Cheat Sheet

Visual Computing

Silvan Metzker, Jacques Hoffmann

February 2025

1 Computer Vision

1.1 Digital images & sensors

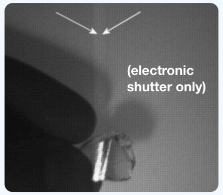
Charge coupled device (CCD)

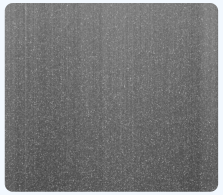
An array of photosites (a bucket of electrical charge) hold charge proportional to incident light intensity during exposure. Charges move down through sensor array, ADC (analog-digital-converter) measures line-by-line.

Blooming: Oversaturated photosites cause vertical channels to "flood"

Bleed-/Smearing: Bright light hits photosites while charge is in transit, worse with shorter exp. time

Dark current: CCDs produce thermally generated charge: random noise despite darkness.





CMOS sensors

Mostly same as CCD, but each sensor has amplifier. Cheaper, lower power, no blooming, on-chip integration with other components, but less sensitive and more noise. Rolling shutter is an issue for sequential read-out of lines.

Nyquist-Shannon sampling theorem

Sample frequency must be at least twice as big as the highest frequency in the signal/image to avoid aliasing.

Quantization: real-valued function get digital values.

Geometric resolution: #pixels per area
Radiometric resolution: #bits per pixel

Additive Gaussian Noise
$I(x,y) = f(x,y) + c$
where $c \sim N(0, \sigma^2)$
s.t.: $\mathrm{pdf}(c) = \left(2\pi\sigma^2\right)^{-1} e^{-\frac{c^2}{2}\sigma^2}$

Poisson (shot) N.
$\mathrm{pdf}(k) = \frac{\lambda^k e^{-\lambda}}{k!}$

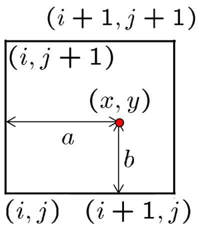
Rician noise (MRI)
$\mathrm{pdf}(I) = \frac{1}{\sigma^2} \exp\left(\frac{-(I^2+f^2)}{2\sigma^2}\right) I_0\left(\frac{If}{\sigma^2}\right)$

Multiplicative N.
$I = f + fc$

Signal to noise ratio (SNR)
$s = \frac{F}{\sigma}$ where
$F = \frac{1}{XY} \sum_{\{x=1\}}^{X} \sum_{\{y=1\}}^{Y} f(x,y)$

PSNR (P: Peak)
$s_{\text{peak}} = \frac{F_{\max}}{\sigma}$

Color camera concepts: Prism (splitting light, 3 sensors, requires good alignment), Filter mosaic (coat filter on sensor, grid, introduces aliasing), Filter wheel (rotate filter in front of lens, static image), CMOS sensor (absorbing colors at different depths)



Bilinear Interpolation: Reconst. from 2D sampling, ass. linear behav.:

$$\begin{aligned} f(x,y) &= (1-a)(1-b) \cdot f(i,j) \\ &+a(1-b) \cdot f(i+1,j) \\ &+ab \cdot f(i+1,j+1) \\ &+(1-a)b \cdot f(i,j+1) \end{aligned}$$

1.2 Image segmentation

Partition image into regions of interest.

Complete segmentation of I: regions $R_1, ..., R_N$ s.t. $I = \bigcup_{i=1}^{N} R_i$ and $R_i \cap R_j = \emptyset$ for $i \neq j$.

Thresholding

produces binary image by labeling pixels *in* or *out* by comparing to some threshold T.

$B(x,y) = 1$ if $I(x,y) \geq T$ else 0, finding T with trial and error, compare results with ground truth.

Chromakeying choose special background color $\boldsymbol{x}$, then segmentation using: $I_\alpha = \|\boldsymbol{I} - \boldsymbol{x}\| > T$. Issues: hard α-mask, variation not same in 3 channels.

Receiver operating characteristic (ROC)

ROC curve describes performance of binary classifier. Plots (y, sensitivity, $\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP + FN}}$) against (x, 1-specificity, $\frac{\mathrm{FP}}{\mathrm{FP + TN}}$). We can choose operating point with gradient $\beta = \frac{N}{P} \cdot \frac{V_{\mathrm{TN}} + C_{\mathrm{FP}}}{V_{\mathrm{TP}} + C_{\mathrm{FN}}}$ where V value and C cost.

Pixel neighbourhoods or 4/8-connectivity: 4-neighb. means horiz. + vert. or 8-neighb. with diag.

Region growing

Start from seed point / region, include pixels if some criteria is satisfied (e.g. pixel-pixel threshold or with std-deviation σ, mean μ of graylevels in region: $(I(x,y) - \mu)^2 < (n\sigma)^2$), iterate until no pixels added.

Seed region(s) by hand or conservative thresholding

Snakes

Active contour, polygon, where each point on contour moves away from seed while inclusion criteria in neighborhood is met, often smoothness constraint. Iteratively minimize energy: $E = E_{\text{tension}} + E_{\text{stiffness}} + E_{\text{image}}$

Mahalanobis Distance $\boldsymbol{I}_\alpha$

With N background samples c_{ref}^i and pixel $\boldsymbol{I}(x,y)$:
$\boldsymbol{I}_\alpha = \sqrt{(\boldsymbol{I}(x,y)-\mu)^\top \Sigma^{-1} (\boldsymbol{I}(x,y)-\mu)}$ with $n \geq 3$ we get:

$$\Sigma = \frac{1}{N-1} \sum_{i=1}^{N} \left(c_{\text{ref}}^i - \mu\right)\left(c_{\text{ref}}^i - \mu\right)^T \text{ with } \mu = \frac{1}{N} \sum x_i$$

We then include pixel $\boldsymbol{I}(x,y)$ if $\boldsymbol{I}_\alpha < T$.

Markov random fields: Cost to assign label to each pixel (ψ_1), cost to assign pair of labels to connected pixels (ψ_2). Solve with graph cuts, source = FG, sink = BG. Minimize energy in polynomial time using MinCut. To get decent results, we need **alpha estimation** / border matting along edges. With data d we get energy:

$$E(y; \theta, \mathrm{d}) = \sum_i \psi_1(y_i; \theta, \mathrm{d}) + \sum_{i,j \in \text{ edges}} \psi_2(y_i, y_j; \theta, \mathrm{d})$$

1.3 Image filtering

Shift-invariant: same for all pixels, e.g. indep. on x, y
Linear: $L(\alpha I_1 + \beta I_2) = \alpha L(I_1) + \beta L(I_2)$
Separable: kernel $K(m,n) = f(m)g(n)$

Local: Does it only depend on local pixels, e.g. not all.
Filtering near edges: clip to black, wrap around, copy edge, reflect across edge, vary filter!
Corr./conv the same if: $K(x,y) = K(-x,-y)$

Correlation

Template matching: $I' = K \circ I$

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

Convolution

Point spread function: $I' = K * I$

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

Linear, associative, shift-invariant, commutative (if dimensions are identical). For the continuous case:

$$g(x) = f(x) * k(x) = \int_{\mathbb{R}} f(a)k(x-a)\,\mathrm{d}a$$

Important kernel examples

Low-pass Mean	$\frac{1}{9}\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$	High-pass	$\begin{pmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{pmatrix}$
Laplacian	$\begin{pmatrix}0&1&0\\1&-4&1\\0&1&0\end{pmatrix}$	Prewitt (x)	$\begin{pmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{pmatrix}$
Gaussian G_σ	$\dfrac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$	Sobel (x)	$\begin{pmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{pmatrix}$
Diff. (x)	$\begin{pmatrix}-1&1\end{pmatrix}$	Central diff.	(x) $\begin{pmatrix}-1&0&1\end{pmatrix}$

The Gaussian kernel is rot. symetric, single lobe, FT is again a Gaussian, separable, easily made efficient.

