1 Computer Vision

1.1 The digital image

An image is a representation of a continuous function. A pixel is a discrete sample of that function.
Digital images have many challenges: transmission interference, compression artifacts, spilling, scratches, sensor noise, bad contrast / resolution, motion blur.

Charge Coupled Device (CCD) An array of photosites capture photons and hold a charge proportional to the light intensity. We measure the charge with an **analog-digital converter (ADC)** line by line.

1. Blooming: finite bucket capacity, oversaturation causes bright vertical line
2. Bleeding/smearing: happens only with electronic shutters, worse for shorter shutter times
3. Dark current: thermally generated charges yield noise despite darkness, worsens with age

CMOS Same sensor elements as CCD. Each has its own amplifier (more noise, reduce by substracting black image). VS CCD: Less sensitive, per-pixel amplification, random pixels access, no blooming, on chip integration.

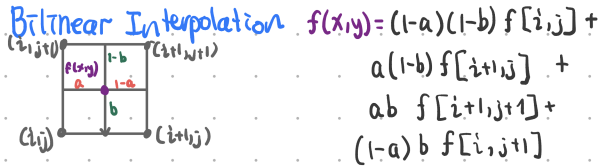
Sampling Methods Cartesian (grid), hexagonal, non-uniform. Undersampling: indistinguishable from lower frequencies

Aliasing Signals 'traveling in disguiseäs other frequencies: there are sine waves that go through the exact same points, undersampling: difference sine wave

Quantization Lossy, sampled but not quantised can be reconstructed. Can approximate. Simple quantization uses $k = 2^b$ equally spaced intervals.

Image Noise Fluctuations on measurement, commonly modeled by additive Gaussian noise with $I(x,y) = f(x,y) + c$ with $c \sim \mathcal{N}(0,1)$ $p(c) = \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(c-\mu)^2}{2\sigma^2})$ Poisson (shot Noise) as $p(k) = \frac{\lambda^k e^{-\lambda}}{k!}$ where λ is the expected number or photons per time interval, proportional to incident scene irredence

Signal to Noise Ration (SNR) index of image quality $s = \frac{F}{\sigma}$ where $F = \frac{1}{XY}\sum_{x=1}^{X}\sum_{y=1}^{Y} f(x,y)$



Resolution Geometric (# pixels / area), Radiometric (# bits/pixel), Image (Cropping)

Colour Cameras

1. Prism: expensive, need 3 sensors, good alignment, high frame rate, high separation, 3 bands, low artifacts
2. Filter Wheel: Medium price, multiple filters in front sensor, good separation, low frame rate, 3+ bands.
3. Filter Mosaic: Single sensor, coat filter directly on sensor, average separation, high frame rate, 3 bands.
4. CMOS layers that directly obsorb colours (better quality)

1.2 Image Segmentation

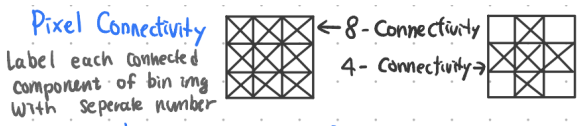
Complete Segmentation finite set of non-overlapping regions: $I = \bigcup_{i=1}^{N} R_i$ and $R_i \cap R_j = \varnothing \forall i \neq j$

Thresholding Compare greylevel with image to decide if in or out bin image `B(x,y)=1 if I(x, q) >= T else 0`

Chromakeying When planning to segment, use special background colour, plain distance measure $I_\alpha = |I - g| > T$. Problems: Variation due to lighting, noise, ..., mixed pixels

Mahalanobis distance more sophisticated segmentation formula (accounts for variance): $\sqrt{(x-\mu)^\top \Sigma^{-1}(x-\mu)} > T$, T is threshold. Σ is the covariance matrix with $\Sigma_{ij} = \mathbb{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right]$, estimate it from n data points (at least 3 needed): $\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^\top$

ROC curve Describes performance of binary classifier. X-axis is FPR $= {}^{\text{FP}}/_{\text{FP}+\text{TN}}$, the y-axis is TPR $= {}^{\text{TP}}/_{\text{TP}+\text{FN}}$. We can choose a good operating point with gradient $\beta = \frac{N}{P} \cdot \frac{V_{\text{TN}} + C_{\text{FP}}}{V_{\text{TP}} + C_{\text{FN}}}$ with V being value and C being cost.



Connected Component Raster Scanning Scanning row by row, if foreground label it to connected to other label else give new label. 2nd pass to find equivalent labels.

Contour-Based Method When region found, follow border then carry on.

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

Seed Region By hand or auto by conservative thresholding.

Inclusion Criteria Grey level thresholding, grey level distribution model. Include if $(I(x,y) - \mu)^2 < (n\sigma)^2$, update μ, σ after each iteration

Snakes (Active Contours) Initialize contour outside region of interested, image often blurred first, interatively minimizes energy function $E = E_{\text{tension}} + E_{\text{stiffness}} + E_{\text{image}}$

Background Subtraction Simple: $\mathbf{I}_\alpha = |\mathbf{I} - \mathbf{I}_{bg}| > \mathbf{T}$, or better $\mathbf{I}_\alpha = |\mathbf{I} - \mathbf{I}_{bg}|^\top \Sigma^{-1} |\mathbf{I} - \mathbf{I}_{bg}| > \mathbf{T}$

Morphological Operators Local pixel transformations for processing region shapes, most often used on binary images. `args: binary image, structuring element`

Structuring Elements Binary array, has an origin, small set to probe image under study. Check Fit, Hit, Miss Conditions.

