
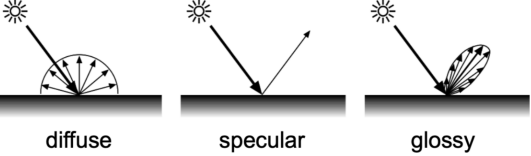
Additionally there is also retro-reflective, which reflects the light back to the source in a way similar to glossy.

Phong Illumination Model

Light shines into the surface but is viewed as an outgoing vertex in the model.

Ambient: Light that shines independent of viewpoint & angle. (Imagine it is object glowing)

Diffuse: General direction of the light which is reflected regardless of viewer's position.

Specular: Shiny light reflection

$$I = \underbrace{I_a k_a}_{\text{Ambient}} + I_p \big(\underbrace{k_d(N \cdot L)}_{\text{Diffusion}} + \underbrace{k_s(R \cdot V)^n}_{\text{Specular}}\big)$$

The material parameters are k_a, k_d, k_s, n. I_a, I_p are light intensities, N normal surface, L the light ray, R the reflection ray, and V the viewing ray. R, V, L, N are all normalised. n is shininess constant. Large n is narrower specular light.

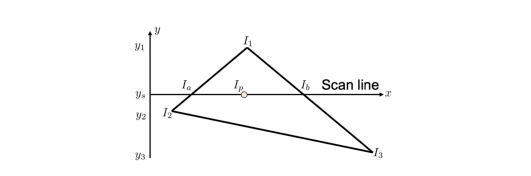$R = \frac{2(N \cdot L)N - L}{\|R\|}$

$V = \frac{Eye\,position - Object\,position}{\|V\|}$

Shading

Flat: 1 color per primitive

Gouraud: Linearly interpolate vertex intensities
1. Calculate vertex normal by averaging face normals.
2. Evaluate illumination model for each vertex
3. Interpolate vertex colors bilinearly on the scan line.

$I_a = I_1 - (I_1 - I_2)\frac{(y_1 - y_s)}{(y_1 - y_2)} \quad I_b = I_1 - (I_1 - I_3)\frac{(y_1 - y_s)}{(y_1 - y_3)}$

$I_p = I_b - (I_b - I_a)\frac{(x_b - x_p)}{(x_b - x_a)}$

Problems: Perspective Distortion. Orientation Dependence due to interpolation. Shared Vertices.

Phong Shading: Linearly interpolate normals
1. Calculate vertex normal by averaging face normals.
2. Interpolate the normal barycentric
3. Evaluate illumination model per fragment in triangle
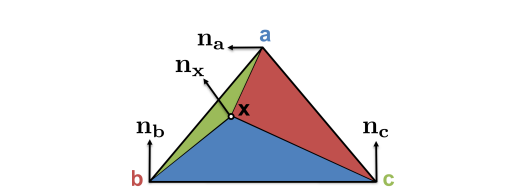
$n_x = \lambda_a n_a + \lambda_b n_b + \lambda_c n_c \quad \lambda_a = \frac{\Delta xbc}{\delta abc} \quad \lambda_b = \frac{\Delta xac}{\delta abc} \quad \lambda_c = 1 - \lambda_a - \lambda_b$

Transparency

Alpha Blending: is the linear interpolation of color front-to-back (obj. 1 is closer than obj. 2):
$I = I_1\alpha_1 + \alpha_2 I_2(1 - \alpha_1)$
$\alpha = 1$: opaque. $\alpha = 0$: transparent.
We render back to front, beginning with opaque object. Can cause issues with overlapping objects. Solution is depth peeling. We do multiple passes where each pass renders the next closest fragment.

15 Geometry & Textures

Challenges, texture: Noisy captured images, visual redundancy over space, callibration inaccuracies, reconstruction inaccuracies, occlusions, visual redundancy over time geometric noise (reconstruction noise & callibrating noise)

Ways to encode geometry:

Explicit: Vertex positions are given explicitly → good for sampling, bad for testing whether inside or outside object.

Implicit: Vertex positions fulfil some equation. → good to test inside/outside object, compact description, tough to model complex shapes, finding all points is expensive.

