Digital Images

Image: Discrete sample of continuous 2D sign.

Charge Coupled Device (CCD): Matrix of photosites which accumulate charge relative their exposure. Read out line by line
- expensive - power consumption - sequential readout

Blooming: Bucket overfills and spills neighbouring buckets. Esp. allong connected buckets

Bleeding/Smearing: Buckets exposed while transf.

Dark Current: Random input even if dark

Complementary metal-oxide semicond. (CMOS)
+cheap +low power - less sensitive +per pixel amplification
as CCD but each pixel has own amplifier
+random pixel access + smart pixels - more noise

Rolling Shutter: Seq. read of lines

Properties: • Image Resolution: crop
• Geometric Resolution: fixed area, different # pixel • Radiometric Resolution: fixed area, different # bits for color

Noise: • Additive Noise: $I(x,y) = f(x,y) + c$ modelled using Gauss or Poisson
• Multiplicative Noise: $I(x,y) = f(x,y) + f(x,y)c$ due to quantisation error, "salt-and-pepper"

Signal to Noise Ratio (SNR): Measure of image quality, $s = \frac{E}{\sigma}$, $E = \frac{1}{xy}\sum_{x=1}^{x}\sum_{y=1}^{y} f(x,y)$

• Peak SNR (PSNR): $s_{peak} = \frac{E_{max}}{\sigma}$

Color: • Prism: split color on 3 sensors
• Filter Mosaic: diff filter on diff pixel
• Filter Wheel: diff img. with diff filters

Segmentation

Divide pixels in two or more sets

$I = \bigcup_{i=1}^{N} R_i$ and $R_i \cap R_j = \emptyset \; \forall i,j, i \neq j$

Thresholding: Binary segmentation B of image I according to pixel brightness:
$B(x,y) = \begin{cases} 1 & \text{if } I(x,y) \geq T \\ 0 & \text{otherwise} \end{cases}$

Chroma keyin: Use blue or green screen, Improve results by ~~subtracting~~ dividing by ~~average~~ total brightness

• Distance Measure: Euclidean: one color
$I_\alpha = |I - g| > T$, g: Bg color (only one)

Mahalanobis: set of colors (or only one)
+takes variation of the distribution into account
$I_\alpha = \sqrt{(I-\mu)^T \Sigma^{-1} (I-\mu)} > T$, μ: mean of samples,
+gives different weight to different color channels
$\Sigma_{ij} = cov(i,j) = E(X_i - \mu_i)(X_j - \mu_j)$
$\approx \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$

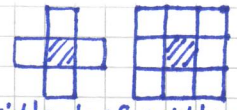
• Pixelwise Color Model: Bg has non-uniform color; use pixelwise distance between Bg image and (Bg + Fg) image

Receiver Operating Characteristic (ROC): Quality analysis of binary classifier; plot $\underbrace{TP/(TP+FN)}_{TPR}$ vs $\underbrace{FP/(FP+TN)}_{FPR}$

• Operating Points: Assign value V and cost C to all 4 outcomes; choose pt with gradient
$\beta = \frac{N}{P} \cdot \frac{V_{TN} + C_{FP}}{V_{TP} + C_{FN}}$

Markov Random Field (MRF):
$E(y; \theta, data) = \sum_i \Psi(y_i; \theta, data) + \sum_{i,j \in edge} \Psi_2(y_i, y_j; \theta, data)$; solved using Max-Flow, Min-Cut

Surface Coherence:



• Neighbourhood: 4-neighbourh.; 8-neighbourh.

• Path: 4/8 connected path between p_1, p_n is set $\{p_1, \dots p_n\}$ s.t. p_i and p_{i+1} are 4/8 NB

• Region: 4/8 connected region is set $\{p_1, \dots p_n\}$ s.t. p_i and p_j $\forall i,j, i \neq j$ $\exists$ 4/8 pixel path;

• Components: different connected regions; labeling allows Segmentation and counting of Obj.

• Region growing: Start with seed pt/region and add neighbour pixels as long as they satisfy some criteria (e.g. threshold)

• Snake: Polygon (set of pts connected by lines), pts move outwards as long as its neighbourhood satisfies some criteria; iteratively minimizes $E = E_{tension} + E_{stiffness} + E_{img}$

Morphological Operations: Local transforms based on comparison of pixel neighbourhood with pattern; smooths boundaries and removes noise/artefacts • Structural Elem.: S

• Fit: $\{y \mid y = x + s, s \in S\} \subset I$
↳ i.e. template matching with mid. pt.

• Hit: $\{y \mid y = x - s, s \in S\} \cap I \neq \emptyset$

• Miss: $\{y \mid y = x - s, s \in S\} \cap I = \emptyset$

1 of pattern must match with 1 of image; 0 of pattern is like "don't care"

• Erosion: $E := I \ominus S$
$E(x) = \begin{cases} 1 & \text{if } S \text{ fits } I \text{ at } x \\ 0 & \text{otherwise} \end{cases}$; $E = \{x \mid x + s \in I \;\; \forall s \in S\}$

• Dilate: $D := I \oplus S$
$D(x) = \begin{cases} 1 & \text{if } S \text{ hits } I \text{ at } x \\ 0 & \text{otherwise} \end{cases}$; $D = \{x \mid x - s, y \in I \wedge s \in S\}$

• Open: $I \circ S := (I \ominus S) \oplus S$
- remove noise

• Close: $I \bullet S := (I \oplus S) \ominus S$
- close holes

consecutive open + close / close + open opens/closes holes in a segment + removes/enlarges islands

• granulometry: Count granules (= connected region) in image by open the image with increasing structure element size and count # regions

• Hit-Miss-Transform: $H = I \otimes S$; hit if whole structure element matches, else miss

• Thin: $I \oslash S := I \setminus (I \otimes S)$ } Dual:
• Thick: $I \odot S := I \cup (I \otimes S)$ } $(I \circ S)^c = I^c \oslash S$

Applied in sequence: $I * \{S_i \mid i \in 1 \dots n\} = (((I * S_1) * S_2) \dots * S_3)$, $* \in \{\oslash, \odot\}$

• Skeleton: Pt where two grassfires meet; $S(x)$

• Medial Axis Transform (MAT): Skeleton with the maximal disc radius retained at each pt

• Skeletonization: use struct. elem. $B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

• n-th skeleton subset: $S_n(X) = (X \ominus_n B) \setminus ((X \ominus_n B) \circ B)$