Note: High-pass usually sums up to 0, low-pass to 1.

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon\to 0}\frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

Hence, diff. leads to the convolution $\begin{pmatrix}-1&1\end{pmatrix}$

Image sharpening: enhances edges by increasing high frequency components: $I' = I + \alpha|k * I|$ where k high-pass filter, $\alpha \in [0,1]$.

1.4 Edge detection

Laplacian Operator: Find 0s in 2nd deriv I'' to locate edges. Rot. invariant, yields very noisy but thin and uninterrupted edges. $\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$

Laplacian is sensitive to noise, thus *apply blur first*:
Laplacian of Gaussian (or suppress edges w/ low gradient magnitude)

$$\mathrm{LoG}(x,y) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

Canny edge detector

Thin, uninterrupted edges, extended more completely than with simple thresh.

1. Smooth image with **Gaussian filter**
2. **Compute grad., mag. & angle** (Sobel, Prewitt, ...)
   $M(x,y) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}, \alpha(x,y) = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$
3. **Nonmaxima suppression:** quantize edges normal to 4 dirs, if smaller than either neighb. suppress
4. **Double thresholding:** $T_{\text{high}}, T_{\text{low}}$, keep if $\geq T_{\text{high}}$ or $\geq T_{\text{low}}$ and 8-conn. through $\geq T_{\text{low}}$ to T_{high} px.
5. **Hysteresis:** Reject weak edge pixels not connected with strong edge pixels

Hough transformation

Fits a curve (line, circles, ...) to set of edge pixels (e.g. $y = mx + c$)

1. Subdivide (m, c) plane into discrete bins init to 0
2. Draw line in (m, c) plane for each edge pixel (x, y), incremebnt bins by 1 along line
3. Detect peaks in (m, c) plane.

Infinite slopes arise, reparameterize line with (θ, x): $x\cos\theta + y\sin\theta = \rho$. For circles with known radius: $(x-a)^2 + (y-b)^2 = r^2$, else 3D Hough. *(Loop through all discrete values of 2 params (x,y) corresponding to bins, compute r for (x,y) update 3D bin (x,y,r) like in 2D case, find maximum in 3D bins)*

Corner detection: Edges only well localized in single direction. We need acc. local., invar. against shift, rot., scale, brightness, noise robust, repeatability. We define Local displacement sensitivity: $S(\Delta x, \Delta y) = \sum_{(x,y)\in\text{ window}}(f(x,y) - f(x+\Delta x, y+\Delta y))^2$. Using the Taylor approx. below and $\boldsymbol{M}$, we get: $f_x = I_x, \ldots$

$f(x+\Delta x, y+\Delta y) \approx f(x,y) + f_x(x,y)\Delta x + f_y(x,y)\Delta y$

$$\boldsymbol{M} = \sum_{(x,y)\in\text{ window}}\begin{pmatrix} f_x^2(x,y) & f_x(x,y)f_y(x,y) \\ f_x(x,y)f_y(x,y) & f_y^2(x,y)\end{pmatrix}$$

$$S(\Delta x, \Delta y) \approx (\Delta x \;\; \Delta y)\boldsymbol{M}(\Delta x \;\; \Delta y)^T \approx \boldsymbol{\Delta}^T\boldsymbol{M}\boldsymbol{\Delta}$$

Harris corner detection

Compute matrix $\boldsymbol{M}$. Compute $C(x,y) = \det(\boldsymbol{M}) - k\cdot(\mathrm{trace}(\boldsymbol{M}))^2 = \lambda_1\lambda_2 - k\cdot(\lambda_1+\lambda_2)^2$. Mark as corner if $C(x,y) > T$. Do non max-suppression and for better local., weight central pixels with weights for sum in $\boldsymbol{M}$: $G(x - x_0, y - y_0, \sigma)$. Compute subpixel local. by fitting parabola to cornerness function. Invariant to shift, rot, brightness offset, not scaling.

Feature Extraction

Template Matching: Check all locations, rotations, and scales. Can be done in time or frequency domain.
SIFT (Scale Invariant Feature Transform):

- Track feature points over images.
- Use Difference of Gaussian (DoG) to detect blobs.
- Compute gradient direction histograms.
- Align images, compare histograms for matches.
- $\mathrm{DoG}(x,y) = \frac{1}{k}e^{\frac{x^2+y^2}{(k\sigma)^2}} - e^{-\frac{x^2+y^2}{\sigma^2}}, k = \sqrt{2}$

1.5 Fourier transform

Fourier transform

represents signal in new basis (in amplitudes & phases of constituent sinusoids).

$$F(f(x))(u) = \int_{\mathbb{R}} f(x)\cdot\exp(-i2\pi ux)\,\mathrm{d}x$$

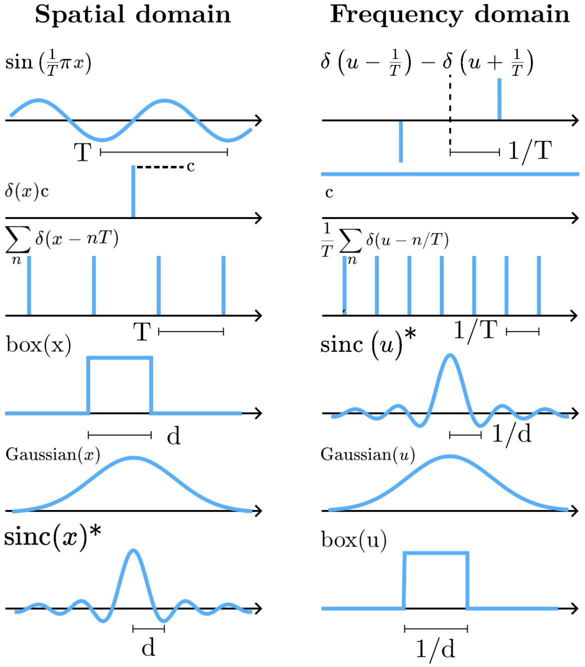
$$F(f(x,y))(u,v) = \iint_{\mathbb{R}^2} f(x,y)e^{-i2\pi(ux+vy)}\,\mathrm{d}x\,\mathrm{d}y$$

Inverse FT exists. Discrete FT: $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{f}$ where $\boldsymbol{F}$ transformed image, $\boldsymbol{U}$ FT base, $\boldsymbol{f}$ vectorized image.

