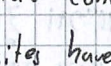
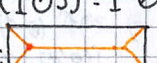
① Image Processing

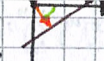
1: The Digital Image

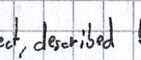
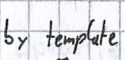
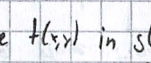
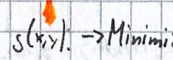
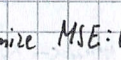
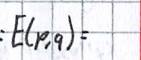
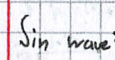
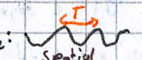
1.1) Image: Image is a 2D signal (function depending on some variable with physical meaning) where the input variables are xy-coord. (+ time for videos) and the function value is usually the brightness (or temperature, depth, ...): $f: \mathbb{R}^n \to S$, $n=2$ and $S=\mathbb{R}^t$ for greyscale images. **1.2) Pixel:** A pixel is a sample of an original image (not a little square) **1.3) Digital Camera:** A Charge Coupled Device (CCD) with a sensor array (array of photosites that contain charge proportional to light intensity during exposure), analog to digital converter (measures + digitizes result, happens line by line) **Blooming:** Photosites have finite capacity → High intensity leads to „bright" lines" around bright objects:  **Bleeding/Smearing:** During transit, the photones can still accumulate charge. In contrast to Blooming, effect until image edge but not as sharp/lit. **Dark Current:** Small current when no photons are present → can lead to noise for dark images. **1.4) CMOS:** Each sensor has own amplifier + ADC (+ possibly other components → „Smart pixels". Allows random pixel access and cheaper, but less sensitive **Rolling Shutter:** Lines are activated sequentially → Can lead to artifacts with moving objects. **1.5) Aliasing:** We store a functions value at many points (sampling) and reconstruct the function. If undersampled → indistinguishable from other frequencies. **Nyquist Frequency:** A signal's max. frequency must be smaller than half the sampling frequency of a discrete ~~sampling~~ signal processing system to allow exact reconstruction. **1.6) Quantization:** Real valued function will get digital values which is lossy (in contrast to sampling which can be (lossless). **1.7) Image Resolution:** No. of pixels (e.g. 1024×1024) **1.8) Geometric Resolution:** How many pixels per area **1.9) Radiometric Resolution:** How many bits per pixel **1.10) Usual Quantization Intervals:** Grayscale: 8 bit (256 values), RGB: 8 bit/channel (2^{24} colors) **1.11) Noise:** Common Model: Additive Gaussian Noise. $I(x,y)=f(x,y)+\varepsilon$, with $\varepsilon \sim N(0,\sigma^2)$ **1.12) Signal to Noise Ratio:** Index of image quality: $s=\frac{F}{\sigma}$, $F=\frac{1}{xy}\sum_{x=1}^{x}\sum_{y=1}^{y} f(x,y)$ Peak SNR: $\frac{F_{max}}{\sigma}$ **1.13) Color Camera: Prism:** Light is separated into RGB using a prism. **Filter Mosaic:** Color filter on sensor → demosaicing needed **Filter Wheel:** Multiple filters are rotated in front of camera.

2: Image Segmentation

Image segmentation partitions an image into regions of interest. A complete segmentation is a set of regions $R_1,\dots,R_n$ s.t.: $I=\bigcup_{i=1}^{N} R_i$ and $R_i \cap R_j=\emptyset$ $\forall i \neq j$ **2.1) Thresholding:** Simple seg. process that produces binary image. $B(x,y)=1$ if $I(x,y) \geq T$. **2.2) Chromakeying:** Controlled background („greenscreen"), then e.g. $I_\alpha=1$ if $\|I-g\|>T$, $g=[0,255,0]$. Problems: Variation can be different per channel / may not want a hard alpha mask: $I_{comp}=I_\alpha I_{fg}+(1-I_\alpha)I_{bg}$. **2.3) Performance:** True class. needs to be known (by hand). Binary class. has TPs (correctly „in"), TNs (correctly „out"), FPs (wrongly „in") and FNs (wrongly „out") **ROC curve:** Plots sensitivity $\left(\frac{TP}{P}\right)$ against 1-specificity $\left(\frac{FP}{N}\right)$:  $P=TP+FN$, $N=FP+TN$. If $V_{TN}, V_{TP}, C_{FN}, C_{FP}$ are values/costs, choose point on ROC with gradient $\beta=\frac{N}{P}\frac{V_{TN}+C_{FP}}{V_{TP}+C_{FN}}$ **2.4) Pixel Connectivity:** 4-neighborhood 8-neighborhood: **Connected Components Labeling:** Labels each component of a binary image with a separate number **Foreground Labeling:** Extract connected components of foreground **Region Growing:** Start from seed(point)/region, add neighbor pixel that satisfy criteria (e.g. threshold) until possible. **2.5) Foreground-Background Segmentation:** $I_\alpha=\|I-I_{bg}\|>T$, where I_{bg} is e.g. learned by fitting a Gaussian per pixel **2.6) Morphological Operators:** Local pixel transformations based on pattern comparison. **8-Neighbor Erode/Minkowsky Subtraction:** Foreground pixels with 8-connected bg pixel are removed **8-Neighbor Dilute/Minkowsky Addition:** BG pixels with 8-connected fg is painted. Smoothes region boundaries. **Structuring Elements:** Compared to pixel neighborhood → determines output. Binary array with origin (0,0) and rel. coord, e.g. (-1,0). A binary image $B \Rightarrow$ sets containing pixels with values 1, e.g. $I=\{(1,1),(2,1),\dots\}$ S **fits** I at x if $\{y: y=x+s, s \in S\} \subset I$ (S is 1 $\Leftrightarrow$ corresponding pixel in I is 1). S **misses** I at x if $\{y: y=x+s, s \in S\} \cap I=\emptyset$ (S is 1 $\Leftrightarrow$ corresponding pixel in I is 0). S **hits** I at x if $\{y: y=x+s, s \in S\} \cap I \neq \emptyset$ (matching for at least one pixel of S) **Dilution:** $D=I \oplus S$. $D(x)=1$ if S hits I at x. **Erosion:** $E=I \ominus S$. $E(x)=1$ if S fits I at x **Opening:** $I \circ S=(I \ominus S) \oplus S$ **Closing:** $I \cdot S=(I \oplus S) \ominus S$ **Hit-and-Miss Transform:** $H=I \otimes S$. Exact match of S, S can have x („don't care"). E.g. corner detector $\begin{smallmatrix} x & 0 & 0 \\ 1 & 1 & 0 \\ x & 1 & x \end{smallmatrix}$ **Thinning:** $I \ominus S=I \setminus (I \otimes S)$ **Thickening:** $I \odot S=I \cup (I \otimes S)$ $[(I \otimes S)\ [(I \odot S)^c=I^c \ominus S]$ [2.31] **Skeleton/Medial Axis Transform:** Union of centres of maximal discs within a region:  Calculated: $B=\begin{smallmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{smallmatrix}$ $S_n(X)=(X \ominus_n B) \setminus [(X \ominus_n B) \circ B]$. Skeleton $S(X)=\bigcup_{n=1}^{\infty} S_n(x)$

