ETH Zürich
HS16

Frédéric Vogel

Visual Computing

12-929-246

Convolution and Filtering

Correlation: $I' = K \circ I$, $I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j) \cdot I(x+i, y+j)$

→ used for template matching (bias towards bright areas → remove mean)

→ $I'(x,y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} K(i,j)\, I(x+i, y+j)$

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

→ same as correlation with reversed kernel

→ if $K(i,j) = K(-i,-j)$ then $K \circ I = K * I$

→ associative!

→ $I'(x,y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} K(-i,-j)\, I(x+i, y+j)$

Near the edge:

→ filter window falls off image

→ extrapolation:

- ↦ clip filter
- ↦ wrap around
- ↦ copy edge
- ↦ reflect across edge

Gaussian Kernel:

→ Weight contribution of neighbouring pixels by nearness

→ $G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

→ Separable: $g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-x^2}{2\pi\sigma^2}\right] \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-y^2}{2\pi\sigma^2}\right]$

→ advantages:

- ↦ Rotationally symmetric
- ↦ Single lobe; neighbours influence decreases monotonically
- ↦ Still one lobe in frequency domain: No corruption from high freq.
- ↦ Simple relationship to σ
- ↦ Easy to implement efficiently

Morphological operators:

→ Fit: $\{y : y = x+s, s\in S\} \subset I$ $(\forall 1 \in S : \text{match})$

→ Hit: $\{y : y = x-s, s\in S\} \cap I \neq \emptyset$ $(\exists 1 \in S : \text{match})$

→ Miss: $\{y : y = x-s, s\in S\} \cap I = \emptyset$ $(\nexists 1 \in S : \text{match})$

→ Erosion: $E = I \ominus S = \{x : x+s \in I \text{ for every } s \in S\}$ $E(x) = \begin{cases} 1 \text{ if } S \text{ fits } I \text{ at } x \\ 0 \text{ otherwise} \end{cases}$

→ Dilation: $D = I \oplus S = \{x : x-s, y \in I \text{ and } s \in S\}$ $D(x) = \begin{cases} 1 \text{ if } S \text{ hits } I \text{ at } x \\ 0 \text{ otherwise} \end{cases}$

→ Opening: $I \circ S = (I \ominus S) \oplus S$

→ Closing: $I \bullet S = (I \oplus S) \ominus S$

Image Filtering:

Linear filtering: $g = f \otimes h$

→ $g(i,j) = \sum_{k,l} f(i+k, j+l)\, h(k,l)$

→ L is a linear operation if $L(\alpha I_1 + \beta I_2) = \alpha L(I_1) + \beta L(I_2)$

Example filters:

blur: $\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$

sharpen: $\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$

gaussian (low pass): $G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

differentials — Prewitt: $\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$ Sobel: $\begin{bmatrix}-1&0&1\\-2&0&1\\-1&0&1\end{bmatrix}$

high-pass — laplacian: $\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$ high-pass: $\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$

→ gradient vs laplacian: ↦ gradient is isotropic, laplacian not
↦ laplacian gives more noise (uses 2nd derivative)

→ filter is separable if $K(m,n) = f(m)\, g(n)$

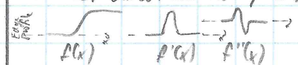
Image Features

Edge detection:

→ detect local gradient $|\text{grad}(f(x,y))| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

→ use differential filters in x and y direction

→ detect discontinuities by considering 2nd derivative



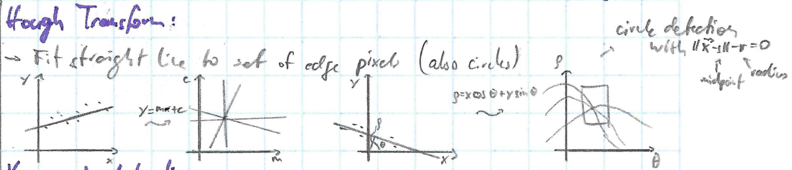
→ Sensitive to fine details and noise → blur first

Canny edge detector:

1. Smooth image with gaussian filter
2. Find magnitude and orientation of gradient (Prewitt, Sobel)
   $M(x,y) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$ $\alpha(x,y) = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right)$
3. Use non-maximal suppression on gradient magnitude image (thin edges)
4. Double thresholding to detect strong and weak edge pixels
   Strong: $M(x,y) \geq \Theta_{high}$, weak: $\Theta_{high} > M(x,y) \geq \Theta_{low}$
5. Reject weak edge pixels not connected with strong edge pixels

Hough Transform:

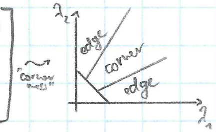
→ Fit straight line to set of edge pixel (also circles)



Keypoint detection

→ maximize eigenvalues of M in window w

$$M = \begin{bmatrix} \sum_{(x,y)\in w} f_x^2(x,y) & \sum_{(x,y)\in w} f_x(x,y) f_y(x,y) \\ \sum_{(x,y)\in w} f_x(x,y) f_y(x,y) & \sum_{(x,y)\in w} f_y^2(x,y) \end{bmatrix}$$



$C(x,y) = \det(M) - k \cdot (\text{trace}(M))^2$
$= \lambda_1 \lambda_2 - k \cdot (\lambda_1 + \lambda_2)$

→ local maximum of cornerness → key points

– using Gaussian weighting function to give more importance to central pixels

Harris corner detector

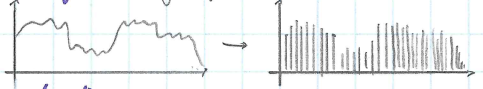
→ Invariant to brightness offset

→ Invariant to shift and rotation

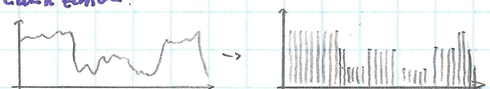
→ Not invariant to scaling

Computer Vision

Sampling: mapping signal to discrete representation



Quantization:



lossy compression
information is lost

Resolution:

→ geometric resolution: # pixels / area

→ radiometric resolution: # bits / pixel

Image Pyramid

→ Search for correspondence: look at coarse scales, then refine with finer scales

→ edge tracking: a "good" edge at a fine scale has parents at a coarser scale

→ control of detail and comp. cost in matching (finding stripes → texture repr.)

Gaussian: gaussian² = gaussian, lowpass filter → redundant representation

Laplacian: Δ of upsampled and normal gaussian pyramid level → band-pass filter
each level has freq. unrepresented at other levels.

Pyramid: repeated smoothing and down-sampling

ETH Zürich
HS16

Frédéric Vogel

Visual Computing

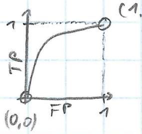
12-929-246

Image Segmentation

Tresholding: $B(x,y) = \begin{cases} 1 & \text{if } I(x,y) \geq T \\ 0 & \text{otherwise} \end{cases}$

Chromakeying: $I_\alpha = |I - g| > T$, $T = \sim 20$, e.g. $g = (0\ 255\ 0)$
↳ variation is NOT same in all 3 channels

ROC: characterizes performance of binary classifier



$TP = \frac{\text{True positives}}{\#\text{positives (P)}}$

$FP = \frac{\text{False positives}}{\#\text{negatives(N)}}$

$N = FP + TN$ $P = TP + FN$

~> choose the point on the ROC curve with gradient
$\beta = \frac{N}{P} \frac{V_{TN} + C_{FP}}{V_{TP} + C_{FN}}$, often $V_{TN} = V_{TP} = 0$ V: Value, C: Cost

Pixel connectivity:  : 4-neighbourhood
: 8-neighbourhood

Connected components labelling: labels each connected component of a binary image with different number

① 1. Iterate through each element.
2. If ¬background element
   i) get neighbouring elements
   ii) if ∄ neighbours → new unique label
      ↳ otherwise → assign label of smallest neighbour
   iii) store equivalence between neighbouring labels

② 1. Iterate through each element:
2. If the element is not background relabel with the lowest equivalent label

Fourier Transform

Aliasing: can't shrink image by taking every 2nd pixel

$$F(g)(u,v) = \iint_{\mathbb{R}^2} g(x,y)\, e^{-i 2\pi (ux+vy)}\, dx\, dy$$

$$F(g)(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x,y)\, e^{-i 2\pi \left(\frac{ux+vy}{N}\right)}$$

$e^{i\phi} = \cos\phi + i \sin\phi$

→ Phase: which basis how dominant, Magnitude: which basis how often

→ $\cos(iy) = \frac{e^{-y} + e^{y}}{2}$ $\cos(y) = \frac{e^{iy} + e^{-iy}}{2}$

→ $\sin(iy) = \frac{e^{-y} - e^{y}}{2i}$ $\sin(y) = \frac{e^{iy} - e^{-iy}}{2i}$

→ $\iint_{\mathbb{R}^2} \left(e^{-i2\pi(ax+by)} - e^{-i2\pi(ax+by)}\right) dx\, dy = \delta(a,b) - \delta(a,b)$

→ if image corrupted with $g(x,y)$ design band-pass at (u_0, v_0) to remove

→ $F \cdot G = \mathcal{F}(f ** g)$ and $F ** G = \mathcal{F}(f \cdot g)$ (Conv. theorem)

Nyquist sampling theorem: Sampling frequency must be at least twice the highest frequency $\omega_s \geq 2\omega$, otherwise band-limit before sampling with low-pass

Optical Flow

→ OF = apparent motion of brightness patterns, used for tracking, segmentation, stabilization, compression, ...