Geometry representations implicit
- Algebraic surfaces: surface is zero set of polynomial in x, y, z
- Constructive solid geometry: build complicated shapes via Boolean operations
- Blobby surfaces: gradually blend surfaces together (levels of sum of gaussians)
- Blending distance functions: a distance function gives distance to closest point on object
- Level set methods: store a grid of values approximating function
- Fractals and L-systems: no precise definition, structures that exhibit self-similarity, details at all scales, self-similarity, details at all scales

Geometry representations explicit
- Point cloud: list of points (x, y, z), often augmented with normals can represent any gemetry, need large dataset, hard to do processing/simulation, hard to draw if undersampled
- Polygonal mesh: Store vertices and polygons, easier processing simulation, more cimplicated DS, most common
- Triangle mesh: store vertices as triplets (x, y, z) tri-

angles as triples of indices (i, j, k)
- Subdivision surfaces: smooth out a control curve, insert new vertex at each edge midpoint and update vertex positions according to fixed rule

Mesh Datastructure

Triangle List: List containing (v_1, v_2, v_3) where v_i is the coordinates ⇒ easy query, but redundant.

Indexed Face Set: List containing vertex ids and another list of vertices with their coordinates ⇒ less storage space.

Polygonal Mesh

Set of connected polygons where every edge belongs to at least one polygon and the intersection of two polygons either empty, a vertex or and edge.

Manifolds: surface homeomorphic to a disk, closed manifolds divides space into two.

Texture Mapping

Enhance details without increasing geometric complexity. Desirable properties: low distortion, bijective mapping, efficiency.

Parametrization: Map (u, v) coordinates of texture to 3D vertex coordinates. E.g. for spheres
$\begin{bmatrix} u \\ v \end{bmatrix} \mapsto \begin{bmatrix} sin(u)sin(v) \\ cos(v) \\ cos(u)sin(v) \end{bmatrix}$

Texture Filtering: To prevent aliasing, we should apply low pass filter to the texture.

Maps:
- Light map: simulates effect of a local light source
- Environment map: render reflective object efficiently
- Bump mapping
- Normal mapping
- Mipmapping

Bump Mapping: Perturbs surface normal. Encodes height difference (grayscale) from mesh. Illusion of geometry, but (self-)shadows and silhouette unchanged.

Normal mapping: Very similar to bump mapping but now stored as (r, g, b) color ⇒ directional perturbations. More detailed

Mipmapping: Store down-sampled versions of a texture using Gaussian Pyramid. Choose resolution based on projected size of triangle. Use linear interpolation between resolutions. Prevents aliasing!

Magnification: Pixel in texture image maps to

area larger than one pixel → Jaggies. Can be solved by bilinear interpolation.

Minification: Pixel in texture image maps to area smaller than one pixel → moiré patterns. Solution: mipmapping.

16 Signal Processing

17 Scan Conversion

Scan Conversion / Rasterisation: Convert vector-based/geometric objects into pixel-based images. Crucial for rendering graphics on computer screens.

Bresenham Line: Choose closest point at each intersect with vertical pixel grid lines. Implicit line equation: $f(x,y) = ax + by + c = 0$; Last colored pixel: $p = (x_p, y_p)$; $d = f(m) = f(x_p + 1, y_p + 1/2)$; If $d < 0$ select lower pixel E else if $d > 0$ select upper pixel NE. For next pixel, Case E: $d_{new} = f(x_p + 2, y_p + 1/2) = a + d = d + \delta y$; Case NE: $d_{new} = f(x_p + 2, y_p + 3/2) = a + b + d = d + \Delta y - \Delta x$

Scan Conversion for Polygons: Most important graphics primitive; CPU can process up to 50 mil triangles/s; Straightforward approach: inside test for every pixel but instead process scan-line after scan-line; Algorithm: 1. Calculate all intersections on a scan-line, 2. sort intersections by ascending x-coordinates, 3. Fill all spans in between two consecutive intersection points if parity is odd.

18 Bézier/Hermite Curves

Spline desired properties:
Interpolation: Spline passes exactly through data points
Continuity: in C^2
Locality: moving one point does nto affect whole curve
$\implies$ impossible to have all at once Cubic polynomials, interpolate + 1st derivate is given tangent. Interpolates, not C^2-continuous, local **Maps:** $\mathbb{R}^1 \to \mathbb{R}^3 : x(u) = (x(u), y(u), z(u))^T$, $\mathbb{R}^2 \to \mathbb{R}^3 :$ $x(u,v) = (x(u,v), y(u,v), z(u,v))^T$ **Bezier Curves:** $x(t) = b_0 B_0^n(t) + \cdots + b_n B_n^n(t)$ where $b_0...b_n$ are the control points.
$n = 3 : x(t) = b_0(1-t)^3 + 3b_1 t(1-t)^2 + 3b_2 t^2(1-t) + b_3 t^3$.
Derivative: $\frac{d}{dt} b^n(t) = n \sum_{i=0}^{n} \left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t) \right) b_i$, which is a Bezier curve with degree $n-1$
Properties: design property: control points give

rough sketch, endpoint interpolation, variation diminishing property: intersection of straight line with curve <= # control points.