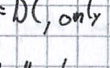
(2) **3: Convolution / Filtering** 3.1) **Filtering:** Modifying pixels based on function of local neighborhood. **Low Pass:** Attenuates ("abschwächen") high frequencies. Spatially → smoothing. **High Pass:** Attenuates low freq. Edge enhancement / detection, suppression of constant gray areas. **Band Pass:** Attenuates very high/low freq. Edge enhancement + noise reduction. Convolution of high pass/low pass. 3.2) **Linear Shift-Invariant Filter:** Linear combination of neighbors, doing same for all pixels (shift-invariant). [Linear if $k[\alpha I_1 + \beta I_2] = \alpha k[I_1] + \beta k[I_2]$. $I'(x,y) = \sum_{(i,j)\in N(x,y)} k(x,y,i,j)$. If k independent (x,y) → shift-invariant. 3.3) **Correlation:** $I' = k \otimes I$ $I'(x,y) = \sum_{j=-K}^{K}\sum_{i=-K}^{K} K(i,j) I(x+i, y+j)$ [template matching] 3.4) **Convolution:** $I'(x,y) = \sum_{j=-K}^{K}\sum_{i=-K}^{K} K(i,j) I(x-i, y-j)$ [point spread function, K is what happens with single pixel]. If $K(i,j) = K(-i,-j) \Rightarrow$ same as correlation. **LP filters:** Blur: $\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ Sharpening: $\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ Weighted smoothing: $\frac{1}{10}\begin{bmatrix}1&1&1\\1&2&1\\1&1&1\end{bmatrix}$ or $\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$ Gaussian (smoothing): $\frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$ (advantages: Rotationally symmetric, single lobe in both domains, simple relationship to σ, easy to implement) **HP filters:** Laplacian op. (2nd derivative): $\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$, example $\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$ 3.5) **Separable Kernels:** Can be written as: $K(m,n) = f(m)f(n)$ [convolution on intermediate result → N^2 to $2N$ complexity] 3.6) **Scale Space:** Convolution of Gaussian with itself is Gaussian with sd. $\sigma\sqrt{2}$. Repeated convolution with Gaussian → Scale space of image. 3.7) **Differential Filters:** Differential in x-direction $\frac{\partial f}{\partial x}$ approx. by $[-1\ 1]$. Prewitt: $\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$ Sobel: $\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}$ (in x-dir., y-dir. by rotating 90°) Used for edge detection. 3.8) **Image Sharpening:** $I' = I + \alpha(k * I)$, where k is a HP filter. **4: Image Features:** 4.1) **Template Matching:** Locate object, described by template $t(x,y)$ in $s(x,y)$ → Minimize MSE: $E(p,q) = \sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty}[s(x,y) - t(x-p, y-q)]^2 = \sum\sum |s(x,y)|^2 + \sum\sum |t(x,y)|^2 - 2\sum\sum s(x,y)\cdot t(x-p, y-q)$ → equiv. to max. „area correlation" = convolution of $s(x,y)$ with $t(-x,-y)$ 4.2) **Edge detection:** Detect local grad. with finite differences: $\begin{bmatrix}-1&1\end{bmatrix}$, $\begin{bmatrix}-1\\0\\1\end{bmatrix}$, Prewitt, Sobel or Roberts: $\begin{bmatrix}0&1\\-1&0\end{bmatrix}$ $\begin{bmatrix}1&0\\0&-1\end{bmatrix}$. Alternative: Zero-crossings of Laplacian. → very sensitive to noise → blur image first (combined in „Laplacian of Gaussian" operator). **Canny Edge Detector:** 1.) Smoothing with Gaussian 2.) Compute gradient magnitude/angle: $M(x,y) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$, $\alpha(x,y) = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$ 3.) Nonmaxima suppression to grad. magnitude image (edge normal quantized to horizontal, −45°, vertical, +45°; if $M(x,y)$ smaller than any of the two neighbors → suppress) 4.) Double thresholding (strong edge if $M(x,y) \geq \Theta_{high}$, weak if $\Theta_{high} \geq M(x,y) \geq \Theta_{low}$) 5.) Reject weak edges not connected to strong ones. 4.3) **Hough Transform:** 1.) Subdivide (m,c) or (p,θ) plane into bins 2.) Draw line [sinusoid for (p,θ)] in (m,c) / (p,θ) space for each edge pixels and increment bins 3.) Detect peaks in plane (p = distance from origin, θ = angle)