Erosion $\ominus$ Set of all points in the image where SE fits into. (Split apart joined objects, strip extrusions,, shrink objects) `1 at origin if SE matches else 0`

Dilation $\oplus$ Set of all points in image where SE hits FG (repair breaks, repair intrusions, enlarges objects)

1.3 Image Filtering

Modification of pixels in an image based on same function of a local neighborhood of pixels

Linear $I'(x,y) = \sum (i,j) \in N(x,y) K(x,y;i,j) I(i,j)$, substitute $I(i,j) = \alpha I_1(x,y) + \beta I_2(x,y)$ to check if $L[\alpha I_1 + \beta I_2] = \alpha L[I_1] + \beta L[I_2]$

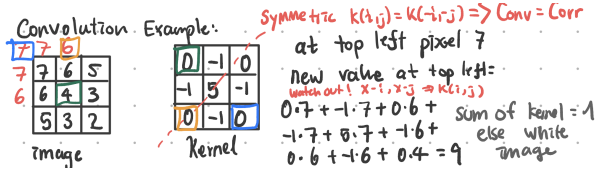
Separable If a kernel can be written as a product of 2 simpler filters. `sobel = [1 2 1].T @ [1 0 1]`

Shift Invariant Doing the same for each pixel, K does not depend on (x,y)

Filter at Edges clip to black, wrap around, copy edge, reflect across edge, vary filter near edge

Correlation $I' = K \circ I$ (e.g. template matching: search for best match by minimizing MSE) Taking the neighbours of a pixel and and performing these operations. → Locating a template in a larger image. Equivelent to Convolution if $K(i,j) = K(-i,-j)$ $I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j)I(x+i, y+j)$

Convolution Same as correlation, except kernels reversed. $I' = K * I$ and $I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j)I(x-i, y-j)$ Continuous: $g(x) = f(x)*k(x) = \int_{\mathbb{R}} f(a)k(x-a)\,da$. linear, associative, shift-invariant and commutative if the dimensions are identical.



1.4 Kernels

Low-pass	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	High-pass	$\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$
Laplacian	$\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$	Prewitt ($\frac{\partial I}{\partial y}$)	$\begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&1\end{bmatrix}$
Sobel ($\frac{\partial I}{\partial y}$)	$\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}$	Band-pass	high $*$ low
Diff. (x)	$\begin{bmatrix}-1&1\end{bmatrix}$	Roberts ($\frac{\partial I}{\partial y}$)	$\begin{bmatrix}0&1\\-1&0\end{bmatrix}$
Diff. (y)	$\begin{bmatrix}-1\\1\end{bmatrix}$	Gaussian (G_σ)	$\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$

Smoothing Kernels (LPF) Allows frequencies lower than certain frequencies to pass and attenuates frequencies above the cutoff frequencies. Further = less effect.

Gaussian Kernel Rotational Symmetric, has a single lobe (neighbors influence decrease monotonically), still one lobe in frequency domain (no corruption from high frequencies), simple relationship to σ. Subtracting 1 from central element of LPF gives LPF with inverted sign: $(f-\delta)*a = f*a-\delta*a-a = -(a-(f*a))$

Scale Space Convolution of a Gaussian with σ with itself is a Gaussian $\sigma\sqrt{2}$, repeated convolution by a Gaussian filter produces scale space of an image.

Image Sharpening (Enhancement) Increases high frequencies components to enhance edges $I' = I + \alpha|K * I|$ where K is a HPF, and $\alpha \in [0,1]$

Differentiation and Convolution `K=[-1 1].T`

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

1.5 Image Features

Edge Detection Magnitude: $|\nabla f(x,y)| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$ Angle(Orientation) $\alpha(x,y) = \tan^{-1}(\frac{\partial f}{\partial y}/\frac{\partial f}{\partial x})$ For continuous space.

Edge Detection Filters Large operators (Prewitt, Sobel) - poor localization, + less noise sensitive + good detection, usually convolved by a Gaussian. Small operators (Roberts) + Good localization, sensitive to noise, poor detection.

Edge Thresholding *Thick Edges, sensitive to noise, scattered thin noisy edges, clean background as discards low gradients*

1. Standard: $||\nabla I(x,y)|| < T$ Definitely not an edge, else definitely an edge.
2. Double Thresholding $T_0 \le ||\nabla I(x,y)|| < T_1$ only edge if neighbour is definitely an edge.

Edge Suddent change of brightness, can use local maxima of first derivative or zero crossings of 2nd derivative.

Laplacian zero-crossings Find 0's in I'' by applying Laplacian kernel to I. This yields very noisy but thin and uninterrupted edges. Very sensitive, so blur first (Laplacian of Gaussian) or suppress edges with low gradient magnitude, Isotropic (Rotationally Invariant)

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

Canny edge detector Has thin interrupted edges that are extended more than with simple thresholding

1. Smooth image with Gaussian
2. Compute grad. mag. and orientation (Sobel, Prewitt)
3. Non-maxima suppresion: quantize edge normal to one of four dirs, if magnitude < either neighbour then suppress, else keep
4. Double thresholding: given T_{high}, T_{low}, strong pixel if $\ge T_{high}$ and weak pixel if $\ge T_{low}$
5. Reject weak pixels not 8-connected through weak pixels to a strong pixel

Hough transform Fits straight lines to edge pixels ($y = mx + c$)

1. Subdivide (m,c)-space into discrete bins with value
2. Draw a line in (m,c)-space for each edge pixel and increment bins by 1 along line
3. Detect peaks, e.g. by thresholding after non-maximum suppresion

Infinite slopes are a problem, so reparametrize line with (θ, p) : $x\cos(\theta) + y\cdot\sin(\theta) = \rho$. For detecting circles with known radius use $(x-a)^2 + (y-b)^2 = r^2$ else with parameters (x_0, y_0, r)

Corner Detection Edges well localized only in one direction. Desire properties of a corner detector: Accurate localization, invariant against shift, rotation, scale, brightness change. Robust against noise. High Repeatability.

Harris corner detection Scan picture with a given window size and compute the second moment matrix

$$M = \begin{bmatrix}\sum I_x(x,y)^2 & \sum I_x(x,y)\cdot I_y(x,y)\\ \sum I_x(x,y)\cdot I_y(x,y) & \sum I_y(x,y)^2\end{bmatrix}$$

Then compute $C(x,y) = \det(M) - k\cdot(\text{trace}(M))^2 = \lambda_1\cdot\lambda_2 - k(\lambda_1+\lambda_2)^2$ and mark as corner if $C(x,y) > T$. Do non-maximum suppression to avoid duplicate detections and for better localization, add gaussian weighting to all terms in M: $G(x-x_0, y-y_0, \sigma)$. Invariant to shift, rotation and brightness offset but not scaling

Scale Invariant Feature Transform (SIFT) Find corresponding feature points in two images. Look for strong responses of Difference of Gaussian (DoG) over scale space and position, consider local maxima in both spaces to find blobs. Compute histogram of gradient directions (ignoring grad. mag. because of lighting etc.) at selected scale and

position and correct rotation by choosing principal direction. Now both pictures are at the same scale & orientation, we can compare gradient histograms to find matching points

$$\text{DoG}(x,y) = {}^1\!/\!_k \cdot e^{\frac{x^2+y^2}{(k\sigma)^2}} - e^{-\frac{x^2+y^2}{\sigma^2}}, \quad \text{e.g. } k=\sqrt{2}$$