Brightness constancy: for small δt brightness doesn't change

→ $I(x, y, t-1) \approx I(x,y,t-1) + \frac{\partial I}{\partial x}\delta x + \underbrace{\frac{\partial I}{\partial y}}_{I_y}\delta y + \underbrace{\frac{\partial I}{\partial t}}_{I_t}\delta t + e$

→ with small δt: $I_x \frac{dx}{dt} + I_y \frac{dy}{dt} + I_t \overset{!}{\approx} 0$

Aperture problem: "The motion direction of a contour is ambiguous, because the motion component parallel to the line cannot be inferred based on visual input

Horn & Schunk: Introduce global smoothness to solve aperture problem

→ $e_s = \iint \left((u_x^2 + u_y^2) + (v_x^2 + v_y^2)\right) dx\, dy$
→ $e_c = \iint \left(I_x u + I_y v + I_t\right)^2 dx\, dy$
minimize $e_s + \lambda e_c$

Lucas-Kanade: Assume single velocity

$$M\vec{u} = \underbrace{\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x t \\ \sum I_y t \end{bmatrix}}_{A^T b}$$

→ solvable when: ↦ A^TA invertible
↦ A^TA not too small (λ_1, λ_2 not too small)
↦ A^TA well conditioned (λ_1/λ_2 not too large)
eigenvalues $\lambda_1 \geq \lambda_2$

→ Corners and textured areas OK

→ Problems: ↦ large motion ($>1_{px}$): Iterative refinement, Coarse-to-fine estimation, Exhaustive neighbourhood search (feat. match)
↦ point does not move like neighbours: motion segment.
↦ brightness constancy doesn't hold: Exhaustive neighbourhood search with normalized correl.

Coarse-to-fine: build pyramid and do HS or LK on each layer to get rid of distance constraint.

Parametric Motion Model: Global (nonlocal) motion model

→ 2D Models: Translation, Affine, Quadratic, Planar projective transform
→ 3D Models: Instantaneous camera motion models, Homography + epipole, Plane + Parallax
→ $E(\text{Transform}) = \sum \left[I(\text{Transform}) - I_0(x)\right]^2$ → least squares

→ Translation: $x + \begin{bmatrix} \delta x \\ \delta y \end{bmatrix}$ Affine: $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot x + \begin{bmatrix} \delta x \\ \delta y \end{bmatrix}$, $u = a_1 + a_2 x + a_3 y$, $v = a_4 + a_5 x + a_6 y$, $Err(\vec{a}) = \sum \left[I_x(u) + I_y(v) + I_t\right]^2$

→ Quadratic: instant approx to planar motion, $u = q_1 + q_2 x + q_3 y + q_7 x^2 + q_8 xy$, $v = q_4 + q_5 x + q_6 y + q_7 xy + q_8 y^2$

→ Projective: exact planar motion $x' = \frac{h_1 + h_2 x + h_3 y}{h_7 + h_8 x + h_9 y}$, $y' = \frac{h_4 + h_5 x + h_6 y}{h_7 + h_8 x + h_9 y}$ $u = x' - x$, $v = y' - y$

→ Instantaneous camera motion: $u = -xy\Omega_x + (1+x^2)\Omega_y - y\Omega_z + (T_x - T_z x)/Z$, $v = -(1+y^2)\Omega_x + xy\Omega_y - x\Omega_z + (T_y - T_z x)/Z$ $Z(x,y)$: local param, Ω_i, T: global param

→ Homography + Epipole: same as Projective with $x' = \frac{\cdots + \gamma t_1}{\cdots + \gamma t_3}$ and $y' = \frac{\cdots + \gamma t_1}{\cdots + \gamma t_3}$

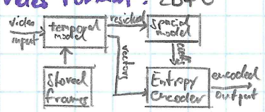
→ Plane + Parallax: $u = x^w - x = \frac{\gamma}{1 + \gamma t_3}(t_3 x - t_1)$, $v = y^w - y = \frac{\gamma}{1 + \gamma t_3}(t_3 y - t_2)$