4.4) **Corner Detection:** Edges only well localized in one direction → use corners. Displacement sensitivity approximated by $\Delta^T M \Delta$, $\|\Delta\| = 1$, M („structure matrix") $= \begin{bmatrix}\sum f_x(x,y)^2 & \sum f_x(x,y) f_y(x,y)\\ \sum f_x(x,y) f_y(x,y) & \sum f_y(x,y)^2\end{bmatrix}$, $\sum = \sum_{(x,y)\in window}$, $f_x = \frac{\partial f}{\partial x}$. Want find (x,y) where M has large eigenvalues → $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$ = edge, $\lambda_1 = \lambda_2$ and small = „flat region". Alternative: „Cornerness" measure (Harris corner detector) $C(x,y) = \det(M) - k\cdot(\mathrm{trace}(M))^2 = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2$, but not scaling-invariant. **Desired Properties:** Accurate localization, invariance against shift/rotation/~~translation~~/scale/~~rotation~~/brightness change, robust against noise + high repeatability. **SIFT:** Feature recovery with position, orientation and scale. Position: Local maxima of DoG (difference of Gaussian) filter with strong response. Scale: Strong DoH responses in scale space. Orientation: Gradient directions in patch are calculated, most seen direction is „canonical orientation" of feature. SIFT descriptor: Used for matching, patch is divided into 4×4 regions, for each region canonical orientation is recorded. **5: Fourier Transform:** Fourier transform represents function on new basis with basis elements $e^{-i2\pi(ux+vy)} = \cos(2\pi(ux+vy)) + i\sin(2\pi(ux+vy))$. $F(g(x,y))(u,v) = \iint_{\mathbb{R}^2} g(x,y) e^{-i2\pi(ux+vy)}\,dx\,dy$. Direction of vector (u,v) gives orientation, magnitude the frequency. Magnitude for natural images very similar, phase (initial offset) differs. 5.1) **Properties:** 1.) Transform is linear 2.) Inverse FT: $f = \mathcal{U}^{-1}F$ or $g(x,y) = \iint_{\mathbb{R}^2} F(u,v) e^{i2\pi(ux+vy)}\,du\,dy$ 3.) Scaling function down ⇔ Scaling transform up 4.) FT of Gaussian is a Gaussian 5.2) **Examples:** Sin wave:  ⇔  ,  ⇔  , Impulse:  ⇔  , Box:  ⇔  [for sinc(x), inverse also holds, i.e. spatial sinc(x) is frequency box]. 5.3) **Convolution Theorem:** The Fourier transform of the convolution of two functions is the product of their Fourier transforms: $F \cdot G = \mathcal{U}(f * g)$. The FT of the product of two functions is the convolution of the FTs: $F * G = \mathcal{U}(f \cdot g)$ 5.4) **Sampling:** Modeled with δ-function: $\mathrm{Sample}_{2D}(f(x,y)) = f(x,y)\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\delta(x-i, y-j)$. HF lead to problems with sampling, should be suppressed before sampling. 5.5) **Computation:** 2D FT can be computed as seq. of 1D FTs: $F(g(x,y)) = \int\left(\int g(x,y) e^{-i2\pi ux}\,dx\right) e^{-i2\pi vy}\,dy$ 5.6) **Image Restoration:** $f(x) \to h(x) \to g(x) \to \tilde{h}(x) \to \tilde{f}(x) \to f(x)$, i.e. the „inverse kernel" $\tilde{h}(x)$ should compensate image degredation, $(\tilde{h} * h)(x) = \delta(x)$ or $\mathcal{F}[\tilde{h}](u,v) \cdot \mathcal{F}[h](u,v) = 1$. $\mathcal{F}[\tilde{h}]$ can completely cancel some frequencies / amplify noise → regularize it. **6: Unitary Transforms:** Image as a column vector of length $M \times N$, $\vec{f}$. Linear image processing is then $\vec{c} = A\vec{f}$. Matrix A (size $MN \times MN$)