• skeleton: $S(X) = \bigcup_{n=1}^{\infty} S_n(X)$

• Reconstruction: $X = \bigcup_{n=0}^{\infty} S_n(X) \oplus_n B$
works only for MAT

Filter

Properties: • **local:** op depends on pixel neighbourhood $N(x,y)$ • **linear:** $L(\alpha I_1 + \beta I_2) = \alpha L I_1 + \beta L I_2$; $I'(x,y) = \sum_{(i,j)\in N(x,y)} K(x,y;i,j)\, I(i,j)$ • **Shift-Invariant:** same op for all pixels; K is indep of (x,y)

Correlation: • $I' = K \circ I$; • $I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j)\, I(x+i, y+j)$; not assoc

Convolution: • $I' = K * I$; • $I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j)\, I(x-i, y-j) = \sum_{(i,j)\in N(x,y)} K(-i,-j)\, I(x+i, y+j)$; is assoc;

• If $K(i,j) = K(-i,-j) \Rightarrow$ Corr = Conv

Fill Edges: • clip filter; • wrap around; • copy edge; • reflect across edge; • vary filter

Separable Filter: $K(m,n) = g(m) f(n)$ ⇔ rank$(K) = 1$

Kernels: • **Box:** $\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ • **Weighted Smooth:** $\frac{1}{10}\begin{bmatrix}1&1&1\\1&2&1\\1&1&1\end{bmatrix}$, $\frac{1}{16}\begin{bmatrix}1&2&1\\2&8&2\\1&2&1\end{bmatrix}$ • **gauss:** $G_\sigma = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$; + separable + rotationally symmetric + single lobe + single lobe in frequ. dom. + simple

• **Prewitt:** $\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$ • **Sobel:** $\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}$ • **Laplacian:** $\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$ • **High-Pass:** $\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$ • **gradient:** $\{-1, 1\}$

Features

Template Matching: 1. remove mean brightness from I gives s 2. convolve s with t: $r(p,q) = \sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty} s(x,y)\, t(x-p, y-q) = s(p,q) * t(-p,-q) \le \sqrt{\left(\sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty} |s(x,y)|^2\right)\left(\sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty} |t(x,y)|^2\right)}$ 3. search peaks which give obj. locations; alternat. min MSE: $E(p,q) = \sum_{x,y=-\infty}^{\infty} (s(x,y) - t(x-p, y-q))^2 = \sum_{x,y=-\infty}^{\infty} |s(x,y)|^2 + \sum_{x,y=-\infty}^{\infty} |t(x,y)|^2 - 2\sum_{x,y=-\infty}^{\infty} s(x,y)\, t(x-p, y-q)$

Edge Detection: local max of first deriv., zero crossing of second deriv.

• **Gradient Vector:** $g(x,y) = \begin{bmatrix} g_x(x,y) \\ g_y(x,y)\end{bmatrix} = \begin{bmatrix}(K_x * I)(x,y) \\ (K_y * I)(x,y)\end{bmatrix}$

• **Gradient Magnitude:** $|g| = \sqrt{g_x^2 + g_y^2}$

• **Gradient Direction:** $\theta = \arctan\left(\frac{g_y}{g_x}\right)$

• **Derivative Kernel K_x/K_y:** • **Difference:** $I_x(x_0,y_0) \approx I(x_0+1, y_0) - I(x_0, y_0) = ([-1,1] * I)(x_0,y_0)$ $[-1,1]$, $\begin{bmatrix}-1\\1\end{bmatrix}$ • **Central Difference:** $[-1, 0, 1]$, $\begin{bmatrix}-1\\0\\1\end{bmatrix}$

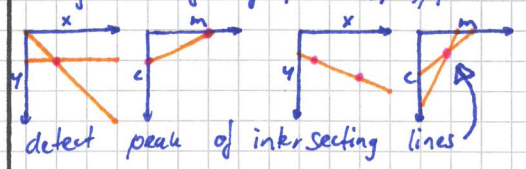
• **Prewitt:** $\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}\begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&1\end{bmatrix}$ • **Sobel:** $\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}$ • **Roberts:** $\begin{bmatrix}0&1\\-1&0\end{bmatrix}\begin{bmatrix}1&0\\0&-1\end{bmatrix}$

• **Laplacian:** detect edges using second derivative $\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$, $\begin{bmatrix}1&1&1\\1&-8&1\\1&1&1\end{bmatrix}$; is isotropic (rotationally invariant); sensitive to noise ⇒ need blurs; respons equally to strong and weak edges

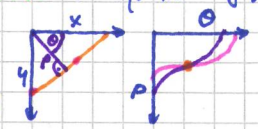
• **Laplacian of Gaussian (LoG):** comb. Lap. and G. $LoG(x,y) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$

• **Canny Edge Detector:** 1. Gaussian blur img. 2. Compute magnitude and angle 3. **Nonmaxima supression:** quantize edge normal to either: horizontal, vertical, +45°, −45°; select single maximum pt; maximum: gradient magnitudes of the two neighbours in edge normal direction are smaller 4. **Hysteresis:** • no edge: $G < T_{low}$; • Strong Edge: $G > T_{high}$; • Weak Edge: $T_{low} < G < T_{high}$ and it is connected to a strong edge via weak edges

• **Hough Transform:** Fit straight line through set of edge pixels (/pts); $y = mx + c$

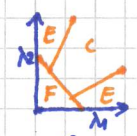


• **Infinite slope problem:** no equation for vertical line $\rho = x\cos\theta + y\sin\theta$



Corner Detection: Local displacement sensitivity: $S(\Delta x, \Delta y) = \sum_{(x,y)\in \text{wind.}} (f(x,y) - f(x+\Delta x, y+\Delta y))^2 \approx \sum_{(x,y)\in \text{wind.}} (f(x,y) - (f(x,y) + f_x(x,y)\cdot\Delta x + f_y(x,y)\,\Delta y))^2 = \sum_{(x,y)\in \text{wind.}} (f_x(x,y)\Delta x + f_y(x,y)\cdot\Delta y)^2 = \sum_{(x,y)\in \text{wind.}} \left([f_x(x,y), f_y(x,y)]\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right)^2 = [\Delta x, \Delta y]\left(\sum_{(x,y)\in \text{wind.}}\begin{bmatrix} f_x^2(x,y) & f_x(x,y) f_y(x,y) \\ f_x(x,y) f_y(x,y) & f_y^2(x,y)\end{bmatrix}\right)\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = [\Delta x, \Delta y]\, M \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}$

• **Harris Corners:** Find pts where $\min \Delta^T M \Delta$, for $\|\Delta\| = 1$ is large, i.e. maximize eignvalues of M;



