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.
Rolling Shutter effect produced by sequential readout of pixels while a digital camera is moving. Result is pixels read at different times are sequentially misaligned, causing image-level distortions dependent on camera (or object) movement.

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:

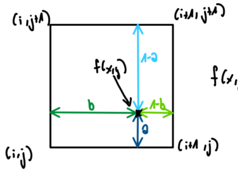
Same sensor elements as CCD, but each sensor has its own amplifier → faster readout, less power consumption, cheaper, more noise.
more noise, lower sensitivity
vs CCD cheaper, lower power, less sensitive, per pixel amplification random pixel access, no blooming, on chip integration

Approach	Prism	Mosaic	Wheel
# Sensors	3	1	1
Separation	High	Avg.	Good
Cost	High	Low	Average
Framerate	High	High	Low
Artifacts	Low	Aliasing	Motion
Bands	3	3	≥ 3
Usage	High-End	Low-end	Scientific

Sampling methods

Cartesian (grid), hexagonal, non-uniform
Quantization: Real valued function will get digital values (integers). A lossy process (original cannot be reconstructed). Simple version: equally spaced $2^b = \#bits$ levels
Linear 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:



$f(x,y) = (1-a)(1-b)f[i,j] + a(1-b)f[i+1,j] + ab\,f[i+1,j+1] + (1-a)b\,f[i,j+1]$

Resolution

- *Image*: px × px
- *Geometric*: #pixels per area
- *Radiometric*: #bits per pixel

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

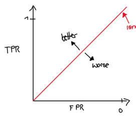
2 Image Segmentation

Complete segmentation

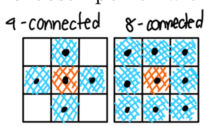
Finite set of non-overlapping regions that cover the whole image $I = \bigcup_{i=1}^{n} R_i$ and $R_i \cap R_j = \emptyset \ \forall i, j, i \neq j$
Thresholding: simple segmentation by comparing greylevel with a threshold to decide if in or out.
Chromakeying: when planning to segment, use special backgroundcolor. (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



$\beta = \frac{N}{P}\frac{V_{TN} + C_{FP}}{V_{TP} + C_{FN}}$ V: Value C: Cost



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 uses mean instead of I_{bg})

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

Operator * mapping image and kernel to images: $I_out = k * I_in$
Local: $I_{out}[i,j]$ depends only on neighbors of $I_{in}[i,j]$
Associative: $((k_1 * k_2) * I) = (k_1 * (k_2 * I))$
Shift invariant: $shift(k * I) = k * shift(I)$
Linear: $k * (\alpha I_1 + \beta I_2) = \alpha(k * I_1) + \beta(k * I_2)$
Linear Combination of neighbors:
$\sum_{(i,j) \in \underbrace{\mathbb{N}(x,y)}_{\text{neighborhood}}} K(x,y,i,j) \underbrace{I}_{\text{Input}}(x+i, y+j)$
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: $(f * g)(t)$
$= \int_{-\infty}^{\infty} f(\tilde{t}) g(t - \tilde{t}) d\tilde{t}$
$= \int_{-\infty}^{\infty} f(t - \tilde{t}) g(\tilde{t}) dt$

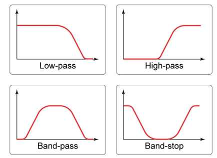
Kernels

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 $(2M+1) \times (2N+1)$, $I'(m,n) = \boxed{f * (\boxed{g * I(N(m,n))})}$
$\boxed{I''(m,n)} = \sum_{j=-N}^{N} g(j) I(m, n-j)$
$\boxed{I'(m,n)} = \sum_{j=-N}^{N} f(i) I''(m-i, n)$
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 ßmooth"regions) Gaussian Filter is a low pass filter, proof: Convolution theorem: Fourier transform H of h is equal to $F \cdot G$ If g is Gaussian, its Fourier Transform G is also Gaussian. Pointwise multiplication of F with G will keep the low frequencies of F unchanged, while the high frequencies will be multiplied by a low number, and therefore, they will be removed.
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 $\in [0,1]$

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

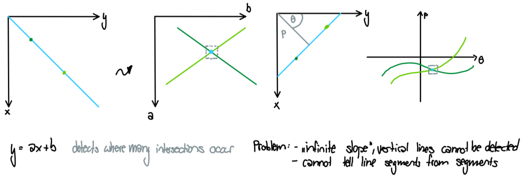
detect discontinuities by considering second derivative $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ or $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ are discrete space approximations. Is isotropic(rotationally invariant), zero crossings make edge locations. Sensitive to fine details and noise (→ 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

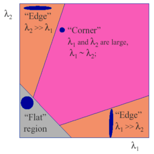
Hough Transform

Fitting a straight line to a set of edge pixels



Alternative parameterization:
$x\cos(\theta) + y\sin(\theta) = \rho$
$(x-a)^2 + (y-a)^2 = r^2$ **For circles:** if r known: calculate circles with radius r around edge pixels → 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)
Each point (x_i, y_i) in the xy-plane gives a sinusoid in the $\theta\rho$ plane. Colinear points lying on the line give curves intersecting at the same point in the polar parameter plane. Local maxima give significant lines.