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cdot e^{-i2\pi\frac{ux+vy}{N}}$$

Relevant: $\cos(x) = \frac{e^{ix}+e^{-ix}}{2}$ $\sin(x) = \frac{e^{ix}-e^{-ix}}{2}$
Dirac delta: $\delta(x) = 0$ if $x \neq 0$ else undefined with $\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1$ Sampling: Mult with seq. of δ-fnts $F(\delta(x - x_0))(u) = e^{-i2\pi u x_0}$.

Property	$\boldsymbol{f(x)}$	$\boldsymbol{F(u)}$
Linearity	$\alpha f_1(x) + \beta f_2(x)$	$\alpha F_1(u) + \beta F_2(u)$
Duality	$F(x)$	$f(-u)$
Convolut.	$(f * g)(x)$	$F(u) \cdot G(u)$
Product	$f(x)g(x)$	$(F * G)(u)$
Timeshift	$f(x - x_0)$	$e^{-2\pi i u x_0} \cdot F(u)$
Freq. shift	$e^{2\pi i u_0 x} f(x)$	$F(u - u_0)$
Different.	$\frac{\mathrm{d}n}{\mathrm{d}x^n} f(x)$	$\left(\frac{i}{2\pi} u\right)^n F(u)$
Multiplic.	$x f(x)$	$\frac{i}{2\pi} \frac{\mathrm{d}}{\mathrm{d}u} F(u)$
Scaling	$f(ax)$	$\frac{1}{\lvert a \rvert} \cdot F\left(\frac{u}{a}\right)$


Spatial domain
Frequency domain
$\sin\left(\frac{1}{T}\pi x\right)$
$\delta\left(u - \frac{1}{T}\right) - \delta\left(u + \frac{1}{T}\right)$
T
1/T
c
$\delta(x)$c
c
$\sum_n \delta(x - nT)$
$\frac{1}{T}\sum_n \delta(u - n/T)$
T
1/T
box(x)
sinc (u)*
d
1/d
Gaussian(x)
Gaussian(u)
sinc(x)*
box(u)
d
1/d

$*$) Holds if $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ (normalized), if $\text{sinc}(x) = \frac{\sin(x)}{x}$ (unnormalized) then:

$$\text{sinc}(\pi t) \to \text{Box}(u) \text{ and } \text{sinc}(\pi t) \to \text{Box}(u)$$

Image restoration

Image degradation is applying kernel h to some image f. The inverse $\tilde{h}$ should compensate: $f(x) \to h(x) \to g(x) \to \tilde{h}(x) \to f$. Determine with $\tilde{h} = F^{-1}\left(\frac{1}{F(h)}\right)$. Cancellation of freq., noise amplif. Regularize using $\tilde{F}(\tilde{h})(u,v) = F(h)/(|F(h)|^2 + \varepsilon)$

1.6 Unitary transforms (PCA / KL)

Images are vectorized row-by-row. Linear image processing algorithms can be written as $g = Ff$. Auto-correl. fun.: $R_{\mathrm{ff}} = E[f_i \cdot f_i^H] = \frac{F \cdot F^H}{n}$.

Eigenmatrix: Φ of autocorrelation matrix R_{ff}: Φ is unitary, columns form set of eigenvectors of R_{ff}: $R_{\mathrm{ff}}\Phi = \Phi\Lambda$ where Λ is a diag. matrix of eigenvecs. R_{ff} is symmetric nonneg. definite, hence $\lambda_i \geq 0$, and normal: $R_{\mathrm{ff}}^H R_{\mathrm{ff}} = R_{\mathrm{ff}} R_{\mathrm{ff}}^H$.

Karhunen-Loeve / Principal component anal.

1. Normalize (only if asked): $x_i' = \frac{x_i}{\|x_i\|}$
2. Center data by subtr. mean: $x_i'' = x_i' - \mu, \mu = \frac{1}{N}\sum x_i'$
3. Compute covar. mat.: $\Sigma = \frac{1}{N-1}\sum x_i'' \cdot (x_i'')^T$
4. Compute eigendecomp. of Σ by solving $\Sigma e = \lambda e$ with e.g. SVD ($\Sigma = U\Lambda U^T$)
5. Define U_k as first k eigenval. of Σ, $U_k = \begin{pmatrix} u_1 & \dots & u_k \end{pmatrix}$ dirs with largest variance.
6. To store: $v_i = \text{PCA}(x_i) = U_k^T(x_i - \mu) = U_k^T \cdot x_i''$

To decompress, use $\text{PCA}^{-1}(v_i) = U_k \cdot v_i + \mu$

Simple recognition, compare in projected space, find nearest neighbour. Find face by computing reconstr. error and minimizing by varying patch pos. Compress data and visualization. Eigenfaces struggle with lighting differences. Fisherfaces improve this by maximizing between-class scatter, minimzing within-class scatter.

PCA storage space

Given n images of size $x \times y$, we want to store the dataset given a budget of Z units of space. What is max number K of princip. comp. allowed?
We need to store dataset mean $\mu : x \times y$, truncated eigenmat. $U_k : (x \times y) \times K$, compr. imgs. $\{y_i\} : n \times K$

1.7 JPG & JPEG

1. Convert RGB → YUV (Y luminance / brightness, UV color / chrominance). Humans more sensitive to color, compress colors with chroma subsampling (e.g. color of upper left pixel for 4x4 grid)
2. Split image into 8x8 blocks for each YUV component, apply 2D DCT to it. 64 values, top left = low freq., bottom right = high freq. DCT: variant of DFT, fast implementation, only real values
3. Compress using int. devision with weighted matrix (Quantization table), compress bottom-right. Zig-zag run length encoding followed by Huffman.

Edges are softened because sharp edges require high freq. JPEG2000 improves by using Haar transform globally, not just on 8x8 blocks, on successively downsampled image (image pyramid)

Image pyramids: iter. applied approx. filter / downsampler. Gaussian pyramid, Laplacian is difference between 2 levels in Gaussian pyramid.
Haar transform: Scaling capture info. at diff. freq., Translate captured info. at diff. loc. Can be represented by filter+downsample. Poor energy compaction.

1.8 Optical flow

Apparent motion of brightness patterns. We set $u = \frac{\mathrm{d}x}{\mathrm{d}t}$, $v = \frac{\mathrm{d}y}{\mathrm{d}t}$, $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_t = \frac{\partial I}{\partial t}$

Assuming **brightness constancy** (e.g. the observed object does not change its brightness):

$$\begin{aligned} I(x,y,t) &= I(x + u\cdot\partial t, y + v\cdot\partial t, t + \partial t) \\ &\approx I(x,y,t) + I_x u + I_y v + I_t = 0 \quad (\star) \\ &\Rightarrow I_x u + I_y v + I_t \approx 0 \quad \text{(opt. flow const.)} \end{aligned}$$

($\star$) *Assuming* **small motion** (e.g. small $\frac{dx}{dt}\partial t, \frac{dx}{dt}\partial t, \partial t$, thus small motion per frame) using 1st-ord. Tailor series.