1.6 Fourier Transform

The Fourier Transform (FT) represents a signal f in terms of amplitudes and phases of its constituent sinusoids.
1D:

$$F(u) = \int_{\mathbb{R}} f(x)\cdot e^{-i2\pi ux}\,dx$$
$$f(x) = \int_{\mathbb{R}} F(u) e^{i2\pi ux}\,du \qquad \text{(inverse)}$$

2D:

$$F(u,v) = \iint_{\mathbb{R}^2} f(x,y)\cdot e^{-i2\pi(ux+vy)}\,dx\,dy$$
$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cdot e^{-i2\pi(\frac{ux+vy}{N})}\,dx\,dy$$
$$f(x,y) = \iint_{\mathbb{R}^2} F(u,v) e^{i2\pi(ux+vy)}\,du\,dv \quad \text{(inverse)}$$

$$e^{-i2\pi(ux+vy)} = \cos(2\pi(ux+vy)) + i\cdot\sin(2\pi(ux+vy))$$

$$\cos(x) = \frac{e^{ix}+e^{-ix}}{2},\ \sin(x) = \frac{e^{ix}-e^{-ix}}{2}$$

Direc-Delta $F(\delta(x-x_0))(u) = e^{-i2\pi ux_0}$

Phase $\varnothing(F) = \tan^{-1}(\text{Im}(F)/\text{Re}(F))$ All natural images have about the same magnitude transform, so phase matters a lot

Magnitude $|F| = \sqrt{\text{Re}(F)^2 + \text{Im}(F)^2}$

Fact $F(g(\frac{x}{a}))(u) = aF(g(x))(au)$

Sampling A sampling function $s(t)$ is an impulse train with period T and its FT $S(f)$: $s(t) = \sum_{n=-\infty}^{\infty}\delta(t-nT)$, $S(f) = \frac{1}{T}\sum_{n=-\infty}^{\infty}\delta(f-\frac{n}{T})$.

Sampled Function $X(t)$ after sampled: $X_s(t) = X(t)s(t)$.

Nyquist Sampling Theorem The sampling frequency must be at least twice the highest frequency $\omega_s \geq 2\omega$. If not possible then bandlimit before with LPF.

Tolerance ϵ, solve for σ $\exp(\frac{-\omega^2\sigma^2}{2}) < \epsilon$

Kernel Size If signal is discrete, you want your kernel to be large enough to capture the significant part of the Gaussian function's curve. $k = 2\lceil 3\sigma\rceil + 1$

FT of sampled function let $\omega = \frac{n}{T}$

$$\begin{aligned}\mathcal{F}(X_s(t)) &= \mathcal{F}(X_s(t)) * \mathcal{F}(s(t))\\ &= X(f)S(f)\\ &= \frac{1}{N}\sum_{n=-\infty}^{\infty} F(u-\frac{n}{T})\end{aligned}$$

Nyquist Proof If we want to reconstruct the original signal $f(t)$ from $\mathcal{F}(X_s(t))$, then $\tilde{\mathcal{F}}(u)\cdot\mathcal{F}(w)$ cannot overlap with its neighbors $\mathcal{F}(\omega-\omega_s)$ and $\mathcal{F}(\omega+\omega_s)$, thus ω_s should be larger than 2ω

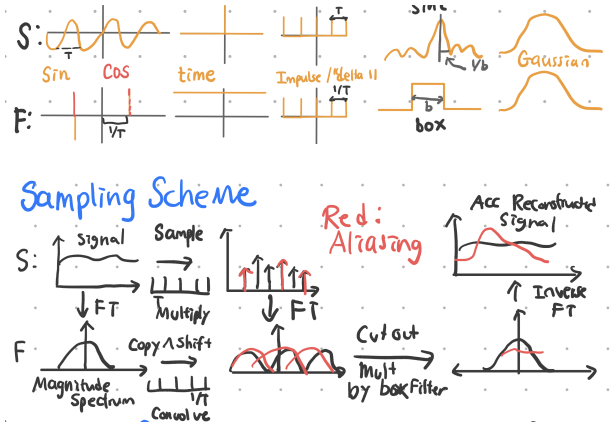
Properties of the FT

Property	$f(x)$	$F(u)$
Linearity	$af_1(x)+bf_2(x)$	$aF_1(u)+bF_2(u)$
Duality	$F(x)$	$f(-u)$
Convolution	$(f*g)(x)$	$F(u)\cdot G(u)$
Product	$f(x)\cdot g(x)$	$(F*G)(u)$
Timeshift	$f(x-x_0)$	$e^{-2\pi iux_0}\cdot F(u)$
Freq. shift	$e^{2\pi iu_0x}f(x)$	$F(u-u_0)$
Differentiation	${}^{dn}\!/\!_{dx^n} f(x)$	$(i_2\pi u)^n F(u)$
Multiplication	$xf(x)$	${}^{i}\!/\!_{2\pi}({}^{d}\!/\!_{du}F(u))$

Fact A filter whose FT is a box is bad because filter is infinite support, but a good LPF is a box filter because it's just the weighted average of neighbor pixels.

Restoration / Deconvolution Given image $g = f*h$ where f is the original and h is a filter, it holds that $G = F\cdot G$ and we can use the inverse of the filter H^{-1} to get $F = G\cdot H^{-1}$ and apply inverse FT to restore f. If noise n is present ($g = f*h+n$), we use a pseudoinverse with limited maximal value $\tilde{H} = \frac{H}{|H|^2+\epsilon}$, where ϵ is a small number. Done because the noise might dominate in some frequencies and the restoration would be incorrect

Fourier Transform of Important Functions



1.7 Unitary Transform

Vectorization Interpret image as vector row-by-row to a column vector.

Linear Image Processing $\mathbf{g} = \mathbf{Hf}$

Image Collection $\mathbf{F} = [f_1, f_2, \ldots f_n]$

Autocorrelation Matrix $\mathbf{R}_{ff} = \frac{F\cdot F^\top}{N}$, its eigen vector with alrgest eigenvalue is direction of largest variance among pictures.