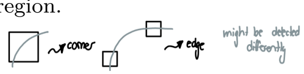
Corner Detection



Edges are only well localized in one direction → 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)

$S(\Delta x, \Delta y) = (\Delta x \Delta y)\left(\sum_{x,y \in \text{window}} \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}\right) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$
$\approx$ 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! $\lambda_1 >> \lambda_2$ → edge, λ_1 and λ_2 large → corner, else → 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. **Limits local gradient**

1. fails when intensity structure within window is poor
2. fails when displacement is large (typical operating range is motion of 1 pixel per iteration!)
3. also brightness is not strictly constant in images

Solution: Pyramid, coarse to fine

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

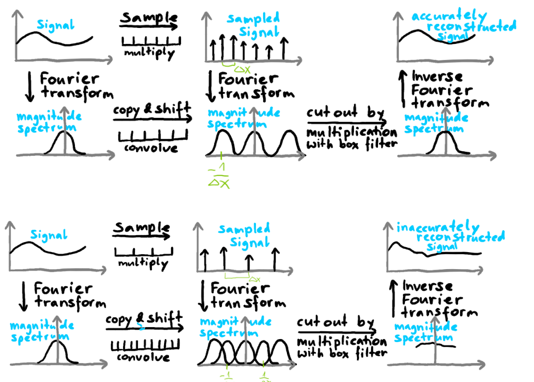
Convolution, Filtering: The Fourier transform of the convolution of two functions is the product of their Fourier transform:
$F \cdot G = U(f ** g)$
Convolution, Sampling: The Fourier transform of the product of two functions is the convolution of the Fourier transform.
$F ** G = U(f \cdot g)$
Represent function on a new basis with basis elements $e^{i2\pi(ux+vy)} = \cos(2\pi(ux+vy)) + i\sin(2\pi(ux+vy))$
$F(f(x))(u) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi ux}dx$,
Inverse Fourier: $f(x) = \int_{\infty}^{\infty} F(u)e^{i2\pi ux}du$ Similar for 2D
2D: $F(f(x,y))(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi i(ux+vy)}dxdy$
For images: transformed image → $F = U * f$ ← vectorized image, U: Fourier matrix
For discrete:
$F(u,v) = \frac{1}{NM}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)e^{-2\pi i(\frac{ux}{N}, \frac{vy}{M})}$
1D-periodic function: $f(t) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{i2\pi nt}{T}}$,
$c_n = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)e^{\frac{-i2\pi nt}{T}}dt$

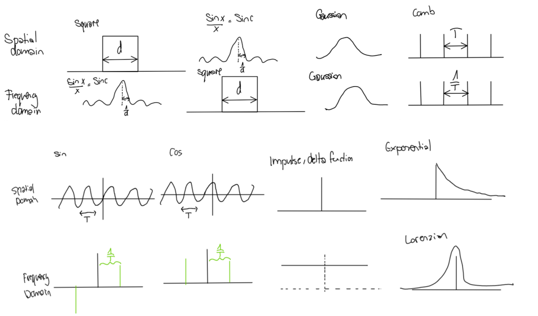
Properties of Fourier transform

Linearity: $F(ax(t) + by(t)) = aX(t) + bY(t)$
Time Shift: $F(x(t \pm t_0)) = X(t)e^{\pm i2\pi f t_0}$
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: $F(x(t) * y(t)) = X(f) \cdot Y(f)$
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:
$F(x_s(t)) = X(t) * S(t) = \frac{1}{T}\sum_{n=-\infty}^{\infty}\delta\left(f - \frac{n}{T}\right) * X(t) = \frac{1}{T}\sum_{n=-\infty}^{\infty} X(f - \frac{n}{T})$

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
Box Filter: $h(x) = \begin{cases} \frac{1}{T}, & \text{if } |x| \leq \frac{T}{2} \\ 0, & \text{otherwise.} \end{cases}$
$f_{recon} = (h * g)(x) =$
$T\int_{-\infty}^{\infty} h(y)\sum_{i=-\infty}^{\infty} f(iT)\delta(x-y-iT)dy$
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: $sinc(x) = \frac{sin(\pi x)}{\pi x}$

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 $\tilde{F}(u) = F(f(t) \cdot S_{\Delta t}(t))$

$= F(u) * S_{\Delta t}(w)$

$= \int_{-\infty}^{\infty} F(\tilde{t}) S_{\Delta t}(w - \tilde{t})\, d\tilde{t}$

$= \int_{-\infty}^{\infty} F(\tilde{t}) \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \delta(w - \tilde{t} - \frac{n}{\Delta T})\, d\tilde{t}$

$= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F(w - n w_s)$.

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:

$f(x) \to h(x) \to g(x) \to \tilde{h}(x) \to f(x)$

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

6 Unitary Transforms

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

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 $\to$ orthonormalevery 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 approximation 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: $R_{ff} a_j^* = \lambda_i a_j^* \quad \forall_j < J$ 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:

$\underset{||w||_2 = 1, z}{\text{argmin}} \sum_{i=1}^{n} ||x_i - z_i w||_2^2$

The optimal solution is given by

$z_i = w^\top x_i$.