Aperture problem: 2 unknowns for every pixel (u,v) but only one equation $\Rightarrow \infty$ solutions, opt. flow defines a line in (u,v) space, compute normal flow. Need additional constraints to solve.

Horn & Schunck algorithm

Assumption: values $u(x,y)$, $v(x,y)$ are smooth and change slowly with x, y. Minimize $e_s + \lambda e_c$ for $\lambda > 0$ where

$$e_s = \iint \left(\left(u_x^2 + x_y^2\right) + \left(v_x^2 + v_y^2\right)\right) \mathrm{d}x \, \mathrm{d}y \text{ (smooth.)}$$
$$e_c = \iint \left(I_x u + I_y v + I_t\right)^2 \mathrm{d}x \, \mathrm{d}y \text{ (bright. const.)}$$

Coupled PDEs solved using iter. methods and finite diffs: $\frac{\partial u}{\partial t} = \Delta u - \lambda\left(I_x u + I_y v + I_t\right) I_x$ and $\frac{\partial v}{\partial t} = \Delta v - \lambda\left(I_x u + I_y v + I_t\right) I_y$. Has errors at boundaries / information spreads from corner-type patterns.

Lucas-Kanade

Assumption: neighb. in $N \times M$ patch Ω have same motion $(u \ v)^\top$: *spatial coherence*, small mov., bright. const.

$$E = \sum_{x,y \in \Omega} \left(I_x(x,y) u + I_y(x,y) v + I_t(x,y)\right)^2$$

$$MU = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix} \text{ } (\textstyle\sum \text{ over patch } \Omega)$$

First compute I_x, I_y, I_z. Let $M = \sum (\nabla I)(\nabla I)^T$ and $b = \left(-\sum I_x I_t \ \ -\sum I_y I_t\right)$ Then at each pixel, compute U by solving $MU = b$ by using least squares $U = M^{-1} b = \left(M^T M\right)^{-1} M^T b$. If M singular / fails if all gradient vec. in same dir (along edge, smooth regions). Works with corners, textures.

Complexity: Image $W \times H$, Kernel $M \times M$, with two conv/correlations: $\mathcal{O}(WHM^2)$, if separable due to just 1D conv/correlations: $\mathcal{O}(WH \cdot 2M) = \mathcal{O}(WHM)$.

Iterative refinement

Estimate OF with Lucas-Kanade. Warp image using estimate OF. Estimate OF using warped image. Refine by repeating and add up all estimates. Fails if intensity structure poor / large mov.

Gradient method fails when intensity structure within window is poor, displacement large etc.

Coarse-to-fine estimation

Create levels by gradual subsampling. Start at coarsest level, estimate OF, iterate and add until reached finest level.

Still fails if large lighting changes happen.

Affine motion

each pixel provides 1 lin. constr., 6 global unknowns. Solve LSE. From bright. const. eq.:

$$I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \approx 0$$

SSD tracking: Large displacements, extract template around pixel, match within search area, use correlation, choose best match.
Bayesian Optical flow: Some low-level human motion illusions can be explained by adding an uncertainty model to LK. E.g. bright. const. with noise.

1.9 Video compression

Visual perception < 24 Hz. Flicker > 60 Hz in periphery.
Bloch's law: if stimulus duration ≤ 100 ms, duration and brightness exchangeable. If brightness is halved, double duration. This enforces > 10 Hz for videos.

Interlaced video format: 2 temporally shifted half images (in bands) increases frequency, reduces spatial resolution. Full image repr. is progressive.

Video compression with temporal redundancy

Predict current frame based on previously coded frames. Introducing 3 types of coded frames:

1. I-frame: Intra-coded frame, coded independently
2. P-frame: Predictively-coded based on previously coded frames (e.g. motion vec. + changes)
3. B-frame: Bi-directionally predicted frame, based on both previous and future frames.

Inefficient for many scene changes or high motion.

Motion-compensated prediction

partition video into moving objects – generally pretty difficult. Practical: **block-matching motion est.**: partition each frame into blocks, describe motion of block / find best matching block in reference frame. No obj. det., good perf. Can be sped up with 3 step log search, and improve precision with half-pixel motions. We encode the residual error (with JPG). Motion vector field (set of motion vec. for each block in frame) is easy to encode, fast, simple, periodic.

MPEG Group of Pictures (GOP) starts with I-frame, ends with B- ("open") or P-frame ("closed")

1.10 CNN and Radon Transform

Given input x, learning target y, loss function $\mathcal{L}$ compute kernel weights θ using prediction $f(x,\theta)$: $\arg\min_\theta \mathcal{L}(y, f(x,\theta))$. Linear classifier $f(x,\theta) = Wx + b$ where $\theta = \{W, b\}$. Use activation func. ϕ (sigmoid, **ReLU**, tanh, …) to introduce non-linearity. Use gradient descent and back propagation (recursive appl. of chain rule to compute gradients) to find θ. Transformers, GANs, stable diffusion etc enable modern VC breakthroughs.

Classification: $f(x_1, \theta)$ as score, take class with larger score. With $y_i \in \mathbb{N}, s_i = f(x_i, \theta)$, we get loss function: $\mathcal{L}(y, f(x,\theta)) = -\sum_{i=1}^N \log\left(\frac{\exp\left(s_{i,y_i}\right)}{\sum_j \exp\left(s_{i,j}\right)}\right)$

Regression: $f(x_1, \theta)$ as value, can be used for classification by comparing value. With $y_i \in \mathbb{R}^n, s_i = f(x_i, \theta)$, we get: $\mathcal{L}(y, f(x,\theta)) = \sum_{i=1}^N \|y_i - s_i\|^2$

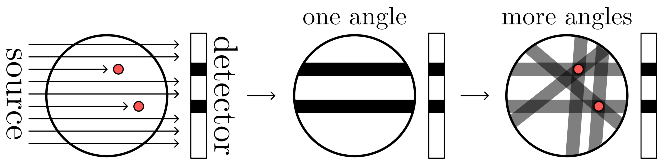
Radon transform

Given object with unknown density $f(x,y)$. Using the inverse of the RT we can find….. X-Ray along line L at distance s has intensity $I(s)$, travelling ∂s reduces intens. by ∂I. reduction depends on intens. and optical density $u(s)$: $\frac{\partial I}{I(s)} = -u(s)\partial s$. Radon transform of $f(x,y)$: $Rf(L) = \int_L f(x) \, |\mathrm{d}x|$. With $(x,y) = (\rho\cos\theta - s\sin\theta, \rho\sin\theta + s\cos\theta)$ we get: $R(\rho,\theta) = \int u(x,y) \, \mathrm{d}s$. We now want to find $u(x,y)$ given $R(\rho,\theta)$.