Disadvantages: global support of basis functions (changing one control point changes entire curve), inserting control points expensive, lack of continuity between different segments

Bernstein Polynomial of degree n: $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$ for $0 \le i \le n$ zero else.
Global support, positive definite, partition of unity, different degrees.
Derivative: $\frac{d}{dt} B_i^n(t) = n \left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t) \right)$
Binomial coefficient:
$\binom{n}{i} = \frac{n!}{i!(n-i)!}$ for $0 \le i \le n$ zero else.
DeCasteljau Algorithm: Recursive method for computing a point on a bezier curve using a systolic array in $O(n^2)$: Given $n+1$ control points $b_0, b_1, \ldots, b_n$ the recursion is defined as follows:

$$b_i^r(t) = (1-t)b_i^{r-1}(t) + tb_{i+1}^{r-1}(t)$$
$$b_i^0(t) = b_i$$
$$\text{for } r = 1, \ldots, n \text{ and } i = 0, \ldots, n-r$$

Intuition: Corner cutting until only one line remains whose intersection with the curve is the result.

Forward difference operator $\Delta : \Delta b_j = b_{j+1} - b_j$
Bezier curve derivative with Δ:
$\frac{d}{dt} b^n(t) = n \sum_{j=0}^{n-1} \Delta b_j \cdot B_i^{n-1}$
Recursive Δ^r:
recursive: $\Delta^r b_j = \Delta^{r-1} b_{j+1} - \Delta^{r-1} b_j$
non-recursive: $\Delta^r b_i = \sum_{j=0}^{r} \binom{r}{j}(-1)^{r-j} b_{j+i}$
Higher order derivative of Bezier curve:
$\frac{d^r}{dt^r} b^n(t) = \frac{n!}{(n-r)!} \sum_{j=0}^{n-r} \Delta^r b_j B_i^{n-r}(t)$
Piecewise Bezier Curves / Splines:
Knots: $u_0 < ... < u_L$,
Intervals: $[u_i, u_{i+1}]$,
local parameter: $t = \frac{u-u_i}{u_{i+1}-u_i} = \frac{u-u_i}{\Delta_i}$.
Segment $s(u) = s_i(t)$,
a Bezier curve that is a function of the local parameter t. $\frac{ds(u)}{du} = \frac{ds_i(t)}{dt} \frac{dt}{du} = \frac{1}{\Delta_i} \frac{ds_i(t)}{dt}$.
Enforce Continuity: Curve in $[u_0, u_2]$ decomposed to bezier segments $b_0, ..., b_n$ in $[u_0, u_1]$ and $b_n, ..., b_{2n}$ in $[u_0, u_1]$, $C^r - Continous$ if $b_{n+1} = b_{n-i}^i(t)$ for $i = 0, ..., r$ and $t = \frac{u-u_0}{u_1-u_0}$. $C^1 -$ $Continuity$: Control points $b_n - 1, b_n, b_{n+1}$ are co-linear.

Matrix form: $x(t) = \sum_{i=0}^{n} c_i C_i(t)$. Basis transform into monomial representation with $M = \{m_{ij}\}$:

$$\begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix} = \begin{bmatrix} m_{00} & \cdots & m_{0n} \\ \vdots & \ddots & \vdots \\ m_{n0} & \cdots & m_{nn} \end{bmatrix} \begin{bmatrix} t^0 \\ \vdots \\ t^n \end{bmatrix}$$

For Bernstein: $m_{ij} = (-1)^{j-i} \binom{n}{j}\binom{j}{i}$

Spline interpolation: Interpolate a set of points $p_0, ..., p_n$ using basis functions. For monomials as basis: $p_i = x(t_i) = \sum_{j=0}^{n} a_j (t_i)^j$, $i \in [0,n]$. Resulting in Vandermonde matrix (ill-conditioned):

$$\begin{bmatrix} 1 & t_0 & \cdots & t_0^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & \cdots & t_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} p_0 \\ \vdots \\ p_n \end{bmatrix}$$