Substituting gives us:

$\hat{w} = \text{argmax}_{||w||_2 = 1}\, w^\top \Sigma w$

Where $\Sigma = \frac{1}{n-1} \sum_{i=1}^{n} x_i x_i^\top$ 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 (Number of EV $\leq$ $min(\#pixels, \#datasamples)$)
- 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 $\to$ all principal components are mutually orthogonal

Kernel PCA

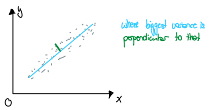
$\Sigma = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^\top = X^\top X \Rightarrow$ kernel trick:

$$\hat{\alpha} = \text{argmax}_\alpha \frac{\alpha^\top K^\top K \alpha}{\alpha^\top K\ \alpha}$$

Closed form solution:

$\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}} v_i \quad K = \sum_{i=1}^{n} \lambda_i v_i v_i^\top, \lambda_1 \geq \cdots \geq 0$

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. $I \cdot U_k^T + \mu$

PCA is just a linear transformation from one coordinate system to another, which can easily be ündoneïn 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.

Calculate units of PCA

> **Exercise:**
> 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. $\hat{I} = I_K \Phi^T + \bar{I}$

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.

Fischerfaces:

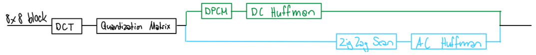
Find directions where ratio between / within individual variance is maximized. Linearly project to basis where dimension with good signal: noise ratio is maximized.

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

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

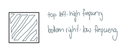
JPEG Compression

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 distinguish 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

Brightness constancy constraint:

$I(x + \frac{dx}{dt}\delta t, y + \frac{dy}{dt}\delta t, t + \delta t) = I(x, y, t) \quad I = Intensity$

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

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

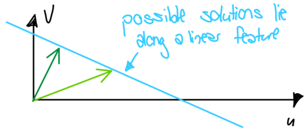
$\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} \approx 0$ or shorthand $I_x \cdot u + I_y \cdot v + I_t \approx 0$

move $I - t$ on one side, vectorize unknowns. For LK, sum up over a window of pixes

> **Derivation:** We assume small displacement and use Taylor-Expansion to get:
> $I(x + \frac{dx}{dt}\delta t, y + \frac{dy}{dt}\delta t, t + \delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}(\frac{dx}{dt}\delta t) + \frac{\partial I}{\partial y}(\frac{dy}{dt}\delta t) + \frac{\partial I}{\partial t}(\delta t)$.
> Subtracting the given equation from this equation, we get:
> $0 = \frac{\partial I}{\partial x}(\frac{dx}{dt}\delta t) + \frac{\partial I}{\partial y}(\frac{dy}{dt}\delta t) + \frac{\partial I}{\partial t}(\delta t)$,
> which can be written as:
> $0 = I_x(\frac{dx}{dt}\delta t) + I_y(\frac{dy}{dt}\delta t) + I_t(\delta t)$
> Finally, we divide by δt, and get:
> $0 = I_x u + I_y v + I_t$,
> as desired.

Sample Exercise:
You have captured a video at 25 frames per second of a car moving at 18 kilometers per hour. The side of the car is parallel to the image plane and the car is moving straight. The car is 2.4 meters long, but in your video it is 192 pixels long. Assume that your optical flow algorithm breaks down for pixel displacements that are larger than 1 pixel.
Start with a coarse image → compute flow → rescale → initialize with the last estimate → repeat.
18 km/h equals 5 meters per second, which equals 20 cm per frame, i.e. $\frac{1}{12}$ of the length of the car. $\frac{1}{12}$ of 192 pixels is 16 pixels. Going from 16 to 8 to 4 to 2 to 1 leads to 5 levels.

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.

Horn-Schunck

Add additional smoothness constraint:
$e_s = \int\int (u_x^2 + u_y^2 + v_x^2 + v_y^2) dxdy$ *close ≈ parallel*
Besides OF constraint:
$e_c = \int\int (I_x u + I_y v + I_t)^2 dxdy$ Minimize $e_s + \lambda e_c$

Lukas-Kanade

Works well for textured area, corners. Not for homogeneous areas, edges since M is singular when all gradient vectors point in the same direction.
Assume spatial coherence: same displacement for neighborhood ($N \times M$ window) → linear least squares problem:

$$\begin{bmatrix} I_x(x_1,y_1) & I_y(x_1,y_1) \\ \dots & \dots \\ I_x(x_{NM},y_{NM}) & I_y(x_{NM},y_{NM}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} I_t(x_1,y_1) \\ \dots \\ I_t(x_{NM},y_{NM}) \end{bmatrix} \implies \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

When solvable? $A^T A$ invertible, eigenvalues λ_1, λ_2 large, $\frac{\lambda_1}{\lambda_2}$ small
Errors: motion is large(r than a pixel)
→ iterative refinement and coarse-to-fine estimation.
A point does not move like its neighbors
→ motion segmentation.
Brightness constancy does not hold:
→ exhaustive neighborhood search with normalized corrolation.
The matrix $M = A^T A$ is singular (for only edges), meaning all gradient vectors point in the same direction.
→ No unique solution. **KLT feature tracker:** to find patches where LSE well-behaved → 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
Parametric (Global) Motion models They offer more constrained solutions than smoothness (Horn-Schunck) and cover larger area than translational model (LK). An example is:
Affine motion: $I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \approx 0$

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.
Parametric Motion can be better:

- more constrained solutions than smoothness (Horn-Schunck)
- intergration over a larger area than a translation-only model can accmmodate (Lucas-Kanade)

9 Video Compression

Interlaced video format: 2 temporally shifted half images
→ increase frequency, decrease spatial resolution → not progressive
Lossy video compression: take advantage of redundancy spatial correlation between pixels, temporal correlation between frames
→ 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 → 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
→ 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 → useful for compression, simple periodic structure (GoP)
- assumes translational motion (fails for complex motion)
→ codes these frames/blocks without prediction produces blocking artifacts
MPEG-GoP IBBPBBPBBI dependencies between 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