PCA (KL Transform) Given data $x\in A$, e.g. an image, we want to compress it to a lower-dimensional space B with a compressor $f: A\to B$ s.t. size$(f(x))\ll$ size(x). Minimize reconstruction error $E = \|x - f^{-1}(f(x))\|$ and maximize the variance of our encoding. Given N data samples $x_i\in\mathbb{R}^d$:

1. Normalize to remove brightness variations: $x_i' = {}^{x_i}\!/\!_{\|x_i\|}$
2. Center data by subtracting mean: $x_i'' = x_i' - \mu, \quad \mu = \frac{1}{N}\sum_{i=1}^N x_i'$
3. Compute covariance matrix: $\Sigma = \frac{1}{N-1}\cdot\sum(x_i'')\cdot(x_i'')^\top$
4. Compute eigendecomp. of Σ by solving $\Sigma e = \lambda e$ with e.g. SVG, i.e. $\Sigma = U\Lambda U^\top$
5. Define U_k as the first k eigenvectors of $\Sigma = [u_1,\ldots,u_k]$, dirs with largest variance (= eigenvalues)

6. $\text{PCA}(x_i) = U_k^\top (x_i' - \mu) = U_k^\top \cdot x_i''$

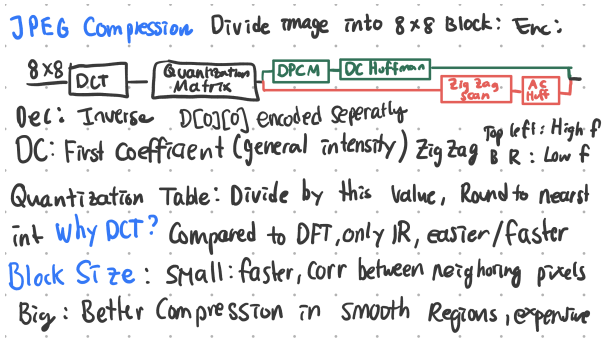
To decompress, use $\text{PCA}^{-1}(y_i) = U_k \cdot y_i + \mu$.

- Face recognition, compare in projected space and find nearest neighbour
- Face location, compute reconstruction error for every small patch, pick min
- Data compression and visualization

Eigenfaces struggle with different lighting conditions. Fisherfaces perform better by trying to maximize between-class scatter and minimizing within-class scatter.

Storage Let image size be $L \times W$, dimension K, number of pictures N. Then $\mu : L \times W$, $U_k : L \times W \times K$, compressed image size $N \times K$. Find K s.t. $NLW > LW + KLW + NK$

JPEG Compression



1. Conversion RGB → YUV; only Y carries brightness info (luminance), UV contain color (chrominance)
2. Humans are more sensitive to brightness than color, so compress colors with chroma subsampling (e.g. only color of upper left pixel for 4×4 grid).
3. Go over image with 8×8 block for each YUV component and apply 2D DCT to it → 64 vals, top left low.freq, bottom right high-freq.
4. Compress by integer division with weighting matrix → compress low-right
5. Zig-zag run length encoding followed by Huffman

High compression (10-100x) and 24 bit color depth is possible, but at the same time artifacts / wrong colors / Moiré. Also edges are softened because sharp edges require ∞ freq. JPEG2000 achieves better results by using the Haar transform globally, not just 8×8, on a successively downsampled image.

1.8 Pyramids and Wavelets

Scale Space Representations From an original signal $f(x)$, generate a parametric family of signals $f^t(x)$, where fine-scale information is successively suppressed.

Scaled Representation Applications Search for correspondence (look at coarse scales, then refine with finer scales), edge tracking (a good edge at fine scale has parents at a coarser scale).

Gaussian Pyramid Smooth with Gaussians $G \times G = G$. Synthesis: Smooth & Sample. Analysis: take top image.

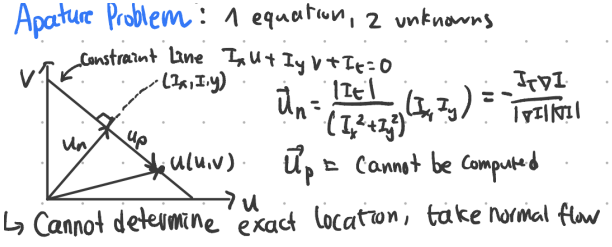
Laplacian Pyramid Synthesis: Preserve different between unsampled Gaussian Pyramid level. Band pass filter - each level represents spatial frequencies. (largely) unrepresented at other levels. - Compression. Analysis: Reconstruct Gaussian Pyramid, take top layer.

1.9 Optical Flow

Apparent motion of brightness patterns, use extracted feature points and compute their velocities at time t. Assumtions: Brightness Constancy, Small Motion, Spatial Coherence.

Brightness Constancy

$$\begin{aligned} I(x,y,t) &= I(x+\delta x, y+\delta y, t+\delta t) \\ &\approx I(x,y,t) + I_x\delta x + I_y\delta y + I_t\delta t \\ &\quad \text{(Taylor approx., } \textbf{good if small motion}\text{)} \\ \implies & I_x\delta x + I_y\delta y + I_t\delta t \approx 0 \\ \implies & I_x \underbrace{\frac{\delta x}{\delta t}}_{u} + I_y \underbrace{\frac{\delta y}{\delta t}}_{v} + I_t \approx 0 \end{aligned}$$



Horn & Schunck Assumption: values $u(x,y), v(x,y)$ are smooth and change slowly (x,y) Minimize $e_s + \lambda e_c$

$$e_c = \iint (I_x u + I_y v + I_t)^2 \, dx\, dy \qquad \text{(brightness const.)}$$

$$e_s = \iint (u_x^2 + u_y^2) + (v_x^2 + v_y^2) \, dx\, dy \qquad \text{(smoothness)}$$

Has errors at boundaries, information spreads from corner-type patterns

Lucas-Kanade Assume all neighbouring pixels in a patch W observer the same motion $[u,v]^\top$ (+ small movement, brightness constancy). Compute I_x, I_y, I_t and minimize

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

Solve least squares (sums are over patch W):

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$$

$(\sum \nabla I \nabla I^\top) u = -\sum \nabla I I_t$Fails if all gradients are in the same direction, e.g. for edges or smooth regions. However, it works well for corners and textured areas.

Iterative refinement Obtain more exact estimate of optical flow:

1. Estimate OF with Lucas-Kanade
2. Use estimated flow to warp image
3. Estimate OF using warped image
4. Repeat
5. Add up all estimates

Fails if intensity structure poor or large displacement.

Coarse-to-fine pyramids Create multiple levels by gradual subsampling of the image. Start with coarsest level, estimate OF. Gradually use aggregated OF estimate as initial estimate of the OF in the next finer level and estimate again with Lucas-Kanade. Iterate until finest level. This still fails if large lighting change happens.