Blossoming: Generalisation of deCasteljau. TODO

19 B-Spline Curves

not interpolating, C^2-continuous, local How many knots does a knot vector need to have?: $k + n + 2$ where k = degrees of freedom
B-Spline: $s(u) = \sum_{i=0}^{k} d_i N_i^n(u)$ with deBoor points d_i and knot vector $u = [u_0, ..., u_{k+n+1}]$ (k is degree of freedom and n polynomial degree).
Recurrence: Recurrence relation: $N_i^n(u) = \frac{(u-u_i)}{u_{i+n}-u_i} N_i^{n-1}(u) + \frac{(u_{i+n+1}-u)}{u_{i+n+1}-u_{i+1}} N_{i+1}^{n-1}(u)$, where
$N_i^0(u) = \begin{cases} 1, & u \in [u_i, u_{i+1}) \\ 0, & \text{else} \end{cases}$. B-Spline bases
of degree has support over $n+1$ intervals of the knot vector.
B-Spline filters: Widely used in signal processing. Cardinal B-Splines over uniform knot sequences can be computed using the convolution operator: $N_i^n = N^{n-1} * N^0 = \int_0^x N^{n-1}(t) N^0(x-t) dt$. N^0: box-function.
Properties: Partition of Unity: $\sum_i N_i^n(u) = 1$. Positivity: $N_i^n(u) \ge 0$. Compact support: $N_i^n(u) = 0$, $\forall u \notin [u_i, u_{i+n+1}]$. Continuity: N_i^n is $(n-1)$ times continuously differentiable, if p knots overlap $(u_j = ... = u_{j+p-1})$ only C^{n-p}. Variation diminishing property. Convex hull property.
deBoor Algortihm: We want to evaluate the B spline curve s(u) at point $u = t$. For given $t \in [u_I, u_{I+1}]$ all $N_i^n(u)$ vanish except for $i \in \{I - n, ..., I\}$. Point $s(t)$ computed by successive linear interpolation. Control point in k-th step: $d_i^k = (1 -$

$a_i^k) d_{i-1}^{k-1} + a_i^k d_i^{k-1}$ where $a_i^k = \frac{t-u_i}{u_{i+n+1-k}-u_i}$, $d_i^0 = d_i$, $d_n^n = s(t)$. Special case: If $0 = u_0 = ... = u_n < u_{n+1} = ... = u_{2n+1}$ with $u_{n+k} = 1$ for $k \in [1, ..., n+1]$ we get $d_i^k(u) = u d_i^{k-1}(u) + (1-u) d_{i+1}^{k-1}(u)$ (deCasteljau)

End conditions: TODO
Interpolating B-Splines: TODO

20 Tensor Product Surfaces

2D to 2D mainly used for warping No NURBS
Tensor Product Surface: 2D/3D curve: $x(u) = \sum_{i=0}^{m} c_i F_i(u)$ with bases F_i and coefficients c_i. For surfaces turn coefficients into functions of a second parameter: $c_i(v) = \sum_{j=0}^{v} \alpha_{i,j} G_j(v)$ resulting in the tensor product surface $x(u,v) = \sum_{i=0}^{m} c_i(v) F_i(u) = \sum_{i=0}^{m} \sum_{j=0}^{n} \alpha_{i,j} F_i(u) G_j(v)$
Bezier Patches: Given bezier curve of degree m $b^m(u) = \sum_{i=0}^{m} b_i B_i^m(u)$ and control points b_i as bezier curves of degree n: $b_i = b_i(v) = \sum_{j=0}^{n} b_{i,j} B_j^n(v)$ construct point on the surface: $b_{m,n}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{i,j} B_i^m(u) B_j^n(v)$
TODO: Control Net.
Properties: affine invariance, convex hull, variation diminishing, boundary curves are bezier curves.

2D deCasteljau: TODO?
Warping: Function from 2D to 2D. TODO?
Matrix Form: TODO?
Derivatives: TODO?
Normal Vectors: TODO?
B: TODO?

21 Subdivision Surfaces

Corner Cutting: Insert two new vertices at $\frac{1}{4}$ and $\frac{3}{4}$ of each edge. Remove old and connect new vertices.

Subdivision surfaces: Generalisation of spline curves/surfaces, arbitrary control meshes, successive refinement, converges to smooth limit surface, connection between splines and meshes. In a sense similar to deCasteljau (corner cutting). No regular structure like curves (arbitrary number of edge neighbours, different subdivision rules for each valence).