The continuous case of a radon transform of a line is:

$$R(\rho,\theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} u(x,y) \delta(\rho - x\cos\theta - y\sin\theta) \partial x \partial y$$

Properties of RT: Linear, shifting input shifts the RT, rotating input rotates RT, RT of 2D convolution is 1D convolution of RT with respect to ρ ($R(f *_{\text{2D}} g) = R(f) *_{\text{1D}} R(g)$, RHS with fixed θ)



Back projection: Given RT, find $u(x, y)$
Fourier slice thm (apply with all proj. angles θ):

1. Measure protection (attenuation) data
2. 1D FT of projection data
3. 2D inverse FT and sum with previous image (backpropagate)

Requires precise attn. meas., sensitive to noise, unstable, hear to implem., blurring in final image. Add high-pass filter in Fourier domain after 2nd step.

MLP (2lrs): Params $= H \cdot W \cdot N + D \cdot N$
(N: Hidden layers size, D: Output Dimension)

CNN (2lrs): Params $= (K \cdot N) + N + (K \cdot N \cdot D) + D$
(K: # Kernel entries, N: Intermediate feature map activations, D: Output feature map activations) Advantages:

- **Spatial Hierarchies:** CNNs capture local patterns and spat. hierarch. (e.g., edges to textures to objects).
- **Parameter Efficiency:** Fewer parameters due to shared weights in convolutional layers.
- **Translation Invariance:** CNNs are robust to shifts in the image due to pooling and convolution.

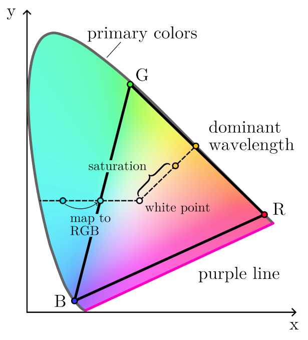
2 Computer graphics

2.1 Graphics pipeline

1. Modeling transform - from object to world-space
2. Viewing transform - from world to camera-space
3. Primitive processing - output primitives from transformed vertecies
4. 3D clipping - remove primitives outside frustum
5. Projection to screen-space
6. Scan conversion - Discretize continuous primitives, interpolate attributes at covered samples
7. Lighting, shading, texturing - compute colors
8. Occlusion handling - update color (e.g. z-buffers)
9. Display

Contemporary pipeline: CPU, Vector processing (per-vertex ops, transforms, lighting flow control), Rasterization, Fragment processing (per-fragment ops, shading, texturing, blending), Display.
Vertex shader **in:** global constants, per-vertex attributes, **out:** vertex color, vertex position
Fragment shader **in:** global constants, interpolated pixel color, **out:** pixel color

2.2 Light & Colors



Light is mixture of many wavelengths. Consider $P(\lambda)$ as intensity at wavelength λ. Humans project inf. dimens. to 3D color (RGB). CIE experiment: some colors are not comb. of RGB. (neg. red needed)

Color spaces

- **RGB** (useful for displays, RGB colors specified)
- **CIE XYZ**: Change of basis to avoid neg. comp.
- **CIE xyY**: Chomaticity (color) (x, y) derived by normalizing XYZ color components:
  $x = \frac{X}{X+Y+Z}, y = \frac{Y}{X+Y+Z}$, ($Y = Y$ is brightness)
  Inversely: $X = x\frac{Y}{y}$, $Z = (1 - x - y)\frac{Y}{y}$, $Y = Y$
- **CIE RGB**: (435.8, 546.1, 700.0nm). Linear combination span triangle, the Color Gamut.
- **CMY**: inverse (subtr.) to RGB. CMY = 1 - RGB.
- **YIQ**: Luminance Y, In-phase I (orange-blue), Quadrature Q (purple-green).
  $$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$
- **HSV**: hue (base color), saturation (purity of color), value / lightness / brightness (intuitive)
- **CIELAB / CIELUV**: Correct CIE chart colors to adjust for perceived "distance" betw. colors, nonlinear warp. MacAdams ellipses nearly circular.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_R C_R & x_G C_G & x_B C_B \\ y_R C_R & y_G C_G & y_B C_B \\ (1-x_R-y_R) C_R & (1-x_G-y_G) C_G & (1-x_B-y_B) C_B \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$
Set $(R, G, B) = (1, 1, 1)$. Map to given white point (e.g. $(0.9505, 1, 1.0890)$)), then find C_R, C_G, C_B.

2.3 Transformations

Change position & orientation of objects, project to screen, animating objects, …

Homogeneous coordinates can represent affine maps (translation) with mat.-mul. Add dimension, project vertices $(x \ \ y \ \ z \ \ w)^T$ onto $(\frac{x}{w} \ \ \frac{y}{w} \ \ \frac{z}{w} \ \ 1)^T$.

Trans. $\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Scale $\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Rot. x $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Rot. y $\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Rot. z $\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Shear x $\begin{pmatrix} 1 & \text{sh}_x^y & \text{sh}_x^z & 0 \\ \text{sh}_y^x & 1 & \text{sh}_y^z & 0 \\ \text{sh}_z^x & \text{sh}_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

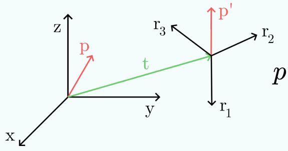
Rot (2D) $\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$ Shear (2D) $\begin{pmatrix} 1 & \text{sh}_x & 0 \\ \text{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Flip x $\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ sh_a^b: Shear a-axis in terms of b, e.g. combine sh_x^z and sh_y^z to shear xy-plane in terms of z

Rigid transforms: translation, rotation. Linear: Rotation, Scaling, Shear. Projective: Rigid + Linear + Persp. + Paral. Note: 2×2 matrices cannot express translations

Commutativity ($M_1 M_2 = M_2 M_1$) holds for: Translations, Rotations (2D), Scaling, Scaling vs Rotation

Change of coord. system



$$p' = \underbrace{\begin{pmatrix} \boldsymbol{r_0} & \boldsymbol{r_2} & \boldsymbol{r_3} & \boldsymbol{t} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{M}} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

Transforming a normal: $\boldsymbol{n}' = (\boldsymbol{M}^{-1})^T \boldsymbol{n}$

Projecting x on n: $\text{proj}_{n(x)} = \frac{x \cdot n}{n \cdot n} n$
Projections: parallel projection vs perspective.

$$M_{\text{per}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix} \quad M_{\text{ort}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Quaternions

Alternative approach for rotation. Similar to $\mathbb{C}$, define $i^2 = j^2 = k^2 = -1$, $ijk = -1$, $ij = k$, $ji = -k$, $jk = i$, $kj = -i$, $ki = j$, $ik = -j$. For $q = a + bi + cj + dk$, we have

- $\|q\| = \sqrt{a^2 + b^2 + c^2 + d^2}$
- $\overline{z} = a - bi - cj - dk$ and $z^{-1} = \overline{z} \,/\, \|z\|$

Rotating a point $p = \begin{pmatrix} x & y & z \end{pmatrix}^T$ around axis $u = \begin{pmatrix} u_1 & u_2 & u_3 \end{pmatrix}$ by angle θ.

1. Convert p to quaternion $p_Q = xi + yj + zk$
2. Convert q to quaternion $q'' = u_1 i + u_2 j + u_3 k$, normalize $q' = q'' / \|q''\|$
3. Rotate quaternion $q = \cos\frac{\theta}{2} + q' \sin\frac{\theta}{2}$ and $q^{-1} = \cos\frac{\theta}{2} - q' \sin\frac{\theta}{2}$
4. Rotated point $p' = qpq^{-1}$. Convert to cartesian.