③ is unitary if $A^{-1}=A^H$ (for real $A \to A^{-1}=A^T \to$ orthogonal). Unitary transforms are „energy conserving" (conserv vector lengths, are rotation and potentially sign flip of coord. system). **6.1) Energy Distribution:** $F=[\vec{f}_1 \vec{f}_2 \dots \vec{f}_n]$ is the image collection, $R_{ff}=\mathbb{E}[\vec{f}_i \vec{f}_i^H]=\frac{F \cdot F^H}{n}$ (with substracted means → covariance matrix) the autocorrelation function/matrix, which is the expected value of the components. Unitary transforms conserve energy, but can distribute it unevenly. $R_{cc}=\mathbb{E}[\vec{c}\vec{c}^H]=\mathbb{E}[A\vec{f}\cdot\vec{f}^H A^H]=AR_{ff}A^{HH}$. Mean squared values („average energies") of coefficients c_i are on diagonal: $\mathbb{E}[c_i^2]=[R_{cc}]_{i,i}$ **6.2) PCA/KL-Transform:** For the eigenmatrix Φ of the autocorrelation matrix R_{ff}: 1.) Φ is unitary 2.) Columns of Φ form a set of eigenvectors of R_{ff}: $R_{ff}\Phi=\Phi\Lambda$ (with $R_{ff}=\Phi\Lambda\Phi^H$) $\Lambda=\mathrm{diag}(\lambda_1,\dots,\lambda_N)$ 3.) R_{ff} is symmetric non-negative definite → $\lambda_i \geq 0 \forall i$ 4.) R_{ff} is a normal matrix: $R_{ff}^H R_{ff}=R_{ff}R_{ff}^H$, unitary eigenmatrix therefore exists. The KL-transform is a unitary transform with $A=\Phi^H$, i.e. $\vec{c}=\Phi^H\vec{f}$, where the values of Φ are ordered according to decreasing eigenvalues. Transform coefficients are pairwise uncorrelated: $R_{cc}=AR_{ff}A^H=\Phi^H R_{ff}\Phi=\Phi^H\Phi\Lambda=\Lambda$. Transform has optimal energy conservation: No other unitary transform packs as much energy into the first j coeff. the MSE by choosing only first j coeff. is minimized. Proof: Consider $\vec{b}=I_j\vec{c}$. $E=\mathrm{Tr}(R_{bb})=\mathrm{Tr}(I_j A R_{ff} A^H)=\sum_{k=0}^{j-1} a_k^T R_{ff} a_k$. Introducing Lagrange multipliers $\sum_{k=0}^{j-1}\lambda_k(1-a_k^T a_k)$ and differentiating w.r.t. a_j gives: $R_{ff}a_j=\lambda_j a_j$ **Eigenimages:** For any unitary transform, the inv. transform $\vec{f}=A^H\vec{c}$ can be interpreted as „superposition" of „basis images" (=columns of A^H) of size MN. For KL: Basis images = Eigenfaces, only limited set needed to approx. images. **Face recognition:** Reduce dim. → faster computation (MN to j) 1.) Normalize input image 2.) Subtract mean face 3.) Calculate $p=\Phi_K^H x$, compare with DB entries. **6.3) LDA:** LDA maximizes between-class variance, while minimizing within-class variance. Leads to „Fisherfaces" → better with varying lightning

7: Pyramids/Wavelets

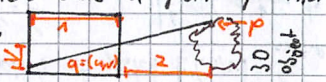
7.1) JPEG: Humans don't resolve high freq. too well → used by JPEG, applies discrete cosine transform (DCT) on 8x8 blocks. DCT coeff. are quantized (potentially lossy) and ordered according to freq. („zig-zag" pattern ). First coeff. = DC, only diff. to prev. DC encoded. Sequential encoding = Single block at a time, progressive encoding = „Merging" of some locations in neighboring blocks. Huffmann encoding (least bits for most common words) used. **7.2) Image Pyramid:** Image blurred and subsampled at every level. Used to search for correspondence, edge tracking, computational cost. Gaussian pyramid → Redundant (info) because LP filter. **Laplacian Pyramid:** Diff. between upsampled Gaussian and Gaussian pyramid level is preserved (synthesis). For analysis, Gaussian pyramid is reconstructed. Band pass filter → (largely) no redundant info. **7.3) Wavelets:** Wavelet is a wave-like oscillation with amplitude that starts at 0, increases and decreases back to 0. Used to extract information from data, generally multiple needed to analyze data fully. **Haar Transform:** Simple wavelet transform, function is „converted" to wavelet series by orthonormal series of wavelets that are generated by two operations: Scaling (for capturing info at diff. freq.) + translation (diff. location). **LeGall filters:** Perfect LP/HP filters would be infinitely large → Use 2 analysis/synthesis filters. **7.4) JPEG 2000:** 1.) Color components are transformed 2.) Image is tiled (possibly 1 tile) 3.) Tiles are wavelet transformed 4.) Quantization 5.) Coding. Steps 1,3,4 can be lossy/lossless

8: Optical Flow:

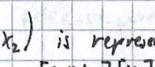
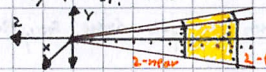
Try to track where pixels move to. Used for tracking, structure from motion, motion segmentation, stabilization, compression, ... **8.1) Brightness Constancy:** Of is the apparent motion of brightness patterns. Let $I(x,y,t)$ be brightness at (x,y) at time t. $I(x+\frac{dx}{dt}\delta t, y+\frac{dy}{dt}\delta t, t+\delta t)=I(x,y,t)$. Taylor expansion: $I(x+\frac{dx}{dt}\delta t, y+\frac{dy}{dt}\delta t, t+\delta t)=I(x,y,t)+\frac{\partial I}{\partial x}\frac{dx}{dt}\delta t+\frac{\partial I}{\partial y}\frac{dy}{dt}\delta t+\frac{\partial I}{\partial t}\delta t+\text{H.O.T.}$ Leads (with the assumption) to the constraint equation: $\frac{\partial I}{\partial x}\frac{dx}{dt}+\frac{\partial I}{\partial y}\frac{dy}{dt}+\frac{\partial I}{\partial t}=0$ **8.2) Aperture Problem:** Let $u=\frac{dx}{dt}$, $v=\frac{dy}{dt}$, $I_x=\frac{\partial I}{\partial x}$, $I_y=\frac{\partial I}{\partial y}$, $I_t=\frac{\partial I}{\partial t}$. $I_x u+I_y v+I_t=0$ → 2 unknowns, only one equation. Defines line in (u,v) space normal to $-\frac{I_t}{\|\nabla I\|}\frac{\nabla I}{\|\nabla I\|}$ **8.3) Horn+Schunck Algorithm:** Additional smoothness constraint: $e_s=\iint((u_x^2+u_y^2)+(v_x^2+v_y^2))dxdy$ [$u_x=\frac{\partial u(x,y)}{\partial x}$, i.e. want to min. „changes in flow"], $e_c=\iint(I_x u+I_y v+I_t)dxdy$. Min. $e_s+\lambda e_c$ [iter. methods, finite differences]. **8.4) Lukas-Kanade Algorithm:** Single velocity for all pixels of patch. $E(u,v)=\sum_{(x,y)\in\Omega}(I_x(x,y)+I_y(x,y)+I_t)^2$. Solved with LSQ → $\begin{bmatrix}\sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2\end{bmatrix}\begin{pmatrix}u\\v\end{pmatrix}=-\begin{pmatrix}\sum I_x I_t\\ \sum I_y I_t\end{pmatrix}$, $\sum=\sum_{(x,y)\in\text{window}}$. If all gradients in patch point into same direction (e.g. along edge) → singular matrix → aperture problem **Iterative Refinement:** Velocity is estimated using Lukas-Kanade, image is warped using estimate, estimate is refined by rep. on diff. pyramid levels **8.5) Coarse-to-fine:** Gradient method fails with poor intensity structure in patch and large displacements. → Use different levels of image pyramid to estimate at different scales. **8.6) Parametric Motion Models:** Use not only translation, but also affine/quadratic models. E.g. affine: $E(A,h)=\sum[I(Ax+h)-I(x)]^2$ Offer more constrained solutions than smoothness / integration over larger area than translation-only. **8.7) Bayesian Optic Flow:** Can explain some low-level human motion illusion by adding uncertainty to Lucas-Kanade → Brightness constancy with noise $I(x,y,t)=I(x+u\delta t, y+v\delta t, t+\delta t)+n$, n ind. Gaussian.