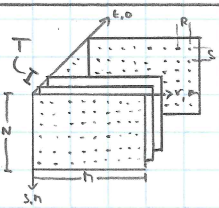
Correlation and SSD: For large displacements: do template matching

Bayesian Flow: Add uncertainty model to Lucas-Kanade tracking (Ω known)

Video Compression

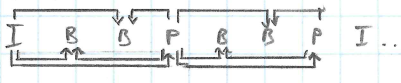
Video Format: 2D+t





Frames:

→ I-Frame: intra-coded, independent of all other frames
→ P-Frame: coded based on previously coded P or I frame
→ B-Frame: bi-directional predicted, based on previous and future coded frames



→ take advantage of temporal redundancy
→ predict current frame based on previously coded frames
→ Ineffective if many scene changes, high motion

Block-matching motion estimation:

→ Partition each frame into blocks, describe motion of each block
↳ assume same motion within block
+ no object identification required

→ ∀ blocks search for best matching block in reference frame (MAE, MSE) Mean absolute / squared Error
↳ Full Search: Examine all candidate blocks
↳ Partial search: Examine carefully selected subset

Summary: → RGB → YUV: small color differences not noticable, brightness more important
→ temporal redundancy: use MC and ME, drop B Frame
→ spacial redundancy: use DCT and get rid of high frequencies
quantisize

Visual Computing

Frédéric Vogel

12-529-236

General formulas

sin/cos table:

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

$\sin(x) = -\sin(x-180)$

$\cos(x) = -\cos(x-180)$

$\underbrace{}_{180° \leq x \leq 360°}$

skalar produkt:

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \sum_{i=1}^{n} a_i \cdot b_i \qquad \vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos(\gamma)$$

Vector produckt:

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} \qquad \vec{a} \times \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \sin(\gamma)$$

sin & cos:

$\sin(-\alpha) = -\sin(\alpha)$ $\qquad$ $\sin(2\alpha) = 2\sin(\alpha)\cos(\alpha)$

$\cos(\alpha) = \sin(90° - \alpha)$ $\qquad$ $\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$

$\cos^2\alpha + \sin^2\alpha = 1$ $\qquad$ $\cos(\alpha \pm \beta) = \cos\alpha\sin\beta \mp \sin\alpha\cos\beta$

$\cos(2\alpha) = \cos^2\alpha - \sin^2\alpha = 2\cos^2\alpha - 1 = 1 - 2\sin^2\alpha$

polar form

$z = r \cdot e^{i\varphi} = r(\cos\varphi + i\sin\varphi)$, $\quad x = r\cos\varphi,\ y = r\sin\varphi$

$r = \sqrt{x^2 + y^2}$

least squares

$A^T A x = A^T b \Rightarrow x = (A^T A)^{-1} A^T b$

Radon Transform

Mathematical formulation of taking projections of an object at different angles, i.e. take the line integral of a fraction at different angles and plot it.

$$R(f(x,y))(\theta, s) = \int f((t\sin\theta + s\cos\theta),(t\cos\theta + s\sin\theta))\,dt$$

The reconstruction of the original image from the radon transform happens through Fourier. Because Fourier transform of a radon slice corresponds to a strip in the Fourier transform of $f(x,y)$.

Textures

Chaos Mosaic:

1. tile input image
2. pick random blocks and place them in random location
3. smooth edges

$\Big\}$ doesn't work for structured textures

Efros & Leung:

→ to synthesize new pixel search input image for similar neighbourhood, pick match at random

→ faster: combine with blocks and synthesize blocks, not pixels

ETH Zürich
HS16

Frédéric Vogel

Visual Computing

12-929-246

Colour

Human perception:

- → S-cone : 420nm
- → M-cone : 530nm
- → L-cone : 560nm

} colour vision

- → Rod : 496nm } brightness vision
- → Metamer : actually different colours that look same to us

CIE Primary System:

- → three light sources: 435.8nm (blue), 546.1nm (green), 700.0nm (red)
- → find weights so that resulting color is same as given spectral light
- → if not possible, add red to spectral light → "negative" red value

CIE XYZ:

- → Y is luminance
- → Z is similar to blue (S-cone)
- → X is mix of cone response curves that are non-negative

} analogous to LMS cone response of human eye

CIE xyY:

- → Y is luminance
- → x, y define chromaticity

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

$X = x\frac{Y}{y}$, $Y = Y$, $Z = \frac{Y}{y} - x\frac{Y}{y} - Y$

CIE xyY Chromaticity diagram:

- → horseshoe → spectral colors, full saturation
- → line between ends of horseshoe: line of purples, not spectral
- → white point at $x = y = \frac{1}{3}$
- → saturation: distance to whitepoint / distance whitepoint - spectral color

CMY: subtractive, used in passive color system (printers) — non-pos. linear comb.