Classification: Primal: faces are split into sub-faces. Dual: Vertices are split into multiple vertices. Approximating: Control points not interpolated. Interpolating: Control points interpolated.

	Primal		Dual
	Triangles	Rectangles	
Approximating	Loop	Catmull-Clark	Doo-Sabin Midedge
Interpolating	Butterfly	Kobbelt	

Doo-Sabin: generalisation of bi-quadratic B-Splines, for polygonal meshes, generates G^1 continuous surfaces.

Catmull-Clark: generalisation of bi-cubic B-Spline, polygonal meshes, G^2

Loop Subdivision: generalisation of box splines, triangle meshes, G^2

Butterfly: triangle meshes, G^1

Local Subdivision matrix: TODO?

22 Visibility & Shadows

Visibility: Some parts of of some surfaces are occluded by other surfaces.

Painter's Algorithm: Render objects/Polygons from furthest to nearest. Problem: cyclic overlaps and intersections.

Z-Buffering: Store depth to the nearest object for each pixel. 1.Initially all ∞. 2. For each Polygon, if the z value of a pixel for this polygon is smaller than the stored z value, replace the stored z value. Problems: limited resolution (only finite number of z values), non-linear (higher resolution for near objects, lower for far objects), setting near plane far from camera exacerbates resolution problem.

Shadows: Important for perception of depth, realism, indicating light position and type (point light or area light).

Features/Limitations	Planar Fake Shadows	Projective Texture Shadows	Shadow Maps	Shadow Volumes
Allows objects to cast shadows on themselves (self-shadowing)	✗	✗	✓	✓
Permits shadows on arbitrary surfaces (i.e. curved)	✗	✓	✓	✓
Generates extra geometric primitives	✗	✗	✗	✓
Limited resolution of intermediate representation can result in jaggy shadow artifacts	✗	✓	✓	✗

Planar Shadows: Draw projection of object on ground.

Projective texture shadows: Separate obstacle (shadow caster) and receiver. Compute b/w image of the obstacle. Use image as projective texture map.

Shadow Maps: Compute the depths from the light source and from the camera. Shadow map stores depths for light source. For each pixel on the camera plane compute the point x in world coordinates consider its distance z_L to the light source and $d(x_L)$ which is the depth in the direction: light source-x. If $d(x_L) < z_L$, x is in shadow. In order to prevent self-shadowing add bias: $d(x_L) < z_L + bias$, too small bias causes self shadows and to large bias removes too much shadow. In order to include points which are outside the FOV for the shadow map one can use cubical shadow maps. To prevent undersampling/aliasing take weighted average of "$d(x_L) + bias < z_L$" tests instead of filtering depth directly. Bigger filters give fake soft shadows, bias tricky

Shadow Volumes: Explicitly represents the volume of space in shadow. If polygon is inside the volume, it is in shadow. Similar to clipping. Naive implementation: O(#polygons * #lights)

Algorithm::
- Shoot ray from eye.
- Incre-/decrement counter every time boundary of a shadow volume is intersected.
- If counter = 0 not in shadow.

Optimisation: use silhouette edges only (where back-facing and front-facing polygon meet).

Limitations: introduces a lot of new geometry, expensive to rasterize long skinny triangles, objects must be watertight for silhouette optimisation, rasterization of polygons sharing and edge must not overlap or have gap.

23 Ray Tracing

Forward Ray Tracing: light source → object → eye.

Backward Ray Tracing: eye → object (secondary rays may be generated) → light source (in order to compute shadows). Basic pipeline: Ray-generation, Intersection, Shading, Repeat.

Ray generation: pinhole camera???, Supersampling: multiple rays per pixel, prevents aliasing.

Ray-Surface Intersections: For origin o and direction d ray: $r(t) = o + td$. For sphere with center c and radius r solve $\|r(t) - c\|^2 - r^2 = 0$ for t. For triangle with corners p_1, p_2, p_3 use barycentric coordinates $x = s_1 p_1 + s_2 p_2 + s_3 p_3$, Intersect ray with triangle plane: $t = -\frac{(o-p_1)n}{dn}$ where $n = (p_2 - p_1) \times (p_3 - p_1)$, Compute s_i, test $s_1 + s_2 + s_3 = 1$ and $0 \leq s_i \leq 1$

Shading: physically correct too costly, instead assume surface reflectance (diffuse, specular, ambient, transparent), use shadow rays for shadows. Extensions: model refraction, multiple light sources, area light for soft shadows, sample and intersect in time for motion blur, depth of field.