6) Arrive at $v^* = -\begin{bmatrix} \sum I_x^2 + \frac{\sigma^2}{\sigma_{pr}^2} & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 + \frac{\sigma^2}{\sigma_{pr}^2} \end{bmatrix}^{-1} \begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$ → One param.: Ratio of observation/prior spread.

2.8) SSD: Can use template matching for large displacements. 9: Video Compression: Visual Perception limited to <24Hz, but still need to avoid aliasing. Interlaced video = Two half images (even/odd rows) displayed consecutively, increasing frequency. Full image repr. = Progressive. Lossy video compression takes advantage of redundancy: Spatial correlation between neighboring pixels, temporal correlation between frames. 9.1) Temporal Redundancy: Current frame is predicted based on prev. coded frames. 3 type of frames: 1.) I-frame: Intra-coded frame, coded independently 2.) P-frame: Predictively-coded frame based on prev. frame 3.) B-frame: Bi-directionally predicted frame, based on prev. and future frame. For P-/B-frames, only residual is encoded. Better performance with motion compensation (MC): Motion described on block-basis with motion vector. Vector generated by exhaustive search or „multi-step" approach: Coarse grid search + refinement (up to subpixel). For B-frames, MC-prediction can be based on prev./future, both (avg.) or none. GOP: MPEG has group of pictures; seq. that starts with I-frame and ends with frame before next I-frame, and has defined nr. of P-/B-frames → Allows decoding without going to start of video. 9.2) Spatial Redundancy: Frames/residuals are encoded similarly to images (DCT) 9.3) PSNR: Error for 1 pixel: Diff. between original/decoded $e_{mse} = \sqrt{\frac{1}{N \cdot M} \sum_{v=1}^{N} \sum_{u=1}^{M}}$. PSNR: $\frac{\text{max. value of } x}{e_{mse}^2}$ 10: Sparsity and Texture: 10.1) Image Denoising: Try to represent image as sparse vector of basis images (designed, e.g. wavelets, or learned) to reduce noise. 10.2) Representing Texture: Want to find the stylized subelements of texture and represent their stats. To find them, filters are applied and response magnitude is recorded. 10.3) Texture Synthesis: Image can be used as source of prob. model. Better results by also modeling co-occurrence. Alternative: Image-based approaches („copy-pasting"), e.g. chaos mosaic that tiles image, picks random blocks and smoothes edges. Or choose blocks s.t. overlap is maximized. 11: Radon Transform In computer tomography we can measure reduction in intensity of an X-ray, given by $I_{finish} = I_{start} e^{-R}$, $R = \int_L u(s)ds$, where $u(s)$ is the optical density of the material (what we want to estimate). Line L can be reparametrized: $L = (p\cos(\theta) - s\sin(\theta), p\sin(\theta) + s\cos(\theta))$, which gives $R(p,\theta) = \int u(p\cos(\theta) - s\sin(\theta), p\sin(\theta) + s\cos(\theta))ds = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} u(x,y)\delta(p - x\cos(\theta) - y\sin(\theta))dxdy$, which is the Radon transform of $u(x,y)$. $R(p,\theta)$ is called sinogram because RT of off-center point is sinusoid 11.1) Backprojection: Given $R(p,\theta)$, can we find $u(x,y)$? → Ill-posed problem, backprojection propagates measured sinograms back into image space to construct $u(x,y)$. 11.2) Central Slice Theorem: The FTs of $R(p,\theta)$ [$\tilde{R}(v,\theta)$] and $u(x,y)$ [$\tilde{u}(v_x, v_y)$] are related. For a given θ, $\tilde{R}(v,\theta) = \tilde{u}(v\cos\theta, v\sin\theta)$. Therefore, for all θ: 1.) Measure attenuation $R(p,\theta)$ 2.) 1D FT: $\tilde{R}(p,\theta)$ 3.) 2D IFT and sum with prev. values to reconstruct $u(x,y)$. After step 2, HP filter for noise reduction can be applied („filtered backprojection"). Computer Graphics Use of computers to synthesize/manipulate visual information. 12.1) Perspective Projection: Where does a point $p = (x,y,z)$ end up in the image? Assuming pinhole model, unit size camera:



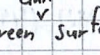
We have $\frac{v}{1} = \frac{y}{z}$, therefore $v = \frac{y}{z}$ and likewise $u = \frac{x}{z}$ 12.2) Rasterization: Conversion of a continuous object to a discrete representation on a pixel grid. Line Rasterization: Only light pixels up where „virtual diamond" is intersected:  Incremental Line Rasterization: Don't check all n^2 pixels, for a line with points (u_1,v_1), (u_2,v_2), slope is $s = \frac{v_2 - v_1}{u_2 - u_1}$. Travel to next pixel according to s. 13: Drawing Triangles: Need to answer: 1.) What pixels does the triangle overlap? (coverage) 2.) What triangle is closest to camera in each pixel? (occlusion) 13.1) Triangle Coverage: Can color pixel according to fraction of pixel area covered by triangle, which can be estimated by sampling (multiple measurements per pixel). Rule needed when edge is on sampling point (OpenGL: „within" if edge is top/left edge). Supersampling = more than one sample per pixel. Resampling = Converting from one sampled resolution to another (e.g. supersampled to display → anti-aliased edges). 13.2) Sampling Coverage: Point-in-triangle test is used to test if sample is inside: Given a triangle by $P_i = (X_i, Y_i)$ $(i \in \{0,1,2\})$, $dX_i = X_{i+1} - X_i$, $dY_i = Y_{i+1} - Y_i$, we define $E_i(x,y) = (x - X_i)dY_i - (y - Y_i)dX_i$. Point is inside if $E_0(x,y) < 0$, $E_1(x,y) < 0$, $E_2(x,y) < 0$. Instead of testing all points, various traversal orders possible → GPUs often do tiled triangle traversal with parallel testing in blocks. 14: Transforms: 14.1) Linear Maps: Reasons: 1.) Computationally easy to solve eq. 2.) Over short distance, all maps approx. by linear one 3.) Composition of linear transforms is linear → uniform representation. Key idea is that they take lines to lines while keeping the origin fixed.

5) Map f is linear if it maps vectors to vectors and $f(u+v)=f(u)+f(v)$, $f(au)=af(u)$. Linear maps between $\mathbb{R}^m$ and $\mathbb{R}^n$ can be expressed as: $f(u)=\sum_{i=1}^{m} u_i a_i$ (for fixed vectors a_i). For $\mathbb{R}^2$, $f(u)=u_1a_1+u_2a_2$ with $a_1=f(e_1), a_2=f(e_2)$, i.e. we only need to know $f(e_1), f(e_2)$ to map the entire space. 14.2) **Scale**: Uniform Scale: $S_a(x)=ax$, Non-uniform: $S(x)=x_1ae_1+x_2be_2$, $A=\begin{bmatrix}a&0\\0&b\end{bmatrix}$ 14.3) **Rotation**: R_θ=rotate counter-clock by θ. Rigid/isometric→shape (distance between points) doesn't change. $R_\theta(x)=x_1a_1+x_2a_2$, $a_1=(\cos\theta,\sin\theta)$, $a_2=(-\sin\theta,\cos\theta)$, $A=\begin{bmatrix}\cos\theta&-\sin\theta\\\sin\theta&\cos\theta\end{bmatrix}$ 14.4) **Reflection**: $Re_y(x)$=Reflection about y-axis. $Re_y(x)=-x_1e_1+x_2e_2$, $A=\begin{bmatrix}-1&0\\0&1\end{bmatrix}$ 14.5) **Shear**: $H_a(x)=x_1\begin{bmatrix}1\\0\end{bmatrix}+x_2\begin{bmatrix}a\\1\end{bmatrix}$, $A=\begin{bmatrix}1&a\\0&1\end{bmatrix}$  14.6) **Translation**: Not linear, but affine ($f(x)=ax+b$ is affine, not linear) 14.7) **Matrix**: Can write linear transformations as matrix-vector product: $\begin{bmatrix}a_{11}&a_{12}\\a_{21}&a_{22}\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=x_1\begin{bmatrix}a_{11}\\a_{21}\end{bmatrix}+x_2\begin{bmatrix}a_{12}\\a_{22}\end{bmatrix}$ 14.8) **2D Homogeneous Coordinates**: 2D-point (x_1,x_2) is represented as 3D vector $\begin{bmatrix}x_1\\x_2\\1\end{bmatrix}$. E.g. rotation becomes $\begin{bmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\1\end{bmatrix}$. In HG, translation is linear: $\begin{bmatrix}1&0&b_1\\0&1&b_2\\0&0&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\1\end{bmatrix}=\begin{bmatrix}x_1+b_1\\x_2+b_2\\1\end{bmatrix}$. $(wx_1,wx_2,w)\ \forall w>0$ represent the same 2D point, →need to divide by w to convert 2D-H to 2D. 2D-H points with $w=0$→2D vectors. 14.9) **Composition**: By matrix mult. Order right-to-left, $f(x)=R_{\frac{\pi}{4}}S_{[2,3]}x$ is scaling, then rotation. **Rotation about point x**: 1.) Translate by $-x$ 2.) Rotate 3.) Translate by x 14.10) **3D-H**: E.g. shear (in x, based on y,z): $\begin{bmatrix}1&a_y&a_z&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$ 14.11) **3D rotations**: About x-axis $\begin{bmatrix}1&0&0\\0&\cos\theta&-\sin\theta\\0&\sin\theta&\cos\theta\end{bmatrix}$, y-axis: $\begin{bmatrix}\cos\theta&0&\sin\theta\\0&1&0\\-\sin\theta&0&\cos\theta\end{bmatrix}$, z: $\begin{bmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{bmatrix}$ 14.12) **Camera transform**: Want to compute transform from world to camera coord. system (camera at origin, looking down z) 1.) Translation by $-x$ (x=camera pos.) 2.) Form orthonormal basis (u,v,w) around camera direction, w 3.) Map u to x-, v to y-, w to z-axis: $R=\begin{bmatrix}u_x&v_x&w_x\\u_y&v_y&w_y\\u_z&v_z&w_z\end{bmatrix}$ 14.13) **Perspective Projection**: Given $\begin{matrix}x\\y\\z\end{matrix}$=$(x_x,x_y,x_z)$, $p_{2D}=(x_x/x_z, x_y/x_z)$. Done with $P=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&0\end{bmatrix}$ 14.14) **View Frustum**: Region in space that will appear on screen:  Let θ be the field of view in y-direction, $f=\cot\left(\frac{\theta}{2}\right)$ and r=aspect ratio=$\frac{\text{width}}{\text{height}}$. Can extend P s.t. frustum is mapped to unit cube→$P(0,0)=\frac{f}{r}$, $P(1,1)=f$, $P(2,2)=\frac{z_{far}+z_{near}}{z_{near}-z_{far}}$, $P(2,3)=\frac{2z_{far}z_{near}}{z_{near}-z_{far}}$ 14.15) **Screen Transform**: After projection, all points from $(-1,-1)$ to $(1,1)$ are on screen, →need to transform to $(0,0)$ to (W,H) for output: 1.) Reflect about x-axis 2.) Translate by $(1,1)$ 3.) Scale by $(W/2, H/2)$