• **Cornerness:** $C(x,y) = \det(M) - k(\text{trace}(M))^2 = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)$ • **Properties:** • invariant to brightness offset; • invariant to shift and rotation; • not invariant to scale

• **Lowe's SIFT:** • **Difference of Gaussian (DoG):** $DoG(x,y) = \frac{1}{k}\exp\left(-\frac{x^2+y^2}{(k\sigma)^2}\right) - \exp\left(-\frac{x^2+y^2}{\sigma^2}\right)$

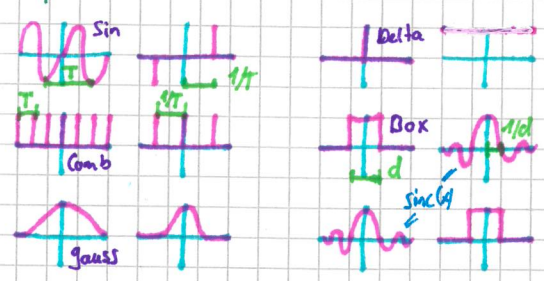
• **Pos. and Scale:** Look for strong response of DoG over scale space; only consider maxima in pos. and scale • **Orientation:** Compute gradients in 16x16 px neighbourhood; creat histogram ~~for 4x4 px blocks~~ (128 bins); dominant bin is ~~orientation~~ of key pt

• **Descriptor:** compute gradients in 16x16 px neighbourhood; create histogram for 4x4 px blocks (8 bins); 16 dominant directions with certain magnitude form desc.

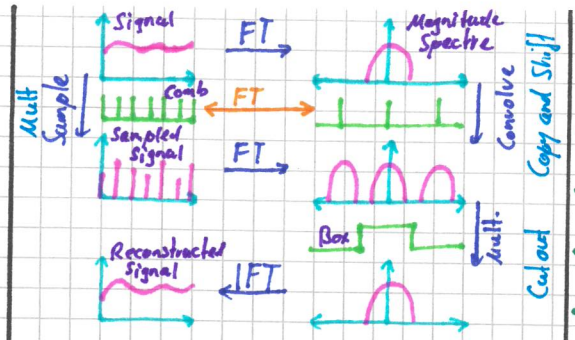
Fourier Transform (FT)

Represent function in new basis

- Basis: $B_{uv}(x,y) = \exp(-i2\pi(ux+vy)) = \cos(2\pi(ux+vy)) - i\sin(2\pi(ux+vy))$
- Frequency: magnitude of vector (u,v)
- Orientation: direction of vector (u,v)
- FT: $F(I)(u,v) = \iint_{\mathbb{R}^2} I(x,y)\, B_{uv}(x,y)\, dx\, dy$
- Inverse FT: $I(x,y) = \iint_{\mathbb{R}^2} F(I)(u,v) \exp(i2\pi(ux+vy))\, du\, dv$
- Discrete FT (DFT): $F(I) = U^T$ (FT Base)
- Magnitude Spectrum: composition of bases
- Phase Spectrum: offset of bases
- Phase is more important in natural imgs
- Important Functions:



- Convolution Theorem:
- $F(g * h) = F(g) \cdot F(h)$; filtering
- $F(g \cdot h) = F(g) * F(h)$; sampling
- Sampling: continuous → discrete
- $\text{Sample}_{2D}(f(x,y)) = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f(x,y)\, \delta(x-i, y-j)$
- $F(\text{Sample}_{2D}(f(x,y))) = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} F(u-i, v-j)$
- 2D FT as 1D FT: is fast



- Aliasing: intersection du to too low sampling frequency:

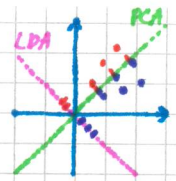
⇒ smooth with gauss (and not box)

- Nyquist Sample Theorem: Sampling frequency must be greaterequal to highest frequ.: $\omega_s \geq 2\omega$
- Image Restoration: $f(x) \to h(x) \to g(x) \to \tilde{h}(x) \to f(x)$
- $\tilde{h}(x)$ should compensate changes by $h(x)$
- determine $\tilde{h}(x)$ as: $F(\tilde{h})(u,v) \cdot F(h)(u,v) = 1$
- Problems: • h cancles some frequencies; • Noise gets amplified • Solution: use regularization: $F(\tilde{h})(u,v) = F(h) / (|F(h)|^2 + \varepsilon)$ to avoid singularities
- Applications: • Defocus Blur • Motion Blur: Dot of light is transformed into short line along some direction • Space-Time Super Resolution: Create/reconstruct intermediate frames • Spatial Super Resolution: Combine many low-res. imgs.
- Noisy-low-Res. Img.: • 1. remove noise by element-wise mult. with low or band pass • zoom and multiply with high-pass to sharpen

Unitary Transform

- Vectorisation: write img row-by-row as col. vector
- Linear Img Processing: can always be written: $g = Hf$
- Unitary is H iff: $H^{-1} = H^{*T} \equiv H^H$
- Energy Conservation: $\|g\|^2 = g^H g = f^H H^H H f = \|f\|^2$; ⇒ is a rotation of coord. system (vect. len. conserved)
- Image Collection: $F = (f_1, f_2, \dots f_n)$
- Auto Correlation ~~Matrix~~ Function: $R_{ff} = \frac{F \cdot F^H}{n} = I$ (positive, uncorrelated)
- Autocorrelation Matrix: $R_{cc} = H R_{ff} H^H$
- Eigenmatrix: Φ of R_{cc}: $R_{ff}\Phi = \Phi\Lambda$
- Karhunen-Loeve Transform (KLT, PCA):
- Project high-dimensional space to low-dim. subspace
- Unitary transf. with $H = \Phi^H$, where cols. of Φ are ordered according to decreasing eigenvalue
- $R_{cc} = H R_{ff} H^H = \Phi^H R_{ff} \Phi = \Phi^H \Phi \Lambda = \Lambda$
- Energy Concentration Property: • optimal concerning energy packed into first J coefficients • Mean square approx. error minimized for linear comb. of first J coefficients
- Eigenimage: Basis consisting of first J coeffs.
- Application: • Lossy Compression • Face detection
- Need to subtract mean to prevent high correlation of uncorrelated data: $g = H_f(f - \hat{f})$
- Problem: First K coeffs. represent a wrong energy since it only tries to max. energy conservation

- Fischer Face / Linear Discriminant Analysis (LDA):