Gradient Descent

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:

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

$\rho = ||I - \eta X^\top X||_{op}^t$ conv. speed for const. η. Opt. fixed $\eta = \frac{2}{\lambda_{\min} + \lambda_{\max}}$ and max. $\eta \le \frac{2}{\lambda_{\max}}$. **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 $\sum_t \eta_t^2 < \infty$
Data-Driven Approach $argmin_\theta \mathcal{L}(y, f(x,\theta))$ with x input, θ kernel weights, $f(x,\theta)$ prediction, y target, $\mathcal{L}$ loss function.
Softmax Classifier *scores* = unnormalized log probabilities of different classes. Maximize correct probability:
$P(Y = k \mid X = x_i) = \frac{e^{f_k(x_i,\theta)}}{\sum_j e^{f_j(x_i,\theta)}}$ through the softmax loss:
$\mathcal{L}(y, f(x,\theta)) = -\sum_{i=1}^N \log P(Y = y_i \mid X = x_i)$. Thus minimize negative log likelihood of correct class.
Logistic Classifier Softmax with only two classes $y_i \in \{0,1\}$

$$\mathcal{L}(y, f(x,\theta)) = \frac{1}{N} y_i \log \frac{e^{f(x_i,\theta)}}{1+e^{f(x_i,\theta)}} + (1-y_i) \log \frac{1}{1+e^{f(x_i,\theta)}}$$

Activation Functions

Activation Functions Introduce non-linearity.
Sigmoid $\frac{1}{1+e^{-x}}$, saturated neurons kill the gradient, outputs not zero-centered, compute expensive
tanh $\tanh(x)$, zero centered, still kills gradients
ReLU $\max(0,x)$, does not saturate, very computationally efficient, converges much faster in practice, actually more biologically plausible, not zero-centered output, not differentiable

- **Leaky ReLU**: $\max(0.1x, x)$
- **ELU**: $\begin{cases} x & x \ge 0 \\ a(e^x - 1) & x < 0 \end{cases}$
- **Maxout**: $\max(w_1^\top x + b_1, w_2^\top x + b_2)$

Multilayer Perceptron (MLP)

Stack several linear classifiers with activation function between layers to get *universal approximator*.
Gradient Descent $\theta_{t+1} = \theta_t + \lambda \nabla \mathcal{L}_\theta$ with λ as learning rate.
SGD Approximate loss sum by considering only a batch.
Forwardpropagation $W \in \mathbb{R}^{out \times in}$ *Input layer:* $v^{(0)} = [x; 1]$ *Output layer:* $f = W^{(L)} v^{(L-1)}$ *Hidden layer:* $z^{(l)} = W^{(l)} v^{(l-1)}$ & output with activation and bias $v^{(l)} = [\varphi(z^{(l)}); 1]$.
Given from L+1, compute, given from FP.

$$(\nabla_{W^{(L)}} l)^\top = \frac{\partial l}{\partial f} \frac{\partial f}{\partial W^{(L)}} = \frac{\partial l}{\partial f} v^{(L-1)}$$

$$(\nabla_{W^{(L-1)}} l)^\top = \frac{\partial l}{\partial f} \frac{\partial f}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial W^{(L-1)}} = \cdots v^{(L-2)}$$

$$(\nabla_{W^{(L-2)}} l)^\top = \frac{\partial l}{\partial f} \frac{\partial f}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial z^{(L-2)}} \frac{\partial z^{(L-2)}}{\partial W^{(L-2)}}$$

Where error $\delta^{(l)} = \varphi(z^{(l)}) \odot (W^{(l+1)\top} \delta^{(l_1)})$ and $\nabla_{W^{(l)}} l = \delta^{(l)} v^{(l-1)\top}$ to calculate the gradient.

CNN

Motivation

1. Sparse interactions
2. Parameter sharing

3. Equivariant representations (change the position of an object should not change the classification of it).
4. Hierarchical perception (low-level features to high-level concepts)

CNN-Formulas
$C = channel$ $F = filterSize$ $inputSize = I$ $padding = P$ $stride = S$

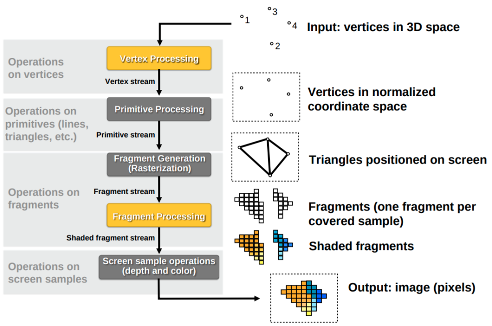
- Output size $l = \frac{I+2P-K}{S} + 1$
- Output dimension $= l \times l \times m$
- Inputs $= W * H * D * C * N$
- Trainable parameters $= F * F * C * \#filters$
- Dimensions: $f(W) \times f(H) \times m, f(i) = \frac{i+2P-K_i}{S} + 1$
- Params: $p = (K_W \cdot K_H \cdot C{+}1) \cdot m$, $+1 \hat{=}$ Bias