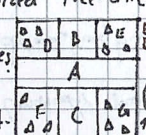
15: Geometry

15.1) **Implicit Representations**: Points aren't known directly, but satisfy some relationship. Pros: 1.) Can be compact 2.) Easy inside/outside check 3.) Other queries (e.g. distance) may also be easy 4.) Exact description 5.) Easy topology changes. Cons: 1.) Expensive to find all points in shape 2.) Difficult to model complex shapes **Algebraic Surfaces**: Surface is the zero set ($f(x,y,z)=0$) of a polynomial in x,y,z, e.g. $x^2+y^2+z^2=1$→unit sphere **Constructive Solid Geometry**: Complicated surfaces built using boolean operations (union, intersection, difference) **Distance Functions**: Gives distance to closest point on object **Level Set Methods**: Grid of values approx. function is stored, surface is where interpolated values $=0$ **Fractals / L-Systems**: Structures that exhibit self-similarity at all scales. L-system: Set of grammar rules, start symbol, semantics. E.g. grammar $\{A\to BBC, B\to a[b]A\}$ with start symbol AAA. Semantics could be a=forward 10, b=(left rand(10,50),... Can generate e.g. trees with very simple rules. 15.2) **Explicit Representations**: Points are given directly→easy sampling, hard inside/outside tests. **Point Cloud**: List of points (x,y,z), sometimes augmented with normals. Drawing in undersampled regions can be difficult. **Polygonal Mesh**: Definition polygon: Vertices $v_0,\dots,v_{n-1}$, edges $\{(v_0,v_1),\dots,(v_{n-1},v_{n-1})\}$, planar + non self-intersecting. Polygonal mesh: Set of connected polygons $M=\langle V,E,F\rangle$ (F=faces). Every edge belongs to at least one polygon, intersection of two polygons is empty, vertex or edge. Vertex degree/valence=Incident edges to vertex. Boundary=Set of edges that belong to only one polygon. Mesh data structure store geometry (vertex locations), topology (connectivity, i.e. edges/faces), attributes (normal, color, ...). Need to support rendering, geometry queries (e.g. vertices of given face), modifications. **Triangle Mesh**: Triangle List: Listing of all 9 coord. per triangle. Indexed face set: Vertices stored as coord. (x,y,z), triangles as triplets of indices (i,j,k) **Subdivision Surfaces**: Control curve is smoothed out, new vertices inverted according to update rule. 15.3) **Geometry Sources**: 3D scanning (point cloud common), digital 3D modeling (→subdivision surfaces).

16: Texture

16.1) **Lambert's Law**: Irradiance at surface prop. to cos of angle between light direction/surface normal. Given flux Φ and  A' ($A=A'\cos\Phi$), $E=\frac{\Phi}{A'}=\frac{\Phi\cos\theta}{A}$ **N-dot-L Lightning**: Dot product between unit surface normal (N) and unit direction to light (L), i.e. $\max(0,\text{dot}(N,L))$. Common abstraction: Infinitely bright light source „at infinity"→same L 16.2) **Color**: When assigning colors to triangle vertices→barycentric coord. for in-between colors. Given triangle (a,b,c), solve $x=a+\beta(b-a)+\gamma(c-a)=(1-\beta-\gamma)a+\beta b+\gamma c=\alpha a+\beta b+\gamma c$, with $\alpha+\beta+\gamma=1$. Then, $x_{color}=\alpha a_{color}+\beta b_{color}+\gamma c_{color}$ Can also be computed as area ratio:  $\alpha=\frac{A_A}{A}$, $\beta=\frac{A_b}{A}$, $\gamma=\frac{A_c}{A}$. Barycentric interpolation in screen XY-coord. perspective-incorrect, needs to be done in 3D obj. space: 1.) Compute depth z at eye