- Idea: • Maximize: "Between-Individual" Variance • Minimize: "Within-individual" Variance
- JPG Compression: • 1. Discrete Cosin Transform (DCT): convert all 8x8 blocks into DC space; B_{00} represents average brightness • 2. Quantization: divide coeffs. with value of quantization matrix corresponding to that basis and round → reduces high frequencies • 3. List Coeffs: Zig-Zag Order • 4 Lossless compression

Pyramides and Wavelets

- Scale Space: Successive suppression of details
- Application: • Correspondence search • Edge tracking • Ballance detail vs comput. cost
- Gaussian Pyramid: Gaussian blur + subsample
- Laplacian Pyramid: DoG of lvls. of Gauss Pyramid; • Compar current with upsampled next
- Orientation Pyramid: gradient filter on all lvls.
- Wavelet: localized in space and frequency
- 1D-Discrete Wavelet Transform: Recursively halve high frequencies • Haar Transform: Simplest 2D Wavelet

Optical Flow

- Motion of pts between consecutive frames
- Application: Segmentation, tracking, slow motion, etc.
- Assumptions: • Small Motion: $\delta x, \delta y, \delta t$ are small
- Brightness Constancy: Brightness of pt does not change: $I(x(t), y(t), t) = C \overset{\text{small motion}}{\Rightarrow} I(x,y,t) = I(x+\delta x, y+\delta y, t+\delta t) \overset{\text{Taylor}}{\approx} I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t$

$\Rightarrow 0 \approx \frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t} = I_x u + I_y v + I_t$

$\Rightarrow$ 1 equ. 2 unknowns Aperture Problem: assume flow is along normal ($\Rightarrow$ Normal Flow)

- Horn & Schunck Algo.: • Smoothness Constraint: $e_s = \iint ((u_x^2 + u_y^2) + (v_x^2 + v_y^2))\,dx\,dy$: assume neighbours move similarly • OF constraint: $e_c = \iint (I_x u + I_y v + I_t)^2\,dx\,dy$ • minimize: $e_s + e_c$
- Lukas-Kanade: Assume same OF for all pts in patch (Spacial Coherence): $E(u,v) = \sum_{x,y \in \Omega} (I_x(x,y)u + I_y(x,y)v + I_t)^2$

$\frac{dE}{du} = \frac{dE}{dv} = 0$

$\Rightarrow$ gives LSE: $\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix}}_{A \text{ (for neighbourhood)}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_n) \end{bmatrix}}_{b}$

get OF

- Solve by minimizing $(Ax-b)^2$ (least-square) • solution given by $(A^TA)x = A^Tb$ • alternative writing: $\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{M = \sum \nabla I \nabla I^T} \begin{bmatrix} u \\ v \end{bmatrix} = -\underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{-\sum \nabla I I_t = b}$ • Algorithm:
- at each pixel compute u by $Mu = b$ • M is singular if all gradient vectors point in same dir.: • along edge • only over single pixel • Aperture Prob.
- Iterative Refinement: • 1: estimate OF • 2: warping along flow • 3: repeat from 1 • not working for to complex functions
- Coarse to Fine: when step larger than 1 pixel • 1: create image pyramid • 2: calculate OF on lvl where step is 1 pixel • 3: warp next lvl according to OF • 4: recalculate OF • 5: repeat 3
- Parametric Motion Model: describe global motion using one affine motion: $I(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \approx 0$
- SSD: • For large displacements • Match template against each pixel in small neighbourhood, calculate SSD and take max
- Bayesian OF: Some low-level motion can be explained by adding uncertainty to LK • Applications: • Video Compression • Destaker • Slowmo • VR

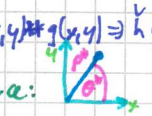
Video Compression

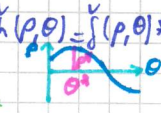
- Interlaced Video Format: • 2 temporally shifted half imgs • Increases frequency • decrease spatial resolution
- Lossy Compression: • Remove redundancy:
- Spacial correlation between neighbouring pixels
- Temporal correlation between frames
- Frame Types: • I-Frame: fully indep. of other frames • P-Frame: encoded based on prev. frame • B-Frame: encoded based on prev. and future frame
- Temporal Redundancy Reduction: • send encoded frame + motion vector; • inefficient on scene change and fast motion
- Motion-Compensated Prediction (MC): Predict motion in consecutive frames • Block-Matching Motion Estimation Algo: 1. divide frame into non-overlapping blocks 2. for each block, find best-matching block in reference frame according to some metrics; search in full frame or some subset • Motion Vector: gives relative offset • Motion Vector Field: Motion Vector for all blocks • Half-Pixel Motion Estimation algo: • Frames are spatially interpolated + can capture half-pixel motion
  + Improves prediction due to averaging
  + Improved compression as noise is reduced
- Algo: • 1.: Do integer motion estimation • 2: • compute interpolant for half-pixel offset for best match; • take half-pixel offset block or integer offset block depending on which is better
- Pros: + good, robust compression + simple
- Cons: - only simple motion - blocking artefacts
- Bidirectional MC: motion estimation for prev. and future frame; i.e. B-frames

Radon Transform

- X-ray moves along straight line: - at distance s has intensity $I(s)$ • travels distance δs reduces intensity by δI, which depends on optical density $u(s)$ of material
- for small δs: $\frac{\delta I}{\delta s}(s) = -u(s)\,\delta s$
- Travel along line L: $I_{final} = I_{start}\, e^{-R}$, $R = \int_L u(s)\,ds$
- RT of $f(x,y)$: $R_f(L) = \int_L f(x)\,|dx|$, $R(\rho,\theta) = \int u(\rho\cos(\theta) - \underbrace{s\sin(\theta)}_{\text{arc len}}, \rho\sin(\theta) + s\cos(\theta))\,ds = \iint_{-\infty}^{\infty} u(x,y)\,\underbrace{\delta(\rho - x\cos(\theta) - y\sin(\theta))}_{\text{Delta func}}\,dx\,dy$

- Properties: • Linearity: $g(x,y) = \sum_q \alpha_q g_q(x,y) \Rightarrow \check{g}(\rho,\theta) = \sum_q \alpha_q \check{g}_q(\rho,\theta)$ • Shifting: $h(x,y) = g(x-x_0, y-y_0) \Rightarrow \check{h}(\rho,\theta) = \check{g}(\rho - x_0\cos(\theta) - y_0\sin(\theta), \theta)$ • Rotation: $h(r,\phi) = g(r, \phi - \phi_0) \Rightarrow \check{h}(\rho,\theta) = \check{g}(\rho, \theta - \phi_0)$ • Convolution: $h(x,y) = f(x,y) * g(x,y) \Rightarrow \check{h}(\rho,\theta) = \check{f}(\rho,\theta) * \check{g}(\rho,\theta)$
- Point Source:

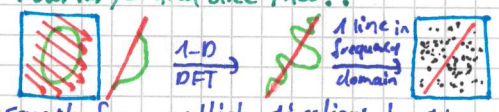




$\Rightarrow$ Sinogram

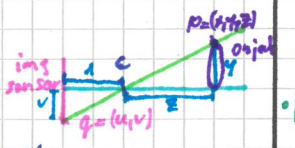
- Backprojection: find $u(x,y)$ given $R(\rho,\theta)$

- Fourier/Central Slice Theo.:



- sample from multiple directions to get more complete frequency domain • Inverse DFT to get original image - sensitive to noise - unstable - hard to implement $\Rightarrow$ blurying
  improve by applying high-pass in frequ. domain

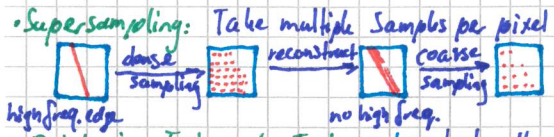
Introduction



- Perspective Projection:
- Vertical Coordinates: $y/1 = y/z$ • Horizontal Coord: $u = \frac{x}{z}$
- Rasterization: • Convert continuous obj. to a discrete representation on pixel grid • Rules: • all intersecting pixels • Diamond: pixels where line intersects ◆
- Find pixels: • check every px: $O(n^2)$ • Incremental: color pixel per x-coord which matches best

Triangles Draw

- Coverage: What pixel does the triangle overlap?
- Occlusion: Which triangle is closest for each pixel?
- Sample pixel and color according to triangle coverage ⇒ artefacts
- Supersampling: Take multiple samples per pixel

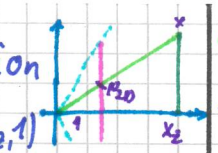


- Point-in-Triangle Test: • naive: test all pts if "inside" all 3 edges • incremental: zigzag pts starting at anchor • tiled: as incremental but consider whole block in parallel
- Sampling: • Delta: $\delta(x) = \begin{cases} 0 & x \neq 0 \\ \text{undef} & \text{else} \end{cases}$ s.t. $\int_{-\infty}^{\infty} \delta(x)\,dx = 1$
- Sample at a: $f(a) = \int_{-\infty}^{\infty} f(x)\,\delta(x-a)\,dx$ • Comb: $Ш_T(x) = T \sum_{i=-\infty}^{\infty} \delta(x - iT)$ • Sample: $Ш_T(x) f(x) = T \sum_{i=-\infty}^{\infty} f(iT)\,\delta(x - iT)$ • Reconstruct: $(\underbrace{f}_{\text{filter}} * \underbrace{g}_{\text{sampled sig.}})(x) = \int_{-\infty}^{\infty} f(y)\,g(x-y)\,dx$ (recons. signal)

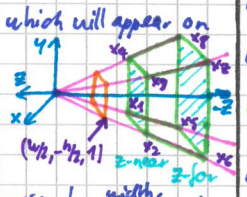
Linear Transforms

- Benefits: • computationally cheap • all maps can be approx. over short distance • comp. is linear (rotation)
- Linear Map: • "Maps lines to lines and keeps origin fixed" • $f(u+v) = f(u) + f(v) \wedge f(au) = af(u)$
- Linear Combination: $f(u) = \sum_{i=1}^{m} u_i a_i$, a_i is fixed
- $\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = Ax$
- Homogeneous Coordinates: • lift 2D/3D pts to 3D/4D space (3D-H/4D-H) • makes affine transforms linear
- Convert to: • add dimension and set $1 \cdot \begin{bmatrix} a \\ b \end{bmatrix} \Rightarrow \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$
- Convert from: • normalize by last dimension and remove it • $\begin{bmatrix} a \\ b \\ w \end{bmatrix} \rightarrow \begin{bmatrix} a/w \\ b/w \end{bmatrix}$
- Scale invariant: • $\alpha a = a, \forall \alpha \neq 0$
- Point: last dimension $\neq 0$: $\begin{bmatrix} a \\ b \\ w \end{bmatrix}, w \neq 0$
- Vector: last dimension $= 0$: $\begin{bmatrix} a \\ b \\ 0 \end{bmatrix}$
- Scale: • 3D: $\begin{bmatrix} s_x & & \\ & s_y & \\ & & s_z \end{bmatrix}$ • 3D-H: $\begin{bmatrix} s_x & & & \\ & s_y & & \\ & & s_z & \\ & & & 1 \end{bmatrix}$
- Rotation: • x: $\begin{bmatrix} 1 & & \\ & \cos\alpha & -\sin\alpha \\ & \sin\alpha & \cos\alpha \end{bmatrix}$ • y: $\begin{bmatrix} \cos\alpha & & \sin\alpha \\ & 1 & \\ -\sin\alpha & & \cos\alpha \end{bmatrix}$
- z: $\begin{bmatrix} \cos\alpha & -\sin\alpha & \\ \sin\alpha & \cos\alpha & \\ & & 1 \end{bmatrix}$ • 2DH/3DH: $\begin{bmatrix} R & \\ & 1 \end{bmatrix}$ • arbit. axis: $\begin{bmatrix} \cos\alpha + u_x^2 z & u_x u_y z - u_z \sin\alpha & u_x u_z z + u_y \sin\alpha \\ u_y u_x z + u_z \sin\alpha & \cos\alpha + u_y^2 z & u_y u_z z - u_x \sin\alpha \\ u_z u_x z - u_y \sin\alpha & u_z u_y z + u_x \sin\alpha & \cos\alpha + u_z^2 z \end{bmatrix}$, $z = (1 - \cos\alpha)$
- Shear: • 3D: $\begin{bmatrix} 1 & d_y & d_z \\ & 1 & \\ & & 1 \end{bmatrix}$ • 3D-H: $\begin{bmatrix} 1 & d_y & d_z & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$ in x direction, based on y, z
- Translate: 3D-H: $\begin{bmatrix} 1 & & & b_x \\ & 1 & & b_y \\ & & 1 & b_z \\ & & & 1 \end{bmatrix}$