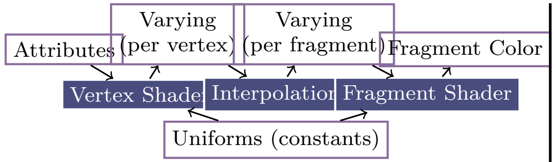
Pooling Layers Pool units to decrease width of output layer. Introduces translation invariance and helps to extract dominant features.
ResNet $v^{(l+1)} = v^{(l)} + r(v^{(l)})$ with skip connections to rely less on depth.
Classification $f(x_i, \theta)$ as the score. Take the class with larger score and use softmax as loss.
Regression $f(x_i, \theta)$ as the value. Can be used for classification by comparing value. Loss could be MSE. Can be used for *depth estimation.*
Pixel Loss, semantic segmentation
$\mathcal{L} = -\sum_i \sum_c y_{ic} \log(p_{ic})$
Optical Flow Loss $\mathcal{L} = \sum_i ((u_i - \hat{u}_i)^2 + (v_i - \hat{v}_i)^2)$
GAN Generate data through randomized input.

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:



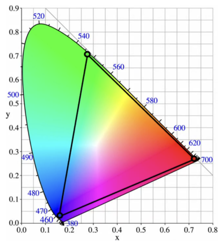


Vertex Processing: Per-vertex operations e.g Transforms and Lighting flow control. This is done with the Vertex Shader. Input: uniforms and per-vertex attributes. Output: Varying per vertex
Fragment Processing: Per-fragment operations e.g. Shading and Texturing Blending. This is done with the Fragment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



Color Gamut Linear combination of 3 colors in $\triangle$.
Purple Line Non-spectral colors between 380 and 770.
Dominant Wavelength From color through whitepoint, boundary intersection.
Saturation Distance from color to white point.
Isoline Line with constant distance to border (w/o PL).

RGB $\rightarrow$ XYZ

$$\begin{bmatrix} \overline{x}(\lambda) \\ \overline{y}(\lambda) \\ \overline{z}(\lambda) \end{bmatrix} = \begin{bmatrix} 2.36 & -0.515 & 0.005 \\ -0.89 & 1.426 & 0.014 \\ -0.46 & 0.088 & 1.009 \end{bmatrix} \begin{bmatrix} \overline{r}(\lambda) \\ \overline{g}(\lambda) \\ \overline{b}(\lambda) \end{bmatrix}$$

XYZ $\rightarrow$ xyY

$$x = \frac{X}{X+Y+Z} \quad y = \frac{Y}{X+Y+Z} \quad Y = Y \quad X = \frac{xY}{y}$$

$$Z = \frac{(1-x-y)Y}{y}$$

RGB $\rightarrow$ CMY

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

RGB $\rightarrow$ YIQ

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

RGB $\rightarrow$ HSV

```
min = min(R, G, B)
max = max(R, G, B)
V = max;
If (max != 0) S = (max - min) / max
Else S = 0;
H = Hue (V, S, R, G, B); // proced.
```

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).

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 in Photoshop.
Lab: CIE does not provide perceptually correct distances. The Lab color space is perceptually uniform, meaning that small changes in the euclidean distance correspond to small changes in perceived color.

13 Transformations

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

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

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

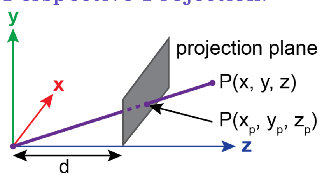
$$z = a + bi + cj + dk$$
$$\begin{pmatrix} u & v & w \end{pmatrix}^T \leftrightarrow 0 + ui + vj + wk$$

Properties: $i^2 = j^2 = k^2 = -1 \quad ijk = -1$
$ij = k \quad ki = j \quad jk = i$
$ji = -k \quad ik = -j \quad kj = -i$
Vector form: $z = s + \mathbf{v}$ $\quad \mathbf{v}$ 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: $\overline{(s_1 + v_1)} = s_1 - v_1, \quad z\overline{z} = \|z\|^2$
Inverse: $z^{-1} = \frac{\overline{z}}{\|z\|^2}, \quad 1 = z^{-1}z = zz^{-1}$
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 $q^{-1} = \overline{q}$
3. $p' = qpq^{-1}$

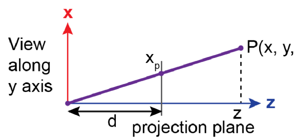
Projections

Perspective Projection:



You can imagine the projection plane to be the screen space and the origin the camera.
$x_p = dx/z \quad y_p = dy/z \quad z = d$

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$



Triangle rule: $x_p/d = x/z$
Parallel Projection: Set the coordinate of the orthogonal of the plane to 0. Assuming the projection plane is x,y, we set z to 0:

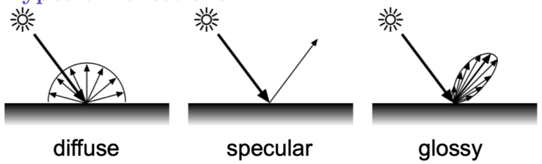
$$M_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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 $L(x,\vec{\omega})=\frac{d^2\Phi(A)}{cos\theta dA(x)d\vec{\omega}}[\frac{W}{m^2 sr}]$ 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.
Reflection function:
$f_r(x,\vec{\omega_i},\vec{\omega_r})=\frac{dL_r(x,\vec{\omega_r})}{L_i(x,\vec{\omega_i})cos\theta_i d\vec{\omega_i}}$
ω_i is the incoming light vector, ω_r the reflected. θ_i: angle of incoming vector to the surface normal.
f_r is constant for diffuse reflections. **Reflection Equation:** Reflected radiance due to illumination from all directions.
$L_r(x,\vec{\omega_r})=\int_{H^2} f_r(x,\vec{\omega_i},\vec{\omega_r})L_i(x,\vec{\omega_i})cos\theta_i d\vec{\omega_i}$
For diffuse reflections, f_r is constant.
$L_r(x)=f_r E_i(x)=f_r\int_{H^2} L_i(x,\vec{\omega_i})cos\theta_i d\vec{\omega_i}$
Types of reflections:



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

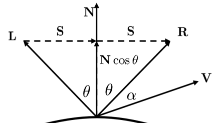
Phong Illumination Model

This is a local illumination model: does not consider indirect light bouncing of from others objects that are hitting the object, unlike the global illumination model. It is approximated by ambient lighting. Light shines into the surface but is viewed as an outgoing vector in the model.
Ambient: Light that shines independent of viewpoint & angle. (Imagine it as 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.

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

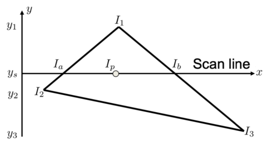


Attenuation Quadratic due to spatial radiation. $f_{att}=(d_L^2)^{-1}$ or often used in OpenGL: $f_{att}=\min((c_1+c_2 d_L+c_3 d_L^2)^{-1},1)$
Cook-Torrence For metal objects which replaces the specular term. Has self-shadowing effects.
Ashikhmin Anisotropic lighting model.

Shading

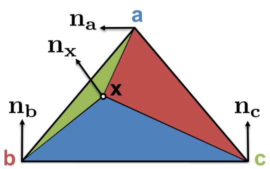
Flat: 1 color per primitive, per triangle
Gouraud: Linearly interpolate vertex intensities

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



$I_a=I_1-(I_1-I_2)\frac{(y_1-y_s)}{(y_1-y_2)}$ $I_b=I_1-(I_1-I_3)\frac{(y_1-y_s)}{(y_1-y_3)}$
$I_p=I_b-(I_b-I_a)\frac{(x_b-x_p)}{(x_b-x_a)}$
Problems: Perspective Distortion. Orientation Dependence due to interpolation. Shared Vertices.
Phong Shading: Linearly interpolate normals, color per pixel, problem: normal not defined/representative

1. Calculate vertex normal by averaging face normals.
2. Interpolate the normal barycentric
3. Evaluate illumination model per fragment in triangle



$n_x=\lambda_a n_a+\lambda_b n_b+\lambda_c n_c$ $\lambda_a=\frac{\Delta xbc}{\Delta abc}$ $\lambda_b=\frac{\Delta xac}{\Delta abc}$ $\lambda_c=1-\lambda_a-\lambda_b$

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
- Signed Distance Function

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) triangles 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

Supersampling

We sample multiple times per pixel for the most accurate color. Final color of pixel averaged from the samples that fall into this pixel. We have different patterns like uniform, jittering, stochastic, poisson.
Lose high frequency information.

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

Exercise: If Uniform Interval, just plug in values. If not replace t by $\frac{x-x_k}{x_{k+1}-x_k}$

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, global
Maps: $\mathbb{R}^1 \to \mathbb{R}^3 : x(u) = (x(u), y(u), z(u))^T$
$\mathbb{R}^2 \to \mathbb{R}^3 : x(u,v) = (x(u,v), y(u,v), z(u,v))^T$
Special cases of B-Spline 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, adding new points increases the degree.
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:

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

Intuition: Corner cutting until only one line remains whose intersection with the curve is the result.
Forward difference operator Δ **:** $\Delta b_j = b_{j+1} - b_j$
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: $\frac{d^r}{dt^r} b^n(t) = \frac{n!}{(n-r)!} \sum_{j=0}^{n-r} \Delta^r b_j B_j^{n-r}(t)$
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

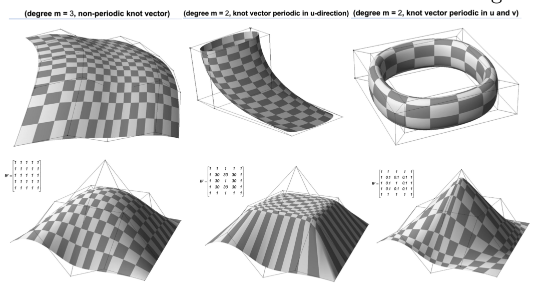
$\frac{ds(u)}{du} = \frac{ds_i(t)}{dt} \frac{dt}{du} = \frac{1}{\Delta_i} \frac{ds_i(t)}{dt}$.
Enforce Continuity: Curve in $[u_0, u_2]$ decomposed to bezier segments $b_0, ..., b_n$ in $[u_0, u_1]$ and $b_n, ..., b_{2n}$ in $[u_0, u_1]$, $C^r - Continous$ if $b_{n+1} = b_{n-i}^i(t)$ for $i = 0, ..., r$ and $t = \frac{u - u_0}{u_1 - u_0}$.
$C^1 - Continuity$: Control points $b_n - 1, b_n, b_{n+1}$ are colinear.
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: $p_i = x(t_i) = \sum_{j=0}^{n} a_j (t_i)^j, \quad i \in [0, n]$. 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.