RGB: additive, used in light and monitors $\begin{bmatrix}R\\G\\B\end{bmatrix} + \begin{bmatrix}C\\M\\Y\end{bmatrix} = \begin{bmatrix}1\\1\\1\end{bmatrix}$ — non-neg. linear comb.

YIQ: based on psychophysical properties of human eye

- → Y: Luminance, I: In-Phase (orange-blue), Q: Quadrature (purple-green)
- → advantage for natural and skin colour, used in USTV NTSC

$$\begin{bmatrix}Y\\I\\Q\end{bmatrix} = \begin{pmatrix}0.299 & 0.587 & 0.114\\0.596 & -0.275 & -0.321\\0.212 & -0.528 & 0.311\end{pmatrix}\begin{bmatrix}R\\G\\B\end{bmatrix}$$

HSV/HSL: user-oriented color space, intuitive for color picking

- → H: Hue (base color), S: saturation, L: Luminance (brightness)
- V: Value (brightness)

McAdams Color Ellipses: region in chromaticity diagram where colours are indistinguishable for human. Gets almost round with CIELAB and CIELUV

Transformations

2D Transformations: $p = (x, y)^T$

- → Translation: $\begin{pmatrix}x'\\y'\end{pmatrix} = \begin{pmatrix}x\\y\end{pmatrix} + \begin{pmatrix}t_x\\t_y\end{pmatrix} \Rightarrow p' = p + t$
- → Scaling: $\begin{pmatrix}x'\\y'\end{pmatrix} = \begin{bmatrix}s_x & 0\\0 & s_y\end{bmatrix}\begin{pmatrix}x\\y\end{pmatrix} \Rightarrow p' = Sp$
- → Rotation θ: $\begin{pmatrix}x'\\y'\end{pmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta\\\sin\theta & \cos\theta\end{bmatrix}\begin{pmatrix}x\\y\end{pmatrix} \Rightarrow p' = Rp$

Homogeneous coordinates: $p = (x, y, 1)^T$ affine map is linear map

- → Translation: $\begin{pmatrix}x'\\y'\\1\end{pmatrix} = \begin{bmatrix}1 & 0 & t_x\\0 & 1 & t_y\\0 & 0 & 1\end{bmatrix}\begin{pmatrix}x\\y\\1\end{pmatrix} \Rightarrow p' = Tp$
- → Rotation: $R = \begin{bmatrix}R & 0\\0 & 1\end{bmatrix}$
- → Scaling: $S = \begin{bmatrix}S & 0\\0 & 1\end{bmatrix}$
- → Shear along x and y-axis

$SH_x = \begin{bmatrix}1 & a & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$, $SH_y = \begin{bmatrix}1 & 0 & 0\\b & 1 & 0\\0 & 0 & 1\end{bmatrix}$

Combination of 2D Transformations:

- → Combine via matrix multiplication
- → Commutativity $M_1 M_2 = M_2 M_1$
  - ↳ Translation – Translation
  - ↳ Rotation – Rotation
  - ↳ Scaling – Scaling
  - ↳ Scaling – Rotation

3D Transformations: $p = (x, y, z, w)^T$

→ Translation
$$\begin{bmatrix}1 & 0 & 0 & t_x\\0 & 1 & 0 & t_y\\0 & 0 & 1 & t_z\\0 & 0 & 0 & 1\end{bmatrix}$$

→ Scaling
$$\begin{bmatrix}s_x & 0 & 0 & 0\\0 & s_y & 0 & 0\\0 & 0 & s_z & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

→ Rotation around x-, y-, z-axis by angle θ

$$\underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\0 & \cos\theta & -\sin\theta & 0\\0 & \sin\theta & \cos\theta & 0\\0 & 0 & 0 & 1\end{bmatrix}}_{x} \quad \underbrace{\begin{bmatrix}\cos\theta & 0 & \sin\theta & 0\\0 & 1 & 0 & 0\\-\sin\theta & 0 & \cos\theta & 0\\0 & 0 & 0 & 1\end{bmatrix}}_{y} \quad \underbrace{\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\\sin\theta & \cos\theta & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}}_{z}$$

→ Shearing parallel to principal planes

$$\underbrace{\begin{bmatrix}1 & 0 & sh_x & 0\\0 & 1 & sh_y & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}}_{SH_{xy}} \quad \underbrace{\begin{bmatrix}1 & sh_x & 0 & 0\\0 & 1 & 0 & 0\\0 & sh_z & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}}_{SH_{xz}} \quad \underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\sh_y & 1 & 0 & 0\\sh_z & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}}_{SH_{yz}}$$