Application of O.F. Video compression - make use of temporal redundancy and predict frames based on previously encoded frames, Video stabilization - estimate flow between frames and warp image using same flow over all pixels so that flow is close to 0.

1.10 Video Compression

Bloch's Law If stimulus duration ≤ 100 ms, we can exchange duration for brightness and vice-versa, e.g. if brightness of stimulus is halved, double the duration $\rightarrow$ can still be detected. This enforces > 10 Hz for videos

Interlaced Video Format 2 temporally shifted half images. Increase frequency $\rightarrow$ decrease spatial resoltuion.

Temporal Processing Takes advantage of similarity between successive frames. **Temporal Redundancy**: Spatial correlation betweeen neighboring pixels. Ineffective when many scene changes or high motion.

I-frame Intra coded, coded independently of all others.

P-frame Predictively coded, coded based on previous coded frames (based on previous I, P frames, can send motion vector + changes)

B-frame Bi-directionally predicted frame, coded based on both previous and future coded frames. (based on previous and following I, P frames, in case something is uncovered).

Motion Estimation Algo (ME)

1. Divide current frame into non-overlapping $N_1 \times N_2$ blocks
2. For each block, find the best matching block in reference frame (usually previous frame)

Best "match": MSE, MAE, Candidate Blocks: All blocks in e.g. (32x32) pixel area, Search Strategy: Full Search: Examine all candidate blocks Partial (fast) search: Examine a carefully selected subset

Motion Compensation Algo (MC) Use the best matching of reference frame as prediction of blocks in current frame *(gives motion vectors from ME to predict content of current from reference frame)*. **Motion Vector** Expresses relative horizontal and vertical offsets (mv_1, mv_2), or motion given a block from one frame to another (each block has its own MV).

Half-pixel ME : Coarse Step: Perform Integer ME on blocks, find best Int-pixel MV. Fine-step: Refine to find best half-pixel ME (by spatial interpolation and best-matching).

1.11 Convolutional Neural Networks

Given a kernel with size F, an image with size N, padding P and a stride S, the output dimensions of applying the filter to the image is $\frac{N+2P-F}{S}+1$. For stride 1 and padding $P = \frac{F-1}{2}$, the input dimension is thus preserved.

1.12 Radon Transform

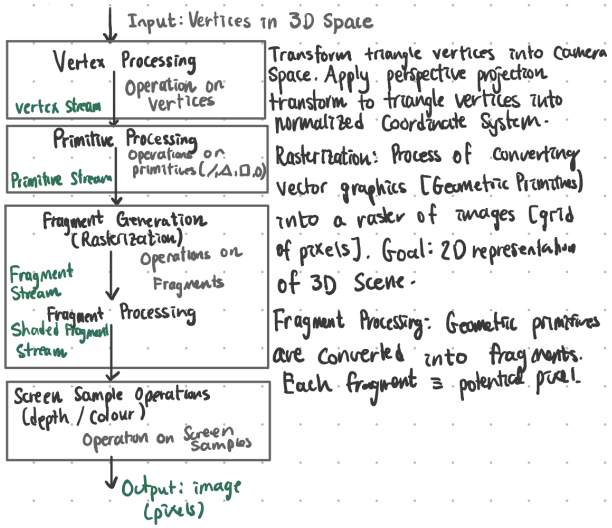
Given an object with unknown density $f(x, y)$, find f by sending rays from all directions through the object and measure absorption on the other side. We assume parallel beams for a given angle and no spreading of a beam. Continuous case: The radon transform of a line is

$$R(\rho, \theta) = \int u(\rho\cos(\theta) - s\sin(\theta), \rho\sin(\theta) + s\cos(\theta))\, ds$$
$$= \iint_{\mathbb{R}^2} u(x, y)\delta(p - x\cos(\theta) - y\sin(\theta))\, dx\, dy$$

We want to find the optical density $u(x, y)$ given $R(\rho, \theta)$.

2 Computer Graphics

2.1 Graphics Pipeline



2.2 Lights and Colours

Luminous Flux [Lumen] Perceived power of light $F = c\int_{380nm}^{780nm} P(\lambda)V(\lambda)d\lambda$ where $P(\lambda)$ is the relative spectral lambda density and $V(\lambda)$ is the relative spectral sensitivity: $c = 683\frac{lm}{W}$

Luminous Intensity [candela] $I = \frac{dF}{dw}$ so $F = \int I dw$

Luminous Y [candela^{-2}] $Y = \frac{d^2F}{dA\cos\epsilon dw}$ and $I = \int Y dA$ so $F = \iint Y dA dw$

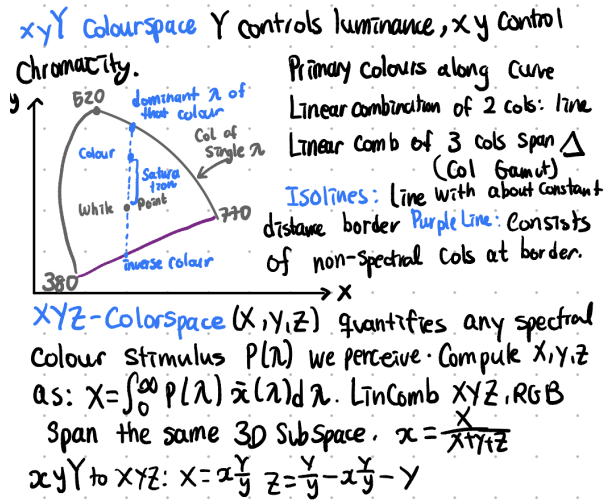
Illumination [lux] $B = \frac{dF}{dA}$

3 Cone Cell Types Short S, Medium M, Long L, S: Blue, M: Green, L: Red

Metamers Light with different spectrum can map to same colour $\rightarrow$ **CIE** observer mixes $\lambda_1, \lambda_2, \lambda_3$

Negative Matching Values Some colours cannot be combination of RGB, add red light to reference if impossible to match, leads to negative red values.