Perspective Projection



- Input: pt in 3D-H: $x = (x_x, x_y, x_z, 1)$
- Output: pt on screen in 2D: $p_{2D} = (x_x/x_z, x_y/x_z)$
- Steps: • 1.: multiply x by $P = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & 1 & 1 \end{bmatrix}$ to get projected pt: $P_x = (x_x, x_y, x_z, x_z)$ • 2.: convert to 2D-H: $p_{2D-H} = (x_x, x_y, x_z)$ • 3.: conv. to 2D: $p_{2D} = (x_x/x_z, x_y/x_z)$
- View Frustum: region in 3D which will appear on screen • Transform to Unitcube:



$$P = \begin{bmatrix} f/\text{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zf + zn}{zn - zf} & \frac{2x\,zf \times zn}{zn - zf} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

• aspect $= \text{width}/\text{height}$,
• θ = field of view in y dir, • $f = \cot(\theta/2)$

Geometry

- Implicit: Points are not known but satisfy some relationship • e.g. $x^2 + y^2 + z^2 = 1$
- Algebraic Surface: surface is zero of polynomial
- Constructive Solid Geometry: construct using Bool ops.
- Blobby Surface: gradually blend surfaces together
- Lvl Set Method: • grid approximates potentially complex function; • surface is at interpolated zeros
- Fractals/L-Systems: Structure that exhibits self-similarity • Components: • Grammar Rules: give symbol transitions • Start Symbol (/set)
- Semantics: way of interpreting strings
- Steps: • Select rule and apply simultaneously to all possible positions; • repeat
- Branching Structure: • "[" push state onto stack; • "]" pop state from stack (⇒ return to last stored position + rotation) • Monopodial: • Trunk extends undeviated to top • Branch extend perpendicularly: Y • Sympodial: • Trunk deviates to top; • Branches extend perpendicularly: Y • Binary: • Trunk terminates at branching pt • Branches extend uniformly: Y
- Stochastic L-System: Multiple rules which apply to the same symbol with associated prob.
- Pro/Con: + inside/outside test is easy + other queries may also be easy + compact form + easy to handle changes in topology – expensive to find all pts on shape – difficult to model complex surfaces
- Explicit: Pts are given directly
- Point Cloud: • set of pts; • sometimes augmented with normal + easy to represent any shape – hard to simulate
- Triangle/Polygonal Mesh: • Store: • geometry: vertex location • Topology: how vertecies are connected (• Other Attributes) • Requires special DS + easy to process/simulate + can model complex shapes • most commonly used method in graphics
- Subdivision Surfaces: • Smooth out control curve by inserting new pts at each edge mid pt and updating vertex pos. according to fixed rule – continuity at vertices? – interpolate or approx.?

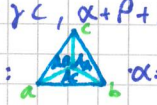
• Pro/Con: + easy to sample – hard for inside/outside + complex shapes
• Sources of Geometry: • 3D scan • 3D Modeling

Surface Shade and Color

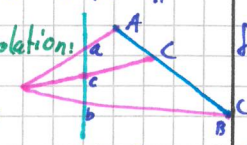
• Irradiance: $E = \frac{\Phi}{A}$ • Energy per time, per area
• Lambert's Law: • $E = \frac{\Phi}{A'} = \frac{\Phi\cos(\theta)}{A}$, • $A = A'\cos(\theta)$

• N-Dot Lighting: • Abstraction: Infinite bright light-source at infinity ⇒ light directions (L) are identical
• Shade: take dot P of unit surface normal (N) and unit direction to light (L)
• Color Interpolation via Barycentric Coords.:
• given: • triangle (a, b, c) with distinct colors at corners
• pt x on triangle:

• goal: color of x?
• Steps: • b-a and c-a form non-orthogonal basis for pts in triangle; • $x = a + \beta(b-a) + \gamma(c-a) = \alpha a + \beta b + \gamma c$, $\alpha + \beta + \gamma = 1$ • Area ratio:
• alternative:



$\alpha = \frac{A_A}{A}$, $\beta = \frac{A_B}{A}$, $\gamma = \frac{A_C}{A}$
• Perspective-Incorrect Interpolation:



• c is half-way between a, b, but C is not half-way between A, B
• Perspective-Correct Interpolation: • 1: Compute depth z at each vertex; • 2: evaluate $Z := 1/z$, $P := f/z$ at each vertex; • 3: Interpolate Z and P using standard (2D) barycentric coords; • 4: at each fragment, divide interpol. P by interpol Z to get Attribute values • Works for any attribute val. that varies lin. accross triang.

Texture

• Put 2D image on 3D surface
• Texture Coordinates: define a mapping from surface coords. (pts on triangle) to pts in text. domain
• Normal Mapping: texture gives flat surface appearance of having more structure
• Filtering Textures: Texture is often not given in the right size/resolution ⇒ aliasing
• Minification: • Area of screen pixel maps to large region of texture • E.g. object far away • blurs text.
• Magnification: area of screen pixel maps to tiny region of texture • E.g. object is very close • Texture map is too small ⇒ interpolate
• Mipmap: • Gaussian Pyramid of Texture • 1/3 Storage overhead • Compute level d:
$d = \log_2 \max\left\{\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right\}$
for two neighbour pts of pt in screen coords, converted to texture coords.

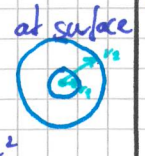
• Trilinear Filtering: provides continuous d by merging texture of d and d+1