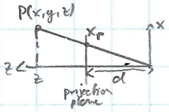
Change of coordinate system:

→ $p' = \begin{bmatrix}r_1 & r_2 & r_3 & t\\0 & 0 & 0 & 1\end{bmatrix} \cdot [p_x, p_y, p_z, 1]^T = TRp$

r_i: i-th new base vector; $r_{1,2,3}$: Rotation; t: Translation

Projection:

perspective



$x_p = dx/z$ $\quad y_p = dy/z$

$$M_{persp} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 1/d & 0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix} = \begin{bmatrix}x\\y\\z\\z/d\end{bmatrix} \rightarrow \begin{bmatrix}\frac{dx}{z}\\\frac{dy}{z}\\d\end{bmatrix}$$

3D-coordinate

divide by last component ⇝ 2D

parallel

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

→ z-buffer value is like d

The Graphics Pipeline

Inputs: Geometry, Materials & Lighting, Virtual Camera

Output: Colors for Display

① Modelling Transform → ② Viewing Transform → ③ 3D Clipping → ④ Lighting Shading Texturing → ⑤ Projection to Screen Space → ⑥ Scan Conversion

1. ① From object to world space
2. ② From world space to camera space
3. ③ Remove parts outside of frustum (avoid unnecessary computations, numerical instabilities)
4. ④ Compute color based on lighting, materials, surface normals
5. ⑤ Project from 3D to 2D space
6. ⑥ Compute fragment attributes, interpolate per-vertex values

CPU → Vertex Processing → Rasterization → Fragment Processing → Display

programmable

Vertex Processing: Per-vertex operations, Transform & Lighting, Flow control

Fragment Processing: Per-fragment operations, shading & Texture, Blending

OpenGL Transformations:

Vertex $(x, y, z, 1)^T$ → Model View Transform → Eye Coordinates → Projection → Clip Coordinates → Perspective Division → Normalized Device Coordinates → Viewport Transform → Window (Screen) Coordinates

Model View Transform: Model → World, World → Camera coordinates

Projection: Parallel or Perspective

Perspective Division:
$$\begin{bmatrix}d_x\\d_y\\d_z\end{bmatrix} = \begin{bmatrix}c'_x/-c_z\\c'_y/-c_z\\c'_z/-c_z\end{bmatrix}$$
d_x, d_y: determines coordinates on screen; d_z: used for depth tests

Viewport Transform: From normalized to screen coordinates

Visual Computing

Frédéric Vogel
12-929-246

Quaternions

- $q = a + ib + jc + kd \quad i^2 = j^2 = k^2 = -1$
  - $ij = k \quad ji = -k \quad \|q\| = \sqrt{a^2+b^2+c^2+d^2}$ magnitude
  - $jk = i \quad kj = -i \quad \frac{q}{\|q\|}$ unit form
  - $ki = j \quad ik = -j$
  - $ijk = -1$
- $z = s + ix + jy + kz = \underset{\text{scalar}}{s} + \underset{\text{vector}}{\vec{v}}$
  - $z_1 = s_1 + v_1,\ z_2 = s_2 + v_2 \mid z_1 \cdot z_2 = s_1 \cdot s_2 - v_1 v_2 + s_1 v_2 + s_2 v_1 + v_1 \times v_2$
  - $z_1 z_2 \neq z_2 z_1,\ \bar{z} = s - v,\ z\bar{z} = \|z\|^2,\ z^{-1} = \frac{\bar{z}}{\|z\|^2},\ 1 = zz^{-1} = z^{-1}z$
- $p = (x, y, z)^T \to$ Quaternion $p = 0 + ix + jy + kz$
  - Rotation: $q = \cos(\theta/2) + \sin(\theta/2)u$ ← axis of rotation, $p' = qpq^{-1}$
  - $q^{-1} = \cos(\theta/2) - \sin(\theta/2)u$ ← normalisiert und als quaternion
- Why?
  - → efficient implementation ↦ Translation simple addition
  - ↦ Easy interpolation ↦ Rotation super easy
  - ↦ No Gimbal lock

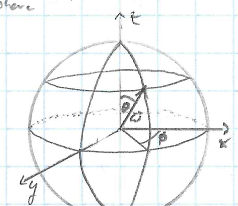
Radiometry

Basics:

- Angle in 2D: $\theta = \frac{l}{r}$
  - circle: 2π radians
- Solid Angle in 3D: $\Omega = \frac{A}{r^2}$ ← surface patch on sphere
  - sphere: 4π steradians