Acceleration: Cost for ray tracing O(#rays * #objects).

Uniform grids:
- Preprocess: Bounding box, grid resolution, rasterize objects, store references to objects.
- Incrementally rasterize ray and stop at intersection with rasterized object.
  Advantages: fast to build, easy to code.
  Disadvantages: not adaptive to scene geometry.

Space partitioning trees: octree, kd-tree, bsp-tree.

24 Animation

Applications: entertainment, education, communication useful in robotics, computer vision engineering and scientific visualization.

Principles of animation:

Squash and stretch: defining the rigidity &mass of an object by distorting its shape during motion

Motion: Spacing actions to define the weight & size of objects & the personality of characters

Anticipation: Preparation of an action

Staging: Presenting an idea so that it is unmistakenly clear

Follow through & overlapping action, straight ahead action & pose-to-pose action, slow in and outgoing

Generating motion:

artist-directed (e.g. keyframing), datadriven (e.g. motion capture), procedural (e.g. simulation)

Keyframing: Specify important events only, computer fills in the rest via interpolation/approximation. Events can be position, color, light intensity, camera configuration.

Motion capture: Track points, compute joint angles using inverse kinematics and use the angles for animation.

25 OpenGL

Resulting Col $=$
Src. Col. $\times$ Src. Fac. $+$ Dest. Col. $\times$ Dest. Fac
projectionMatrix transforms points from camera space to screen space.
modelviewMatrix transforms points from object space to camera space.

26 Radon Transformation

The Radon transform $Rf(\theta, s)$ of a function $f(x, y)$ is defined as:
$$Rf(\theta, s) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta(x\cos(\theta) + y\sin(\theta) - s) \, dx \, dy$$

- θ is the angle of the projection,
- s is the distance parameter,
- $\delta(\cdot)$ represents the Dirac delta function.

27 Rest

angle	0°	30°	45°	60°	90°	120°	135°	150°	180°
	0	$\pi/6$	$\pi/4$	$\pi/3$	$\pi/2$	$2\pi/3$	$3\pi/4$	$5\pi/6$	π
sin	$\frac{\sqrt{0}}{2}$	$\frac{\sqrt{1}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{4}}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{1}}{2}$	$\frac{\sqrt{0}}{2}$
cos	$\frac{\sqrt{4}}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{1}}{2}$	$\frac{\sqrt{0}}{2}$	$-\frac{\sqrt{1}}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{3}}{2}$	$-\frac{\sqrt{4}}{2}$
tan	$\frac{\sqrt{0}}{\sqrt{4}}$	$\frac{\sqrt{1}}{\sqrt{3}}$	$\frac{\sqrt{2}}{\sqrt{2}}$	$\frac{\sqrt{3}}{\sqrt{1}}$	■	$-\frac{\sqrt{3}}{\sqrt{1}}$	$-\frac{\sqrt{2}}{\sqrt{2}}$	$-\frac{\sqrt{1}}{\sqrt{3}}$	$\frac{\sqrt{0}}{\sqrt{4}}$
cot	■	$\frac{\sqrt{3}}{\sqrt{1}}$	$\frac{\sqrt{2}}{\sqrt{2}}$	$\frac{\sqrt{1}}{\sqrt{3}}$	0	$-\frac{\sqrt{1}}{\sqrt{3}}$	$-\frac{\sqrt{2}}{\sqrt{2}}$	$-\frac{\sqrt{3}}{\sqrt{1}}$	■
csc	■	$\frac{2}{\sqrt{1}}$	$\frac{2}{\sqrt{2}}$	$\frac{2}{\sqrt{3}}$	$\frac{2}{\sqrt{4}}$	$\frac{2}{\sqrt{3}}$	$\frac{2}{\sqrt{2}}$	$\frac{2}{\sqrt{1}}$	■
sec	$\frac{2}{\sqrt{4}}$	$\frac{2}{\sqrt{3}}$	$\frac{2}{\sqrt{2}}$	$\frac{2}{\sqrt{1}}$	■	$-\frac{2}{\sqrt{1}}$	$-\frac{2}{\sqrt{2}}$	$-\frac{2}{\sqrt{3}}$	$-\frac{2}{\sqrt{4}}$