Rasterization Pipeline

• Depth Buffer/Z-Buffer: stores depth to closest triangle for each sample pt
• white: far; • black: close • Occlusion:
• 1: set ZBuffer to white • 2: color ZBuffer if triangle is close than all previously added triangles • 3: repeat for all triangles to be added
+ can maintain a color buffer simultaneously + works for interpenetrating surfaces + works for super sampling + constant Z-Buffer access time + works for any shapes – range of depth value is limited ⇒ need near/far plane
• Compositing:
• Opacity α: describes opacity of object:
• Fully opaque: $\alpha = 1$ • Fully transparent: $\alpha = 0$
• "Over" operator: Compositing img $B = (B_r, B_g, B_b)^T$ with α_B over img $A = (A_r, A_g, A_b)^T$ with α_A
• Composite Color: $C = \alpha_B B + (1-\alpha_B)\alpha_A A$
• Composite Opacity: $\alpha_C = \alpha_B + (1-\alpha_B)\alpha_A$
• Premultiplied alpha: $A' = \alpha_A (A_r, A_g, A_b, 1)^T$
$B' = \alpha_B (B_r, B_g, B_b, 1)^T \Rightarrow C' = B' + (1-\alpha_B) A'$
• Rendering Mixture of Opaque & Transparent:
• 1: • render opaque surfaces using ZBuffer occlusion
• 2: • disable Z-Buffer updates • render transparent surfaces in back-to-front order • depth test passes ⇒ triangle is composed over content of color buffer
• End-to-End Pipeline: • 1: transform triangle vertices into camera space • 2: transform view-frustrum to unit cube • 3: remove triangles outside of the cube • clip triangles that extend beyon the cube • 4: transform vertex x, y pos. from normalized coordinates into screen coords. (based on w, h)
• 5: Precompute triangle edge equations and triangle attribute equations • 6: Sample coverage, evaluate attributes Z, u, v at all covered samples • 7: compute triangle color at sample pt (color interpolation, texture map, etc.)
• 8: Perform depth test (if enabled) and update depth value at covered samples (if necessary)
• 9: update color buffer (if depth test passes)
• Shadow Mapping: • 1: generate Z-Buffer from location of light • 2: everything directly seen from there is directly lit • 3: generate Z-Buffer + Color Map from location of camera
• 4: transform every screen sample to light coord. frame and perform depth test: fail ⇒ in shadow
• Reflections: • 1: Place camera origin at location of reflective object • 2: render six views • 3: use real camera ray reflected about surface normal to determine which texel in cube map is hit

Rendering Equation

• Color: • Emission Spectrum: Intensity of a given wavelength emitted • Absorption Spectrum: fraction of absorption of given wavelength • color is what gets reflected • Color Model: • Additive: combining light • Subtractive: combining paint color • Color Space: • HSV: • hue, Saturation, value • Y'CbCr: • Y': intensity • Cb: blue-yellow deviation from gray • Cr: red-cyan deviation from gray • RGB • Model of Light: • Radiant Energy: Total number of hits • Radiant Flux: hits per second • Irradiance: hits per second per unit area • Radiant Energy Density: hits per unit area • Energy carried by a photon: $Q = \frac{hc}{\lambda}$ • Flux: • Energy per unit time (watts) received by the sensor / emmitted by a light source • $\Phi = \lim_{\Delta t \to 0} \frac{\Delta Q}{\Delta t} = \frac{dQ}{dt}$ [J/s] • $Q = \int_{t_0}^{t_1} \Phi(t)\,dt$ • Isotropic Point Source: photons are emitted from some pt: • Irradiance falloff with distance: • assume: light is emitting flux Φ in a uniform angular distribution • Irradiance at surface of two spheres $\{r_1, r_2\}$



• $E_1 = \frac{\Phi}{4\pi r_1^2} \rightarrow \Phi = 4\pi r_1^2 E_1$
• $E_2 = \frac{\Phi}{4\pi r_2^2} \rightarrow \Phi = 4\pi r_2^2 E_2$
$\Big\}\ \frac{E_2}{E_1} = \frac{r_1^2}{r_2^2}$

• Radiance: Energy along a ray defined by origin pt p and direction ω:

$$\underbrace{E}_{\text{irradiance}} = \int_{\underbrace{H^2}_{\text{hemisphere}}} \underbrace{L(\omega)}_{\text{Radiant in dir. } \omega} \underbrace{\cos(\theta)}_{\text{angle between } \omega \text{ and normal}} \underbrace{d\omega}_{\text{Lambert's Law}}$$

• Incident Radiant: • L_i • Exitant Radiant: L_o • $L_i(p,\omega) \neq L_o(p,\omega)$

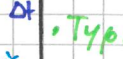
• Rendering Equation:

$$\underbrace{L_o(p,\omega_o)}_{\text{point of interest, direction of interest}} = L_e(p,\omega_o) + \int_{H^2} \underbrace{f_r(p,\omega_i \to \omega_o)}_{\text{scattering function}} L_i(p,\omega_i) \cos(\theta_i)\, d\omega_i$$

• Equation is recursive

• Scattering Function: • Reflection determines surface appearance • Characteristics: • bounce off surface • transmit through surface • bounce around inside surface (exit pt ≠ entry pt) • absorbe and re-emit

• Bidirectional Reflect Distrib Func. (BRDF): • light bounces off surface • $f_r(\omega_i \to \omega_o)$ • how much light of dir. ω_i gets scattered to ω_o • Properties: • $f_r \geq 0$ • $\int_{H^2} f_r \cos(\theta)\, d\omega_i \leq 1$ • $f_r(\omega_i \to \omega_o) = f_r(\omega_o \to \omega_i)$

• Types: • Ideal Specular:  • Ideal Diffuse: • Glossy Specular: • Retro-Reflective:

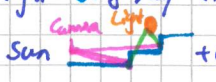
• BSSRDF: • exit pt ≠ entry pt • required for many surfaces

Raytracing

• Rasterization and ray casting both solve the "visibility" determing problem

• Rasterization: • Proceeds in triangle order • Most processing is based on 2D primitives • Stores depth buffer

• Raycast: • Proceeds in screen sample order • No depth buffer needed • Transparent surfaces are naturally processed front-to-back • Stores entire screen • coverage: does ray hit triangle? • occlusion: closest hit

• Shadow Ray Tracing: • 1: shoot ray from camera to object • 2: shoot ray from obj. intersection to light • 3: if ray hits something ⇒ shadow, else sun 

+ correct hard shaddows + no aliasing (unlike rasterization)

• Reflections: recursively reflect rays of objects

• Ray Parameterization Equation: $\underbrace{r(t)}_{\text{pt along ray}} = \underbrace{o}_{\text{origin}} + t\,\underbrace{d}_{\text{unit dir.}}$

• Ray-Implicit Surface Intersection: • 1: insert ray $r(t)$ into surface $f(x) \Rightarrow f(r(t))$ and solve for t • 2: insert t_i back into $r(t) \Rightarrow p(t_i)$

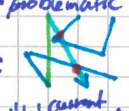
• Ray-Triangle Intersection: • 1: Convert triangle (a,b,c) to implicit surface by parametrization using barycentric coordinates: $f(u,v) = (1-u-v)a + ub + vc$

• 2: ~~insert~~ $f(u,v) = r(t)$ and solve for u,v,t: $[b-a,\ c-a,\ -d]\begin{bmatrix} u \\ v \\ t \end{bmatrix} = o - a$ • 3: all u,v,t are positive ⇒ hit

• Ray-Scene (N primitives) Intersection: • goal: find first primitive which the ray hits • Naive: check all primitives for hit and take closest • $O(N)$ • axis-aligned-box: • split scene into V (primitive disjoint) boxes • only check all primitives of box if box is hit by ray • calculate x and y plane (of box) intersection • min gives first intersection

• Bilding Volume Hierarchy (BVH): • recursively split boxes and represent as tree • Need to check all?