Direction:

- point on unit sphere
- parametrized by two angles:
  - $\vec{\omega} = (\theta, \phi)$ (zenith, azimuth)



$\vec{\omega}_x = \sin\theta\cos\phi \quad \vec{\omega}_y = \sin\theta\sin\phi,\ \vec{\omega}_z = \cos\theta$

latitude $= \frac{90}{\pi}(\pi - \theta)$ longitude $= \frac{90}{\pi}\phi$

Flux: total amount of energy passing through surface per unit time $\Phi(A)\ [W]$

Irradiance: Flux per unit area arriving at surface
$E(x) = \frac{d\Phi(A)}{dA(x)} \left[\frac{W}{m^2}\right]$ $\quad E = \frac{\Phi}{A}\cos\theta$ (Lambert's Cosine Law)

Radiosity: $B(x) = \frac{d\Phi(A)}{dA(x)} \left[\frac{W}{m^2}\right]$ → leaving surface (reflection)

Radiant intensity: Flux per solid angle, directional density
$I(\vec{\omega}) = \frac{d\Phi}{d\vec{\omega}} \left[\frac{W}{sr}\right]$

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

→ most fundamental for raytracing, constant along ray

Reflection

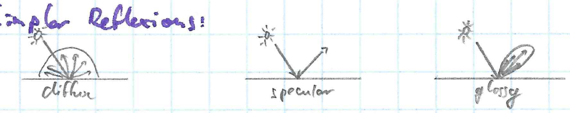
BRDF: bidirectional reflectance distribution function
→ $f_r(x, \vec{\omega}_i, \vec{\omega}_r) = \frac{dL_r(x, \vec{\omega}_r)}{dE_i(x, \vec{\omega}_i)} = \frac{dL_r(x, \vec{\omega}_r)}{L_i(x, \vec{\omega}_i)\cos\theta_i\, d\vec{\omega}_i} \left[\frac{1}{sr}\right]$

Differential irradiance due to a cone of direction around $\vec{\omega}_i$

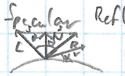
Reflection Equation: derived from BRDF
→ $L_r(x, \vec{\omega}_r) = \int_{H^2} f_r(x, \vec{\omega}_i, \vec{\omega}_r) L_i(x, \vec{\omega}_i) \cos\theta_i\, d\vec{\omega}_i$
→ local illumination model, reflected radiance

Complex Reflections:
→ layered materials, anisotropic reflections, subsurface scattering, volumetric structure

Simpler Reflections:



- Ambient light: scattered by environment, coming from all directions
  - ↳ reflection independent of: Camera and light position, surface orientation
  - ↳ reflected intensity: $I = I_a k_a$, I_a = light source, k_a mat param
- Diffuse Reflection: directed light I_p, dependent on: surface orientation, light source position
  - ↳ independent of: camera position (reflects equally in all directions)
  - ↳ reflected intensity: $I = I_p k_d \cos\theta = I_p k_d (N \cdot L)$ (dot product, N: Normal, L: vector to src)
- Simple Model: $I = I_a k_a + I_p k_d (N \cdot L) \cdot f_{att}$
  - Quadratic attenuation due to spacial radiation $f_{att} = \frac{1}{d_L^2}$
- Specular Reflection: depends on angle between reflection and viewing ray



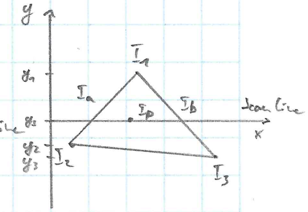
$R = N\cos\theta + S = 2N\cos\theta - L = 2N(N \cdot L) - L$
$\cos\alpha = R \cdot V = (2N(N \cdot L) - L) \cdot V$

Phong Illumination Model: Approx specular reflection by cosine powers
→ $I_\lambda = I_{a_\lambda} k_a O_{d_\lambda} + f_{att} I_{p_\lambda} [k_d O_{d_\lambda} (N \cdot L) + k_s (R \cdot V)^n]$

Shading

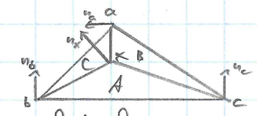
Flat Shading: One color per primitive (in screen space)

Gouraud Shading: linear interpolation of vertex intensities (in screen space)
1. Calculate face normals
2. Calculate vertex normals by averaging
3. Eval. illumination model for each vertex
4. Interpolate vertex color bilinearly on current line