2.4 Lighting & Shading

Lighting: Modelling physical interactions between materials and light sources
Shading: Process of determining color of pixel.
Pinhole pointing along Z axis: $x = \frac{fX}{Z}$
$\frac{dx}{dt} = \frac{f(ZV_x - XV_z)}{Z^2}, \delta x = f \delta t \frac{ZV_x - XV_z}{Z^2}$ (V is veloc./deriv.)
Note: Pinhole camera measures radiance

Basic quanitities of radiometry

Flux: $\Phi(A)$ total amount of energy passing through surface or space per unit time $\left[\frac{\text{J}}{\text{s}} = \text{W}\right]$
Luminous flux: $\int P(\lambda) V(\lambda) \, \text{d}\lambda$. Perceived power of light weighted by human sensitiv. [lumen]
Irradiance: $E(x) = \frac{\text{d}\Phi(A)}{\text{d}A(x)}$ flux per unit area arriving at surface $\left[\frac{\text{W}}{\text{m}^2}\right]$.
Radiosity: $B(x) = \frac{\text{d}\Phi(A)}{\text{d}A(x)}$ flux per unit area leaving a surface $\left[\frac{\text{W}}{\text{m}^2}\right]$
Radient / Luminous Intensity: $I(\vec{\omega}) = \frac{\text{d}\Phi}{\text{d}\vec{\omega}}$ outgoing flux per solid angle $\left[\frac{\text{W}}{\text{sr}}\right]$. $\Phi = \int_{S^2} I(\vec{\omega}) \, \text{d}\vec{\omega}$
Radiance: $L(x, \vec{\omega}) = \frac{\text{d}I(\vec{\omega})}{\text{d}A(x)} = \frac{\text{d}^2\Phi(A)}{\text{d}\vec{\omega} \, \text{d}A(x) \cos\theta}$ flux per solid angle per perp. area = intensity per unit area

Lamberts cosine law: Irradiance at surface is proportional to cosine of angle of incident light and surface normal: $E = (\Phi \,/\, A) \cos\theta$
Bidir. reflectance distr. func. (BRDF): relation between incident radiance and diff. refl. radiance.
$f_r(x, \vec{\omega}_i, \vec{\omega}_r) = \frac{\text{d}L_r(x, \vec{\omega}_r)}{\text{d}E_i(x \vec{\omega}_i)} = \frac{\text{d}L_r(x, \vec{\omega}_r)}{L_i(x, \vec{\omega}_i) \cos\theta_i \, \text{d}\vec{\omega}_i}$
Reflection equation: reflected radiance due to incident illumination from all directions (from BRDF): $\int_{H^2} f_r(x, \vec{\omega}_i, \vec{\omega}_r) L_i(x, \vec{\omega}_i) \cos\theta_i d\vec{\omega}_i = L_r(x, \vec{\omega}_r)$
Types of reflections: Exact comp. is slow → Specular, ideal diffuse, glossy specular, retro-reflective.
Attenuation: $f_{\text{att}} = \frac{1}{d_L^2}$ due to spatial radiation.

Phong Illumination Model

Approximate specular reflection by cosine powers

$$I_\lambda = \underbrace{I_{a_\lambda} k_a O_{d_\lambda}}_{\text{Ambient}} + f_{\text{att}} I_{p_\lambda} [\underbrace{k_d O_{d_\lambda} (N \cdot L)}_{\text{Diffuse}} + \underbrace{k_s (\boldsymbol{R} \cdot \boldsymbol{V})^n}_{\text{Specular}}]$$

h_i own emission coefficient, I_a ambient light intensity, k_a ambient light coefficient, I_p directed light source intensity, k_d diffuse reflection coefficient, $\theta \in [0, \frac{\pi}{2}]$ angle surface normal N and light source vector L, attenuation factor f_{att}, $O_{d\lambda}$ value of spectrum of object color at the point λ.

k_a, k_d, k_s, n are material dependent constants.

2.5 Shading models

Flat shading: one color per primitive

Gouraud Shading

Lin.interpol. vertex intensities

1. Calculate face normals
2. Calculate vertex normals by averaging *(weighted by angle)*
3. Evaluate illumination model for each vertex
4. Interpolate vertex colors bilinearly: For each y the scan line first interp. the color at the scanline/triangle-edge interesection linearly (by y), which we then use to interp. all pixels inbetween by x.

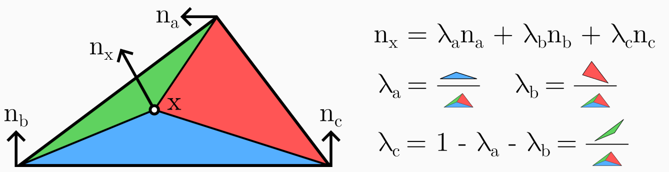
Problems with scan line interpolation are perspective distortion, orientation dependence, shared vertices. Quality depends on size of primitives.

Phong Shading

Get $\boldsymbol{R}$ by reflecting the light vec. $\boldsymbol{L}$ ar. the normal $\boldsymbol{N}$:

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

Barycentric interpolation of normals on triangles.



Properties: Lagrange: $x = a \Rightarrow n_x = n_a$
Partition of unity: $\lambda_a + \lambda_b + \lambda_c = 1$
Reproduction: $\lambda_a \cdot a + \lambda_b \cdot b + \lambda_c \cdot c = x$.

Problems: normal not defined / representative.

Baryc. coord. of point q is λ s.t. $\lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3 = q$

Flat, Gouraud in screen space, Phong in obj. space

Transparency: (2 obj., P_1 & P_2). Perceived intensity $I_\lambda = I'_{\lambda_1} + I'_{\lambda_2}$ where I'_{λ_1} is emission of P_1 and I'_{λ_2} is intensity filtered by P_1. We model it as follows: $I_\lambda = I_{\lambda_1} \alpha_1 \Delta t + I_{\lambda_2} e^{-\alpha_1 \Delta t}$ where α absorption, Δt thickness. Linearization: $I_\lambda = I_{\lambda_1} \alpha_1 \Delta t + I_{\lambda_2} (1 - \alpha_1 \Delta t)$. If last object, set $\Delta t = 1$. Usually: $\alpha_A A + (1 - \alpha_A) \alpha_B B$
Problem: rendering order, sorted traversal of polygons and back-to-front rendering.

Back-to-front order not always clear (as it could be in the both in front and back of another object, resort to depth peeling, multiple passes, each pass renders next closest fragment.

2.6 Geometry and textures

Considerations: Storage, acquisition of shapes, creation of shapes, editing shapes, rendering shapes.

Explicit repr. can easily model complex shapes and sample points, but take lots of storage:

- **Subdivision surfaces**, define surfaces by primitives, use recursive algorithm for refining
- **Point set surfaces**, store points as surface
- **Polygonal meshes**, explicit set of vertices with position, possibly with additional information. Intersection of polygons: either empty, vertex, edge
- **Parametric surfaces**, represent points as functions of parameters ((u, v)), e.g., ((x, y, z) = f(u, v)).