• Space Partitioning: • Partition space into disjoint regions • Uniform grid: • grid has uniform size • cell contains all primitives that it overlaps • Walk ray through volume in order • Heuristics: # voxels ~ N ⇒ intersection cost: $O(\sqrt[3]{N})$

• KD-Tree: recursively partition space using axis-aligned planes • Can terminate after first "hit" (Node) found • Primitive overlap multiple cells: • problematic as "first hit" may not be the closest one:  • Solution: hit point must be within current node • Quad-Tree/Oct-Tree: node has 4/8 or no children • can adapt to more dense locations

Animation

- Artist directed: keyframing
- Data-driven: motion capture
- Procedural: simulation

- **Keyframing:** • Keyframe: • Specify certain event (position, color, lighting, camera config, etc.) at certain times • In-Betweens: • Interpolated/approximated frames between keyframes
- **Interpolation:** • nearest neighbour • linear
- **Spline:** piecewise polynomial • use cubic poly.: $a(t) = at^3 + bt^2 + ct + d$ • 4n DOF • Properties:
- **Interpolation:** pass exactly through data pts
- **Continuity:** at least twice differentiable
- **Locality:** moving one pt does not affect unbounded
- Can only get 2 properties at the same time
- **Types:** • Natural: • Interpolation (2n eq) + continuity (2(n-1) eq)
- set curvature to zero (2 eq) at endpoints
- **Hermit/Bezier:** • Interpolation (2n eq) + locality
- set tangent to some constant (2n eq) • Catmull-Rom: set tangent equal to slope when connecting neighbours: • B-Spline: • locality + continuity • recursively defined
- **Rigging:** user-defined mapping between small number of parameters and deformation on mesh
- **Rigid Transform:** only rotation and translation
- **Blend Shape:** • linear combination of weighted "poses" – only works for facial expressions – cannot change topology + simple
- **Cage-Based:** embed model in coarse mesh
- model object vertecies as linear combination of cage vertecies

- **Skeleton:** • animate skeleton and let mesh follow automatically • Full tissue simulation: + best animation – computationally expensive
- **Linear Blend Skinning:** • move skin along with bone • Attempt 1: • assign each vertex to closest bone • calculate vertex pos. using forward kinematics: $v = {}_w R_j \, {}_w \bar{R}_j^{-1} v'$ + vertices close to bone + small movements • Attempt 2: • assign each vertex to multiple bones • compute vertex pos. as convex combination: $v = \sum_j \alpha_{jw} \, {}_R j_w \, \bar{R}_j^{-1} v'$ + smoother skin
- **Animation:** • Approach 1: ~~manually by moving~~ Keyframing rigs • Approach 2: Motion capture

Physically-Based Animation

- Some things are to complex to animate using classical methods
- **Animation Equation:** $F = m \cdot a$
- **generalized Coordinates:** $q(t)$ • **Velocity:** $\dot{q}(t) = \frac{d}{dt} q(t)$ • **acceleration:** $\ddot{q}(t) = \frac{d}{dt} \dot{q}(t)$
- **Ordinary Differential Equation (ODE):** $\frac{d}{dt} q = f(q, \dot{q}, t)$ • **Velocity Function:** f
- **Particle System:** • Models complex phenomena
- Each particle has behaviour desc. by forces, e.g. attraction, repulsion, alignment,

- **Numerical Solving of ODE:** • given: initial conditions $q(0)$, $\dot{q}(0)$ • find: $q(t)$
- **Numerical Integration:** discrete approximation q_i at t_i for time-continuous $q(t)$ • approx. continuous $q(t+h) = q(t) + \int_t^{t+h} \dot{q}(t)\,dt$
- **Rectangle Rule** • **Midpoint Rule** • **Trapezoid Rule**

- **Update:** $q_{i+1} = q_i + \Delta q_i$
- **Forward Euler:** New state explicitly written of known data + simple – unstable – need very small timestep
- **Pos. Update:** $q_{i+1} = q_i + h \cdot \dot{q}_i$
- **Veloc. Update:** $\dot{q}_{i+1} = \dot{q}_i + h\ddot{q}_i = q_i + hM^{-1}F(q_i, \dot{q}_i)$
- **Backward Euler:** New state implicite and must solve for it + stable for lin. ODEs
- **Pos. Update:** $q_{i+1} = q_i + h \cdot \dot{q}_{i+1}$
- **Veloc. Update:** $\dot{q}_{i+1} = \dot{q}_i + h \cdot \ddot{q}_{i+1} = \dot{q}_i + hM^{-1}F(q_{i+1}, \dot{q}_{i+1})$
- F is generally non-lin. ⇒ solve non-lin equ.
- **Partial Differential Equations (PDE):** describes function through its partial derivatives with respect to multiple variables • $u_{xy} = \frac{\partial^2}{\partial x \partial y} u(x, y, \ldots)$
- **Nabla Op.:** $\nabla = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_d}\right)^T$, $\nabla s = \left(\frac{\partial s}{\partial x_1}, \ldots, \frac{\partial s}{\partial x_d}\right)^T$
- **Laplace Op.:** $\Delta = \nabla^T \cdot \nabla = \nabla_x^2 = \sum_{k=1}^{d} \frac{\partial^2}{\partial x_d^2}$
- **Order of PDE:** order of highest partial derivative
- **Linear PDE:** if the unknown function u and its partial derivatives only occure linearly

- **2nd order PDE in 2 vars.:** $F(x, y, u, u_x, u_y) = Au_{xx} + 2Bu_{xy} + Cu_{yy}$ • **Hyperbolic:** $B^2 - AC > 0$ • **Parabolic:** $B^2 - AC = 0$ • **Elliptic:** $B^2 - AC < 0$
- **Types:** • **Laplace Equ. (Elliptic):** What is the smoothest func. interpolating given boundary data • $\Delta u = 0$ • **Heat Equ. (Parabolic):** How does an initial distribution of heat spread • $\dot{u} = \Delta u$
- **Wave Equ. (Hyperbolic):** How do waves of a rock in a pond evolve • $\ddot{u} = \Delta u$
- **Space Discritisation:** • Two main ways
- **Lagrangian:** Track moving particles and read what they measure + conceptually easy + not limited to grid – good particle distribution is hard – find neighbour is expensive
- **Eulerian:** Read at fixed locations in space + easy computation + easy to represent – limited to grid