- $I_a = I_1 - (I_1 - I_2)\frac{(y_1 - y_s)}{(y_1 - y_2)}$
- $I_b = I_1 - (I_1 - I_3)\frac{(y_1 - y_s)}{(y_1 - y_3)}$
- $I_p = I_b - (I_b - I_a)\frac{(x_b - x_p)}{(x_b - x_a)}$
- → Problems: perspective distortion, orientation dependence, shared vertices

Phong shading:



- $n_x = \lambda_a n_a + \lambda_b n_b + \lambda_c n_c$
- $\lambda_a = \frac{A}{\mathrm{all}},\ \lambda_b = \frac{B}{\mathrm{all}},\ \lambda_c = \frac{C}{\mathrm{all}}$
- $1 = \lambda_a + \lambda_b + \lambda_c$, $x = a \Rightarrow n_x = n_a$, $\lambda_a \cdot a + \lambda_b \cdot b + \lambda_c \cdot c = x$
- → Color of x determined by the interpolated normal
- → Problem: normal not defined / representative

Transparency: Alpha channel: $I_\lambda = I_{\lambda_1}\alpha_1 + I_{\lambda_2}(1 - \alpha_1)$
- Problem: rendering order
  - ↳ Back 2 Front: simple, ↯ overlap or crossing
  - ↳ Depth Peeling: multiple passes, each pass renders next closest fragment

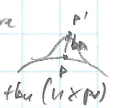
Texture Filtering

Aliasing: use low-pass filter to avoid aliasing by reducing high freqs
→ in screen space: $f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$ isotropic 2D Gaussian

Light Map: simulates effect of a local light source by an alpha mask, can be precomputed and adapted dynamically

Environment Map: Rendering reflective objects effectively
Texture coordinates are computed by intersecting reflected ray with surrounding sphere / box

Bump Map: perturb surface normal according to texture
→ $n = \frac{\partial p}{\partial u} \times \frac{\partial p}{\partial v} = p_u \times p_v$ (parameter domain), $p' = p + \frac{b n}{|n|}$, $n' = \frac{\partial p'}{\partial u} \times \frac{\partial p'}{\partial v}$ $= n + b_v (n \times p_u) + b_u (n \times p_v)$

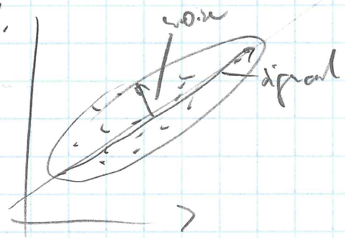


→ no bump on silhouette, no self-shading, no self-occlusion

Mid Map: Downsample texture for far away objects
→ improved efficiency, avoids aliasing

PCA

1.



- Calculate mean for each column
- Substract mean vector from each row in X
- Covariance Matrix $C = \frac{1}{n-1} B^T \cdot B$
- Eigenvalues and Eigenvectors of C
- Project data using truncated version of EC

PY

lin. $v(t) = \frac{d}{dt} x(t)$

sin. $\omega(t) = \dot{R}(t) = \omega(t) \cdot R(t) \rightarrow$

gen. $\dot{v}(t) = \omega(t) \otimes r(t) \otimes v(t)$

$\omega(t) \otimes x'(t)$
$\omega(t) \otimes y'(t)$
$\omega(t) \otimes z'(t)$

Rigid body

$\dot{x}(t) = \frac{P(t)}{m}$

$\dot{P}(t) = F_i(t)$

ODEs

$\dot{q}(t) = \frac{1}{2}(I^{-1}(t) L(t)) q(t)$

$\dot{L}(t) = (r_i(t) - x(t)) \otimes F_i(t)$

Governing equation

$$\text{State } X(x(t), R(t), P(t), L(t))^T, \frac{d}{dt} X(t) = \begin{pmatrix} v(t) \\ \omega(t) \cdot R(t) \\ F(t) \\ \tau(t) \end{pmatrix}$$

Geometry

Circle : $p: \mathbb{R} \to \mathbb{R}^2 \quad t \mapsto p(t) = (x(t), y(t)) \qquad p(t) = r(\cos(t), \sin(t))$

Bezier : $s(t) = \sum_{i=0}^{n} p_i B_i^n(t) \qquad B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-1}$

Space Curves: $f: X \to Y \quad X \subseteq \mathbb{R}^{n=1} \quad Y \subseteq \mathbb{R}^{n=3}$

$$s(t) = (x(t), y(t), z(t))$$

Normal + Tangent Plane: $s_u = \frac{\partial s(u,v)}{\partial u} \qquad s_v = \frac{\partial s(u,v)}{\partial v} \qquad n = \frac{s_u \otimes s_v}{\| s_u \otimes s_v \|}$