Implicit repr. easily test inside / outside, compact storage, but sampling expensive, hard to model:

- **Implicit surfaces**, surface: zeros of a function
- **Level set**, a function that is zero on the surface
- **Fractals**, self-similar, recursive, infinite detail

Manifolds: Each edge is shared by at most two faces; the surface is connected and locally looks like a plane.

Mesh data structures: Locations, how vertices are connected, attributes such as normals, color etc. Must support rendering, geometry queries, modifications. E.g.

- Triangle list (list of coord. and list of indices to that coord. list where every 3 elements repr. a triangle).
- Indexed Face set (array of vertices + list of indices, e.g. OBJ, OFF, WRL, costly queries, modifications)

Texture mapping

Enhance details without additional geom. complexity. Map Texture (u, v) coords to geom. (x, y, z) coords. Issues: aliasing, level-of-detail, (e.g. **sphere mapping**: $(u, v) \to (\sin u \sin v, \cos v, \cos u \sin v)$). We want low-distortion, bijective mapping, efficiency.

Bandlimiting: Restrict amplitude of spectrum to zero for frequency beyond cut-off frequency.

Anti-aliasing filters:

- Gaussian (low-pass, separable). Closer pixels have higher weight
- Sinc $S_{\omega_c}(x, y) = \frac{\omega_c \operatorname{sinc}(\frac{\omega_c x}{\pi})}{\pi} \times \frac{\omega_c \operatorname{sinc}(\frac{\omega_c y}{\pi})}{\pi}$ ideal low-pass filter, IIR filter. ω_c cutoff freq. Hard to implement, has infinite impulse respons.
- B-Spline: Allow for locally adaptive smoothing, adapt size of kernel to signal chars.

Magnification: for pixels mapping to area larger than pixel (jaggies), use bilinear interpolation.

Mipmapping: Store texture at multiple resolutions, choose based on projected size of triangle (far away → lower res). Avoids aliasing, improves efficiency, storage overhead. Uses at most $\sum_{k=1}^{\infty} \left(\frac{1}{4}\right)^k = \frac{1}{3}$ the storage.

Supersampling: Multiple color samples per pixel, final color is average - different patterns of samples possible (uniform, jittering, stochastic, Poisson).

Light map: Simulate effect of local light source. Can be pre-computed and dynamically adapted
Environment map: Mirror environment with imaginary sphere or cube for reflective objects
Bump map: Perturb surface normal according to texture, represents small-scale geometry. No bumps on silhouette, no self-occlusions, no self-shadowing.
Displacement map: Directly modifies the geometry by offsetting vertices based on texture. Fixes bump map issues, but more computationally expensive.

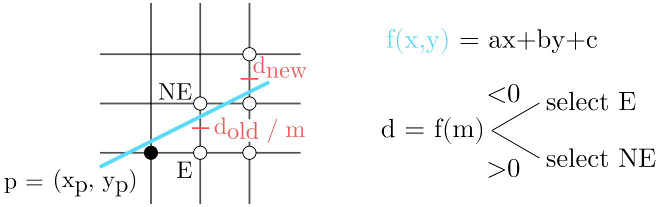
Procedural texture: Generate texture from noise (Perlin, Gabor) from Guassian pyramid of noise and summing layers with weights.

Perspective interpolation in world-space can yield non-linear variation in screen-space. Optimal resampling filter is hence spatially variant.

	Primal		**Dual**
	Triangles	Rectangles	
Approx.	Loop	Catmull-Clark	Doo-Sabin
Interpol.	Butterfly	Kobbelt	

2.7 Scan conversion (rasterization)

Generate discrete pixel values.
Supersampling: Sample in uniform grid, calcualte percentage and choose color multiplied by percentage.
Bresenham line: choose closest pixel at each intersection. Fast decision, criterion based on midpoint m and intersection point q. After computing first value d, we only need addition, bitshifts. $d_{\text{new}} = d_{\text{old}} + a$



Scan conversion of polygons:

1. Calculate intersections on scan line
2. Sort intersection points by ascending x-coords.
3. Fill spans between two consecutive intersection points if parity in interval is odd (so inside)

2.8 Bézier Curves & Splines

Bézier Curves

$\boldsymbol{x}(t) = \boldsymbol{b}_0 B_0^n(t) + ... + \boldsymbol{b}_n B_n^n(t)$
where B_i^n are the Bernstein polyn. $(\binom{n}{i} = \frac{n!}{i!(n-i)!})$:
$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$ if $i \neq 0$ else 0
(partition of unity, pos. definite, recursion, symmetry). Coefficients $\boldsymbol{b}_0, ..., \boldsymbol{b}_n$ are called control- / Bézier-points, together define the control polygon.

Degree is highest order of polyn., order is deg. + 1. Bézier-Curves have **affine invariance** (affine transf. of curve accompl. by affine transf. of control pts.). Curve lies in **convex hull** of control polygon $\text{conv}(P) :=$ $\left\{\sum_{i=1}^{n} \lambda_i \boldsymbol{p}_i \mid \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1\right\}$. Control polyg. gives **rough sketch** of curve. Since $B_0^n(0) = B_n^n(1) = 1$, **interpol. endpoints $\boldsymbol{b}_0, \boldsymbol{b}_n$**. **Max. number of intersect**. of line with curve is $\leq$ number of intersect. with control polygon. Note: Bézier curves are special cases of B-spline

Disadvantages: global support of basis functions (change one point, change entire curve), new control pts yields higher degree, C^r continuity between segments of Bézier-Curves (deriving r times .

Interpolate points $\boldsymbol{p}_0, ..., \boldsymbol{p}_n$ using basis fcts.

B-Spline functions

B-Spline curve $\boldsymbol{s}(u)$ built from piecewise polyn. bases $s(u) = \sum_{i=0}^{k} \boldsymbol{d}_i N_i^n(u)$
Coefficients $\boldsymbol{d}_i$ are called “de Boor” pts. Bases are piecewise, recursively def. polyn. over sequence of knots $u_0 < u_1 < u_2 < \ldots$ defined by knot vec. $T = \boldsymbol{u} = \begin{pmatrix} u_0 & \cdots & u_{k+n+1} \end{pmatrix}$

Partition of unity $\sum_i N_i^n(u) \equiv 1$, positivity $N_i^{n(u)} \geq 0$, compact support $N_i^{n(u)} = 0,\ \forall u \notin [u_i, u_{i+n+1}]$, continuity N_i^n is $(n-1)$ times cont. differentiable.

$$N_i^n(u) = (u - u_i) \cdot \left(N_i^{n-1} \frac{u}{u_{i+n} - u_i}\right)$$

$$+(u_{i+n+1} - u) \cdot \left(N_{i+1}^{n-1} \frac{u}{u_{i+n+1} - u_{i+1}}\right)$$