Colour Spaces



Chromaticity Colour, colourness

Luminance Brightness, Intensity

RGB Useful for monitors and displays

HSV Hue, Saturation, Value. Dimensions correspond to natural notions of "characteristics" of color. Useful for color picking

RGB to HSV
```
min=min(R,G,B);max=max(R,G,B)
V=max;s=(max-min)/max if max != 0 else 0;
H=Hue(V,S,R,G,B)
```

YIQ Y - luminance, I - in-phase (orange-blue), Q - quadrature (purple-green), based on psycho-physical properties of eye, good for skin colors, used for television

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.229 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

CMY(K) Subtractive color space, used for printing. To convert: $[CMY] = [111] - [RGB]$

LAB / LUV Perceptually correct distances, nonlinear warps of CIE chart such that MacAdams ellipses become approx. circular.

White point calibration

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} x_R C_R & x_G C_G & x_B C_B \\ y_R C_R & y_G C_G & y_B C_B \\ (1-x_R-y_R)C_R & (1-x_G-y_G)C_G & (1-x_B-y_B)C_B \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Set (R, G, B) to $(1, 1, 1)$. Map it to the given white point, e.g. $(0.9505, 1, 1.0890)$, then find C_R, C_G, C_B

2.3 Transformations

Homogeneous coordinates Allow representing translation by matrix multiplication, done by projection onto hyperplane: $p = \begin{bmatrix} x & y & z & w \end{bmatrix}^\top \to \begin{bmatrix} \frac{x}{w} & \frac{y}{w} & \frac{z}{w} & 1 \end{bmatrix}$

3D Transformations Examples:

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

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

Rot. (z) $\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ Shear (2D) $\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Shear (x) $\begin{bmatrix} 1 & 0 & \mathrm{sh}_x & 0 \\ 0 & 1 & \mathrm{sh}_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

If we have e.g. $\begin{bmatrix} -1 & 0 & 2 \\ 0 & -1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$, we first scale/rotate, then translate (follows from matrix multiplication. laws). Rotation that does not leave XY is around Z axis. Rot 2D counterclockwise similar to Z in 3D. If clockwise take -1 * sine values. **APPLY MATMUL FROM RIGHT TO LEFT!!!!**

Change of coord. system : $p' = \begin{bmatrix} | & | & | & | \\ r_1 & r_2 & r_2 & t \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{bmatrix} \implies p' =$

TRp Transforming a normal n: $n' = (M^{-1})^\top n$

Quaternions Alternative approach for rotation. Properties:

$$i^2 = j^2 = k^2 = -1, \quad ijk = -1, \quad ij = k, \quad ji = -k$$
$$jk = i, \quad kj = -i, \quad ki = j, \quad ik = -j$$
$$\|q\| = \sqrt{a^2 + b^2 + c^2 + d^2},$$

where $q = a + \begin{bmatrix} b & c & d \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} = a + v$,v not vector!

$$\overline{z} = a - bi - cj - dk, \quad z^{-1} = \frac{\overline{z}}{\|z\|}$$

Rotation with quaternions Rotate $p = \begin{bmatrix} x & y & z \end{bmatrix}$ around axis $u = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}$ by angle θ.

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

Orientation of coord. system Thumb: x-axis, index finger: y-axis, middle finger: z-axis. To get direction of cross product, use thumb & index finger as multiplicands and middle finger is direction. Why? Efficient implementation, easy interpolation, no gimble lock.

2.4 Lighting

Luminous Flux $\Phi(A) = \int P(\lambda)V(\lambda)\,d\lambda$
Perceived power of light, weighted with human sensitivity [lumen]

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

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

Radient/Luminous Intensity $I(\vec{w}) = {}^{d\Phi}/_{d\vec{w}}$
Outgoing flux per solid angle [W/sr]

Radiance/Luminance $L(x, \vec{w}) = {}^{d^2\Phi(A)}/_{\cos\theta dA(x) d\vec{w}}$
Flux per solid angle per perpendicular area = effective intensity per unit area

Lambert's Cosine Law Irradiance at surface is proportional to cosine of angle between light direction and surface normal: $E = \frac{\Phi}{A}\cos\theta$

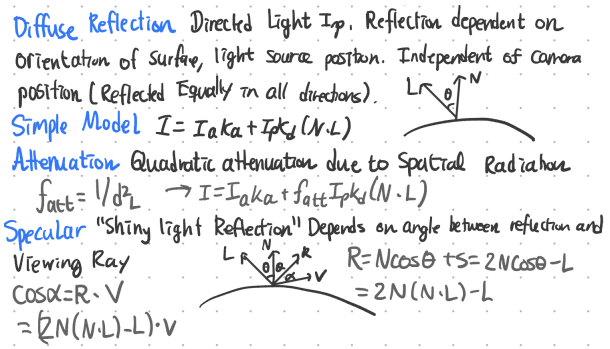
BRDF Bidirectional Reflectance Distribution Function encodes behavior of light that bounces off surface, given incoming direction w_i, hos much gets scattered in outgoing direction. sr^{-1}

$$f_r(x, w_i, w_r) = \frac{dL_r(x, w_r)}{dE_i(x, w_i)} = \frac{dL_r(x, w_r)}{L_i(x, w_i)\cos\theta_i dw_i}$$

Reflection Equation Can be derived from BRDF equation. Describes *local* illumination model. Reflected radiance due to incident illumination from all directions.

$$L_r(x, w_r) = \int_{H^2} f_r(x, w_i, w_r) L_i(x, w_i)\cos\theta_i dw_i$$

Ambient Light Scattered by enviroment coming from all directions, reflection independent of camera, light position, surface orientation.

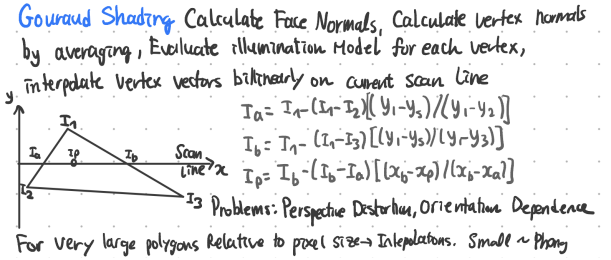


Phong Reflection / Illumination Model

$$I = \underbrace{I_a k_a}_{\text{ambient}} + f_{att} I_p [\underbrace{k_d (N \cdot L)}_{\text{diffuse}} + \underbrace{k_s (R \cdot V)^n}_{\text{specular}}]$$

$R = N \cdot \cos(\theta) + S = 2N(N \cdot L) - L$. Material parameters: k_a, k_d, k_s, n; light parameters: I_a, I_p; geometry parameters: N, L, V, R; attenuation due to spatial radiation: f_{att}. Big n leads to small highlights in terms of surface area

Flat shading One color per primitive, in screen space



Problems with scan line interpolations are perspective distortion, orientation dependence and shared vertices. Quality depends on primitive size

Phong shading Barycentric interpolation of vertex normals, in object space. Properties: $x = a \implies n_x = n_a$, $\lambda_a + \lambda_b + \lambda_c = 1$, $\lambda_a a + \lambda_b b + \lambda_c c = x$ Problem: The normal may not be defined or not representative.