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 and n = polynomial degree
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}, higher continuity leads to smother transitions between different segments and smooth derivative curves. 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: How curve behaves at end points. For closed loop periodic deBoor points and knot vector: $d_0 = d_{k++}, u_0 = u_{k+1}$

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: $b_{m,n}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{i,j} B_i^m(u) B_j^n(v)$
Properties: affine invariance, convex hull, variation diminishing, boundary curves are bezier curves.
2D deCasteljau: Algorithm for computing point on surface.
Warping: Function from 2D to 2D, distorting an image
NURBS: Non uniform rational b-splines. $\neq$ Tensor Product Surfaces since bases not separable. Top row: different B-splines, bottom row: nurb surface with different weigths



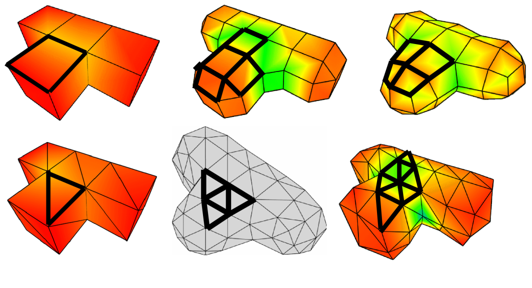
21 Subdivision Surfaces

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 subfaces. 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	

Geometric continuity: Weaker form of continuity focusing on the visual appearance, e.g. G^n curve might be C^{n-1} for a finite set of points and C^n everywhere else.
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 continuous
Top row: Start, Doo-Sabin, Catmull-Clark.
Bottom-row: Start, Loop Subdivision, Butterfly



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: li-

mited 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 on the ground.
Limitations: No self shadows or on other objects. Problems with curved surfaces.

Projective texture shadows Separate obstacle and receiver. Compute b/w image from light and use as projective texture.
Limitations: Need to specify obstacle & receiver. No self-shadows.

Shadow Maps

1. Compute depths from light $d(x_L)$ and camera.
2. For each pixel in camera plane:
   (a) Compute point in world coordinates
   (b) Project onto light plane z_L
   (c) If $d(x_L) < z_L$, then x is in shadow.
3. Add bias for stability ($d(x_L)+b < z_L$).
4. A point to shadow can be outside the FoV of shadow map, thus use cubical shadow map or spot lights.
5. Should **not** filter depth, but take weighted average.

Shadow Volumes

1. Explicitly represent the volume of space in shadow. If polygon in volume, it is in shadow.
   (a) Shoot a ray from the camera
   (b) ++/– counter each time volume is intersected.
   (c) if counter >0, then primitive is in shadow
2. Use silhouette edges only!
3. *Limitations*:
   (a) Lots of geometry
   (b) Expensive to rasterize long skinny triangles
   (c) Object must be watertight
   (d) Rasterization of polygons sharing an edge must not overlap and not have gap.

23 Ray Tracing

Rasterization vs Raycasting

Rasterization: Proceed in triangle order, most processing based on 2D primitives (3D that was projected. Store depth buffer).

Raytracing: Proceeds in screen sample order, never have to store depth buffer (just current ray), natural order for rendering transparent surfaces. Must store entire scene.

Shadow mapping: Render scene (depth buffer only) from location of light. Everything ßeen"(depth test success) from this PoV is directly lit, if depth test fail → shadow.

Shadows ray tracing: shoot ßhadow"rays towards light source from points where camera rays intersect scene. If nothing in the way → lighted, else → shadow.

Environment mapping: approximate appearance of reflective surface by placing a ray origin at location of reflective object, render six views (for a cube). Use camera ray reflected about surface normal to determine which texel in cube is "hit".

Reflections: ray tracing: recursive ray tracing, compute a secondary ray from surface in reflection direction.

Ray Casting Shoot ray through from the camera through the pixels and in first intersection, evaluate the illumination model.

Forward Raytracing Rays from light source (not efficient).

Backward Raytracing Shoot rays from the camera.

The Pipeline

1. *Ray Generation*: Shoot ray from origin.
2. *Intersection*: Calculate first intersection. Calculate illumination at that point by recursion (either reflect or refract).
3. *Shading*: Shoot ray from intersection to directly to light source. Intersection $\implies$ Point in shadow.

Supersampling Shoot multiple rays to remove aliasing.

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).

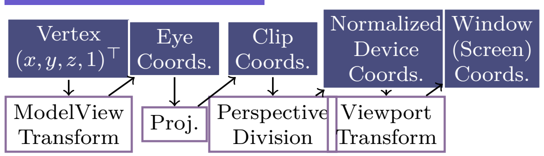
Uniform grids:

- 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 OpenGL

OpenGL Transformations



Model View Transform First model to world coordinates: $\begin{bmatrix} r_1 & r_2 & r_3 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \text{model} \\ 1 \end{bmatrix} = \begin{bmatrix} \text{world} \\ 1 \end{bmatrix}$ Then world to camera:

$$\begin{bmatrix} \text{left} & \text{up} & \text{-dir} & \text{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \text{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} \text{world} \\ 1 \end{bmatrix}$$

Projection
Either parallel:

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{n-f} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c \\ 1 \end{bmatrix} = \begin{bmatrix} c' \\ 1 \end{bmatrix}$$

Or perspective:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} c \\ 1 \end{bmatrix} = \begin{bmatrix} c' \\ -c_z \end{bmatrix}$$

Perspective Divison $\frac{1}{-c_z}c' = \begin{bmatrix} d_x & d_y & d_z \end{bmatrix}^\top$ which are the normalized device coordinates. d_x, d_y position and d_z depth.