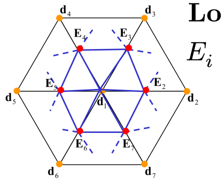
Examples with $n = 1$ and $n = 2$:

$$N_i^1(u) = \begin{cases} \frac{u - u_i}{u_{i+1} - u_i} \text{ if } u \in [u_i, u_{i+1}] \\ \frac{u_{i+2} - u}{u_{i+2} - u_{i+1}} \text{ if } u \in [u_{i+1}, u_{i+2}] \end{cases}, \quad N_i^2(u) =$$

$$(u - u_i) \cdot \left(N_i^1 \frac{u}{u_{i+2} - u_i}\right) + (u_{i+3} - u) \cdot \left(N_{i+1}^1 \frac{u}{u_{i+3} - u_{i+1}}\right)$$

2.9 Subdivision surfaces

Generalization of spline curves / surfaces allowing arbitrary control meshes using successive refinement (subdivision), converging to smooth limit surfaces, connecting splines and meshes.
Interpolating: new points are an interpolation of control points *vs* **Approximating:** controls points are moved, and new points are added inbetween.
Primal: faces are split *vs* **Dual:** vertices are split

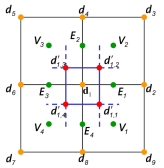


Loop Subdivision

$$E_i = \left(\tfrac{3}{8}\right) \cdot (d_1 + d_i) + \left(\tfrac{1}{8}\right) \cdot (d_{i-1} + d_{i+1})$$

$$d_{1'} = \alpha_n \cdot d_1 + \frac{1 - \alpha_n}{n} \cdot \sum_{j=2}^{n+1} (d_j)$$

$$\alpha_n = \tfrac{3}{8} + \left(\tfrac{3}{8} + \tfrac{1}{4n} \cos(2\pi)\right)^2$$



Doo-Sabin Subdivision

$$V_2 = \frac{1}{n} \sum_{j=1}^{n} d_j, E_i = \tfrac{1}{2}(d_1 + d_{2i})$$

$$d_{(1,j)'} = \tfrac{1}{4}(d_1 + E_j + E_{j-1} + V_j)$$

2.10 Visibility and shadows

Painter’s algorithm: Render objects from furthest to nearest. Issues with cyclic overlaps &intersec.
Z-Buffer: store depth to nearest obj for each px. Initialize to ∞, then iter. over polygons and update z-buffer. Limited resolution, non-linear, place far from camera
Planar shadows: draw projection of obj. on ground. No self-shadows, no shadows on other objects, curved surfaces.
Projective texture shadows: Separate obstacles and receivers, compute b/w img. of of obst. from light, use as projective textrue. Need to specify obstacle & receiver, no self-shadows.

Shadow maps

Compute depths from light & camera. For each px, get world coords, project to light plane, compare $d(x_L)$ (dist. light → projected point) to z_L (dist. light → x_L). If $d(x_L) < z_L$, x is in shadow. Add bias: $d(x_L) + b < z_L$ to prevent self-shadowing. Use cubical shadow map or spotlights if shadow point is out of view. Filter depth test to reduce aliasing.

Shadow volumes: Count shadow volume boundary crossings along a ray—if > 0, the polygon is shadowed. Use silhouette edges to optimize, but this adds geometry and increases rasterization cost. Objects must be watertight with precise rasterization.

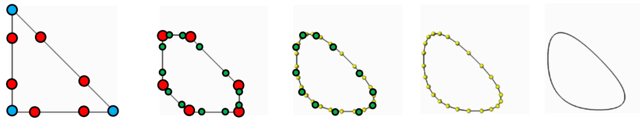
2.11 Ray Tracing

Send rays into scene to determine color of px. On obj. hit, send mult. rays (diffused, reflected, refracted) further until we hit light source, or reach some amount of bounces. Figure out whether point in shadow by shooting rays to all light sources. For anti-aliasing, use mult. rays per px. (Pipel.: Ray Generation, Intersection, Shading)

Ray-surface intersections: Ray equation $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$. For the following, solve for t and then put get position with ray equ. **Plane:** $0 = \boldsymbol{n} \cdot (\boldsymbol{o} + t\boldsymbol{d}) - c$ **Sphere:** $\|\boldsymbol{o} + t\boldsymbol{d} - \boldsymbol{c}\|^2 - r^2 = 0$. **Triangle**: Barycentric coords. $\boldsymbol{x} = s_1 \boldsymbol{p}_1 + s_2 \boldsymbol{p}_2 + s_3 \boldsymbol{p}_3$. Intersect: $(\boldsymbol{o} + t\boldsymbol{d} - \boldsymbol{p}_1) \cdot n = 0$. Using the following: $\boldsymbol{n} = (\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)$ we get $t = -\frac{(\boldsymbol{o} - \boldsymbol{p}_1) \cdot \boldsymbol{n}}{\boldsymbol{d} \cdot \boldsymbol{n}}$. Now compute the coeffs. $\boldsymbol{s}_i$. Test whether $s_1 + s_2 + s_3 = 1$ and $0 \leq s_i \leq 1$. If so, inside triangle.

Ray-tracing shading extensions: Refraction, mult. lights, area lights for soft shadows, motion blur (sample objs, intersect in time), depth of field
Cost: $O(N_x N_y N_o) = \#\text{px} \cdot \#\text{objects}$

Accelerate: Accelerate with less intersections, introduce uniform grids or space partitioning trees.
Uniform grids: Divide space into cells, test ray against cells, then against objects in cell.
Space partitioning trees: octree, kd-tree, bsp-tree
Corner Cutting: Insert vertices at $\frac{1}{4}$, $\frac{3}{4}$ of each edge, remove old vertecies, connect them and repeat.



deCasteljau: Compute a triangular representation, successively interpolate, “corner cutting”:

$$\begin{array}{llll} b_0 & & & \\ b_1 & b_0^1 & & \\ b_2 & b_1^1 & b_0^2 & \\ b_3 & b_2^1 & b_1^2 & b_0^3 \end{array}$$

Consider the three control points b_0, b_1, b_2:

$$b_0^1(t) = (1-t)b_0 + tb_1$$

$$b_1^1(t) = (1-t)b_1 + tb_2$$

$$b_0^2(t) = (1-t)b_0^1(t) + tb_1^1(t)$$

deBoor algorithm: generalizes deCasteljau, evaluate B-spline of degree n at u, set $d_i^0 = d_i$, finally $d_0^n = s(u)$

$$d_i^k = (1 - a_i^k) d_i^{k-1} + a_i^k d_{i+1}^{k-1}, \quad a_i^k = \frac{u - u_{i+k-1}}{u_{i+n} - u_{i+k-1}}$$

Note that a_i^k vanishes outside of $[u_{i+k-1}, u_{i+n}]$!

```
d_i = [d[j + i - n] for j in range(0, n + 1)]
for k in range(1, n + 1):
  for j in range(n, k - 1, -1):
    a_i_k = (u - u_knots[j + i - n]) / (u_knots[j
+ i + 1 - k] - u_knots[j + i - n])
    d_i[j] = (1.0 - a_i_k) * d_i[j - 1] + a_i_k *
d_i[j]
return d_i[n]  # This is d_0^n = s(u)
```