Transparency, alpha blending Linearizes exponential attenuation of intensity $I_\lambda = I_{\lambda_1}\alpha_1\Delta t + I_{\lambda_2}e^{-\alpha_1\Delta t} \implies$ linearization $\implies I_\lambda = I_{\lambda_1}\alpha_1\Delta t + I_{\lambda_2}(1-\alpha_1\Delta t)$. If it's the last object, set $\Delta t = 1$.
Problem: Rendering order, we need sorted traversal of polygons → back-to-front rendering, Issue: When objects overlap, overlapping front not considered. Solution: Depth peeling, multiple passes, each pass renders the next closest fragment.

Differences Ambient light provides general, non-directional illumination. Diffuse light scatters in many directions, softly illuminating objects, emphasizing color and texture without harsh shadows. Specular light creates focused, shiny reflections, highlighting an object's glossy qualities.

2.5 Geometry & Textures

Explicit representations Point cloud, subdivision surface (define surface with primitives, use recursive algorithm for refining), polygon mesh. Can easily model complex shapes and sample points from it, but it can take lots of storage

Implicit representations Signed distance function, algebraic surface $(x(u,v), y(u,v), z(u,v))$, level set. Can easily test inside/outside, compact storage but sampling all points is expensive, complex shapes hard to model

Texture Mappings Goal: Map Texture (u,v)-coords. to geometry (x,y,z)-coords. Example of sphere mapping: $(u,v) \to (\sin(u)\sin(v), \cos(v), \cos(u)\sin(v))$. We want low distortion, a bijective mapping that is efficiently computable.

Light Maps Save computation power by precomputing static lighting and applying it to texture (can be dynamically adapted)

Environment Maps Mirror environment with imaginary sphere / cube for easier computation of reflective objects

Bump Maps Perturb normals to fake fine detail, store normal displacement in grayscale value. A **Normal map** directly stores perturbed normals as (r,g,b) color where $n' = (2r-1, 2g-1, 2b-1)^\top$. Limitations: No bumps on silhouette, no self-occlusions, no self-shadowing. In contrast, **displacement mapping** actually modifies geometry, but more complex & expensive

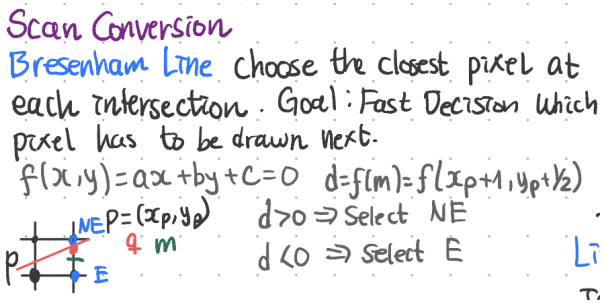
Procedural Textures Generate textures from noise, e.g. Perlin or Gabor noise by creating a gaussian pyramid of noise and summing all layers in a weighted way

Mip-Mapping Store down-sampled (blurred to avoid aliasing) versions of texture, use low-res versions for far away objects and interpolate inbetween. This avoids asliasing and improves compute efficiency but incurs a storage overhead ($\frac{1}{3}$ of the original texture storage)

Perspective Projection Linear variations in world coordinates can yield non-linear variation in screen coords → optimal resampling filter is spatially variant

Geometry aliasing Happens at polygon edges. Solution: Introduce multiple samples per pixel (supersampling). Different patterns possible: Uniform, jittering, stochastic, Poisson, ...

2.6 Scan Conversion

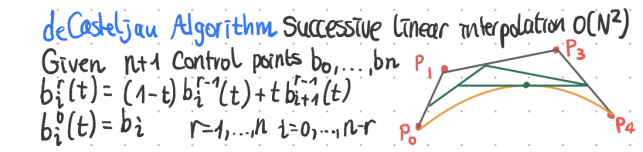


Scan Conversion of Polygons

1. Calculate all intersections on scan lines
2. Sort intersection points by ascending x-coordinates
3. Fill all spans between consecutive intersections if parity is odd

2.7 Curves & Splines

Bézier curves $x(t) = b_0 B_0^n(t) + \ldots + b_n B_n^n(t)$
$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$, if $i < 0 \vee i > 0 : B_i^n(t) = 0$
(reminder: $\binom{n}{i} = {}^{n!}/_{i!(n-i)!}$) The B_i^n's are the Bernstein polynomials. Coefficients b_i are control points. It can be constructed visually with the deCasteljau algorithm.



Properties of Bernstein Partition of Unity, positively, recursion, symmetry.

Properties of Bézier Curves Affine Invariance affine transform of all points on the curve is accomplished by affine transform of control points. **Convex Hull** curve lies within convex hull of its control polygon. **Design Property** Control Polygon gives a rough sketch of the curve. **End Point Interpolation** $B_0^n = B_n^n = 1$. **Variation diminishing property** The maximum number of intersections of a line with the curve is $\leq$ to the number of intersections with its control polygon. **Limitations** Global support of basis function, insertion of new control points come along degree evaluation. C^r-continuity between individual segments of Bézier curves.

Cubic Bézier $x(t) = b_0(1-t)^3 + 3b_1 t(1-t)^2 + 3b_2 t^2(1-t) + b_3 t^3$

B-splines $x(t) = \sum_{i=0}^k d_i N_i^n(t)$, basis function is recursively defined by convolution of box func.: $B^0(t) = 1$ if $t \in [-1,0] \implies B^1(t) = \int_{\mathbb{R}} B^0(s)B^0(t-s)\,ds$ und $B^n(t) = B^{n-1}(t) * B^0(t)$.

deBoor Points $s(u) = \sum_{i=1}^N d_i^N(u)$. Coefficients d_i of b-spline. Basis are piece-wise, recursively defined polynomial over sequence of knots $u_0 < u_1 < u_2 < \ldots$ defined by knot vector $T = U = [U_0, \ldots, U_{k+n+1}]$

B-Spline Properties Partition of unity $\sum_{i=1}^N N_i^n(u) = 1$, positivity, compact support, continuity C^{N-1}, N basis functions = N control points, bézier curves are special cases of b-splines.