Viewport Transform screen cord. $= \begin{bmatrix} \frac{w}{2}d_x + (o_x + \frac{w}{2}) \\ \frac{h}{2}d_y + (o_y + \frac{h}{2}) \\ \frac{f-n}{2}d_z + \frac{f+n}{2} \end{bmatrix}$

25 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.

Properties

- Linear
- Shifting only changes the ρ coordinate
- Rotation of the coordinate system also rotates the Radon transformation
- The Radon transform of a 2D convolution is a 1D convolution of the Radon transformed function with respect to ρ

Reconstructing Image
Assume: attenuation of material in each px constant and $\propto$ area of the px illuminated by the beam. $k_{ij} = \frac{\text{are of pixel } j \text{ illuminated by ray } i}{\text{total area of pixel } j}$ for $i \in [l], j \in [nm]$. Thus the model reads: $Kf = g$ with f BW plane/volumetric image to be retrieved, g attenuation measurement from the CT system. Can be solved with normal equations. Big system!

Central Slice Theorem $G(q,0) = F(q\cos 0, q\sin 0)$. 1D Fourier transformation of the measurement $g = Rf$ (for fixed θ) is equal to 2D Fourier trans. of $f(x,y)$ at a particular point.

Filtered backprojection

1. Measure attenuation (projection) data
2. 1D-FT of projection data
3. High-Pass filter in Fourier domain ($2\pi|w|/K$)
4. 2D-Inverse FT
5. Sum over all images

Issues without HPF:

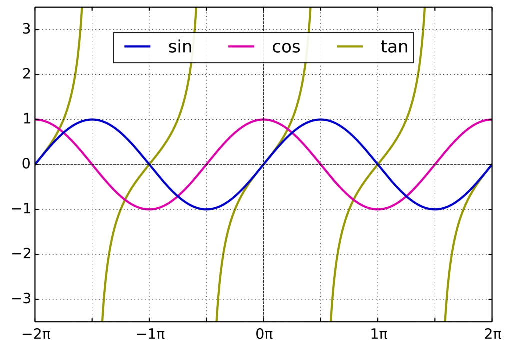
- Requires many precise attenuation measurements
- Sensitive to noise
- Unstable & hard to implement accurately
- blurring the final image

25 Math

Trigometry

$sin(x) = \frac{e^{ix}-e^{-ix}}{2i}$ $\quad cos(x) = \frac{e^{ix}+e^{-ix}}{2}$
$sin(2x) = 2sin(x)cos(x)$
$cos(2x) = cos^2(x) - sin^2(x)$
$sin(x+y) = sin(x)cos(y) + cos(x)sin(y)$
$cos(x+y) = cos(x)cos(y) + sin(x)cos(y)$
$sin^2(x) + cos^2(x) = 1$

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	$\sqrt{\frac{0}{4}}$	$\sqrt{\frac{1}{3}}$	$\sqrt{\frac{2}{2}}$	$\sqrt{\frac{3}{1}}$	■	$-\sqrt{\frac{3}{1}}$	$-\sqrt{\frac{2}{2}}$	$-\sqrt{\frac{1}{3}}$	$-\sqrt{\frac{0}{4}}$
cot	■	$\sqrt{\frac{3}{1}}$	$\sqrt{\frac{2}{2}}$	$\sqrt{\frac{1}{3}}$	0	$-\sqrt{\frac{1}{3}}$	$-\sqrt{\frac{2}{2}}$	$-\sqrt{\frac{3}{1}}$	■
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	$\frac{2}{\sqrt{4}}$	$\frac{2}{\sqrt{3}}$	$\frac{2}{\sqrt{2}}$	$\frac{2}{\sqrt{1}}$	■	$-\frac{2}{\sqrt{1}}$	$-\frac{2}{\sqrt{2}}$	$-\frac{2}{\sqrt{3}}$	$-\frac{2}{\sqrt{4}}$



Distance between two points: $(a_1,a_2) and (b_1,b_2) \to (a_1-b_1)+(a_2-b_2)$

Closest point on a 2d line: 2D line: $N^Tx = c$ plug in $N^T(p+tN) = c$ and compute $p+tN = p+(c-N^Tp)N$

Closest point line segment: find closest point on line, then check if between endpoints $(a+t(b-a)$ check if $t \in [0,1])$ else closest endpoint

Point-line intersection: plug point in line equation.

Line-line intersection: $ax = b, cx = d \to \begin{smallmatrix} a1 & a2 \\ c1 & c2 \end{smallmatrix} \begin{smallmatrix} x1 \\ x2 \end{smallmatrix} = \begin{smallmatrix} b \\ d \end{smallmatrix}$

Intersecting ray with implicit surface: All points s.t. $f(x) = 0$ and ray $r(t) = o+td \implies$ solve $f(r(t)) = 0$ for t

Ray-plane intersection: Given plane $N^Tx = c$, ray $r(t) = o+td$, replace x with ray equation, solve for $t \implies$ point $= o + \frac{c-N^To}{N^Td}d$

Ray-triangle intersection: Parameterize triangle given by vertices p_0, p_1, p_2 (barycent.coords) $f(u,v) = (1-u-v)p_0 + up_1 + p_2$ solve for u,v,t : $[p_1-p_0, p_2-p_0-d][uvt]^T = o-p_0$