Linear B-Spline Given two points $\mathbf{p}_0, \mathbf{p}_1$ as well as two knots t_0, t_1, then the convex combination of these two lines give

$$\mathbf{d}_0^1(t|\mathbf{p}_0,\mathbf{p}_1;t_0,t_1) = \frac{t_1-t}{t_1-t_0}\mathbf{p}_0 + \frac{t-t_0}{t_1-t_0}\mathbf{p}_1$$

Recurrence Relation

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

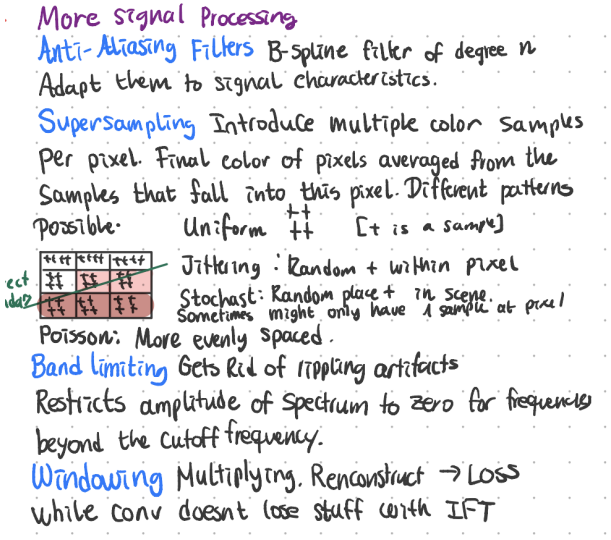
where $N_i^0 = (u) = 1$ if $u \in [u_i, u_{i+1}]$ else 0

De Boor Algorithm Control point on kth step $d_i^k = (1-a_i^k)d_{i-1}^{k-1} + a_i^k d_i^{k-1}$ and $a_i^k = \frac{t-u_i}{u_{i+n+1-k}-u_i}$ where $d_i^0 = d_i$ and $d_n^n = s(t)$

Tensor Product Surface

Subdivision Surfaces Generalization of spline curves/surfaces, converge to smooth limit surface, successive refinement. *Primal*: Faces are split into sub-faces, *Dual*: Vertices are split into multiple vertices *Approx.*: Control points are not interpolated *Interpol.*: Control points are interpolated

2.8 More Signal Processing



2.9 Visibility and Shadows

Visibility Problem Some parts of some surfaces are occluded

Solution 1: Painters Algorithm Render objects/polygons from furthest to nearest. Problems: Cyclic Overlaps, Intersections

Z(Depth)-buffering : Store depth to nearest object for each pixel. Algo: 1. Initialize all z values to infinity 2. For each polygon, if z value of a pixel for this polygon smaller than the stored z value. If larger, then its behind, otherwise replace the stored z value. Problem: Limited Resolution (from the camera, near: higher res, far: lower res), Z-fighting: when two or more objects are exactly or very closely positioned along the Z-axis (depth in 3D space)

Why Shadows? Depth cue, scene lighting, realism

Basic Shadows :

1. Planar: Draw project of the object on the ground, Limitations: Self shadows, shadows on other objects, curved surfaces.
2. Projective Texture Shadows: Separate obstacle and receiver. Compute b/w image of the obstacle from light. Use image as projective texture. Limitations: Need to specify obstacle and receiver, No self-shadows.

Shadow Maps Compute the depths from light, camera. Algo: For each pixel on the camera place

- Compute the point in world coordinates
- Project point onto the light plane
- Compare $d(\mathbf{x}_L)$ (shadow map) and z_L
- If $d(\mathbf{x}_L) < z_L$, $\mathbf{x}$ is in shadow

Limitations:

- Bias: For a visible point $d(\mathbf{x}_L) < z_L$, might get z-fighting again. Solution: $d(\mathbf{x}_L) + bias < z_L$ (tricky)
- Field of view: A point to shadow can be outside the field of view of shadow map. Use cubical shadow map or spot lights.
- Aliasing: Undersampling of shadow map.

Shadow Volumes Explicitly represent the volume of space in shadow, if the polygon is inside the volume, it is in the shadow (similar to clipping). Naive implementation $\mathcal{O}(\#\text{polygons} \times \#\text{lights})$ Algo:

- Shoot a ray from the eye
- Incre-/decrement a counter each time boundary of shadow volume is intersected.
- If counter > 0, primitive is in shadow, elif $= 0$: primitive not in shadow

Optimization: Use silhouette edges only (where a back-facing & front-facing polygon meet)

2.10 Ray Tracing

For each pixel, send a ray into the scene. On object hit, send multiple rays (diff., reflected, refracted) further until we hit a light source or reach some # of bounces. Figure out whether point in shadow by shooting rays to all light sources. For anti-aliasing, use multiple rays per pixel.

Ray-Surface Intersections Ray equation: $r(t) = o + td$ Sphere intersection: Solve for $t : \|o + td - c\|^2 - r^2 = 0$ Triangle: first intersect with triangles plane: $t = -\frac{(o+p_1)n}{d \cdot n}$, where $n = (p_2 - p_1) \times (p_3 - p_1)$. Then compute barycentric coeff. of intersection point and check whether $s_1 + s_2 + s_3 = 1 \wedge 0 \leq s_i \leq 1 \implies$ if yes, inside triangle

Ray Tracing Extensions

- Add refraction
- Add area lights by having multiple shadow rays
- Motion blur: render objects at diff. times per frame
- Depth of field

Acceleration Data Structures

Brute force Intersect every ray with every primitive

Uniform grids Determine sensible grid resolution, compute AABB's, incrementally rasterize ray, compute intersection with objects in each cell, stop when intersection found, if multiple take closest. (Easy to code, building data structure is fast, BUT does not adapt to non-uniform scenes ($\rightarrow$ hierarchical grids.

Space partitioning trees Octree, kd-tree, bsp-tree

OpenGL `projectionMatrix` transforms points from camera to screen space. `modelviewMatrix` transforms points from object to camera space. If `modelviewMatrix` is not a uniform transformation, then the original normals aren't perpendicular anymore. That's why we use a `normalMatrix` to transform the normals s.t. they remain perpendicular in the new space. `gl_Position` contains the coordinates of a vertex in 3D space in homogeneous form.