6 vertex. 2.) Eval. $z=1/2$, $p=f/2$ 3.) Interpolate z/p with barycentric coord. 4.) Divide interpolated p by interpolated z to get f. **16.3) Texture Mapping:** Many uses: Surface material prop., normal mapping (texture value used to perturb surface normal), precomputed lightning/shadows. Texture coord. define mapping from surface coord. (points on triangle) to points in texture domain. They are linearly interpolated over the triangle → only defined for vertices. Texture mapping (for each covered screen sample): 1.) (u,v) = eval. texcoord. at (x,y) 2.) Sample texture^{color} at (u,v) 3.) Set samples color to ter. color. Texture sample positions may not be uniformly distributed in texture space. To avoid aliasing → pre-filter textures. Mipmap = Precomputed texture at different resolutions/filter levels. Choose level according to texture coord. distance of neighboring pixels (far object → strongly filtered level). **Sec 17. Rasterization Pipeline** **17.1) Occlusion:** Depth buffer (z-buffer) can be used to decide which object is visible at each pixel. Stores the depth of closest triangle that has been processed$^{per\ sample}$ so far → processing order doesn't matter, works with interpenetrating triangles/supersampling. **17.2) Compositing:** Alpha = Additional image channel for opacity ($\alpha=1$ → fully opaque, $\alpha=0$ → fully transparent). "over" operator composites image A over B. A over $B \neq B$ over A. If we composite B over A, composited color is: $C=\alpha_B B + (1-\alpha_B)\alpha_A A$. Can also store $A'=[\alpha_A A_r \ \alpha_A A_g \ \alpha_A A_b \ \alpha_A]^T$, then $C'=B'+(1-\alpha_B)A'$. Rendering: Check if depth test passed, update color but not depth buffer (for semi-transparent objects) → Must render in back-to-front order. For mixture of opaque/transparent: 1.) Render opaque surfaces using normal z-buffer 2.) Disable z-buffer updates, render semi-transparent back-to-front and composite color over current color at sample. **17.3) Real-Time Graphics pipeline:** Input: Vertices in 3D space. 1.) Vertex operations (transformation matrices) 2.) Operations on primitives (lines, triangles, etc...) 3.) Fragment operations: Fragment generation (rasterization) which produces fragment stream, fragment processing (with textures) which takes stream and produces shaded fragment stream 4.) Operations on screen samples (depth/color). **17.4) Shader Programs:** Define behavior of vertex/fragment processing stages. Describe operation on single vertex/fragment. **18: Rendering Equation** **18.1) Light:** Light is electromagnetic radiation, color determined by the frequency. "White" light = Mixture of all visible frequencies. Emission spectrum of a light source (additive model): How much light at different wavelengths is produced → useful for comparing light sources. Absorption spectrum (subtractive model): How much light is absorbed → useful for comparison/characterizing color of paint, ink, ... Brain: We have S, M, L cones (short, medium, long wavelengths) in eyes, brain interprets (S, M, L) response as color. **18.2) Color Spaces:** Color from some space (like an artist's palette) is specified using a color model. Additive models e.g. used for combining colored lights) like RGB, subtractive models (e.g. used for combining pain colors) like CMYK. There's also HSV (hue, saturation, value) and Y'CbCr (intensity, blue-yellow deviation from gray, red-cyan deviation). **18.3) Radiometry:** Assume photons travel in straight lines (rays). Each photon carries energy, want to record amount of energy (→ brightness). Radiant Energy: Total number of photons hitting scene. Radiant Flux: Hits per second. Irradiance: Hits per second per unit area. Radiance: Energy along ray, defined by origin point p and direction ω. Complete description of light in an environment → Rendering is about computing radiance. Need to distinguish incident ($L_i(p,\omega)$) and exitant ($L_o(p,\omega)$) radiance at point on surface. In general, $L_i(p,\omega) \neq L_o(p,\omega)$. Irradiance from radiance: $E=\int_{H^2} L_i(p,\omega)\cos\theta d\omega$, where θ is angle between ω and normal. **18.4) Isotropic Point Source:** Slightly more realistic lightning model, where light spreads in every direction. Irradiance on surface at distance r: $E=\frac{\Phi}{4\pi r^2}$ → darker quadratically with distance. **18.5) Rendering Equation:** $L_o(p,\omega_o)=L_e(p,\omega_o)+\int_{H^2} f_r(p,\omega_i\to\omega_o)L_i(p,\omega_i)\cos\theta_i d\omega_i$, where: L_o = outgoing/observed radiance, p = point of interest, ω_o = direction of interest, L_e = emitted radiance (from e.g. light source), H^2 = all directions of hemisphere, L_i = incoming radiance, θ_i = angle between incoming direction/normal (Lambert's law). f_r is reflection/scattering function and describes how reflection affects outgoing radiance. Choice determines surface appearance: 1.) Ideal specular (perfect mirror) 2.) Ideal diffuse (uniform reflection in all directions) 3.) Glossy specular (majority reflected in reflection direction) 4.) Retro-reflective (back towards source) **18.6) BRDF:** Bi-directional reflectance distribution function, type of reflection function with properties: 1.) $f_r(\omega_i\to\omega_o)\geq 0$ 2.) $\int_{H^2} f_r(\omega_i\to\omega_o)\cos\theta d\omega_i \leq 1$ 3.) Helmholtz-reciprocity: $f_r(\omega_i\to\omega_o)=f_r(\omega_o\to\omega_i)$ [violated by subsurface scattering → (light exiting at different points than it enters)] **19: Raytracing** We „shoot" a ray for

⑰ every screen sample, color sample according to hit object. Proceed in screen sample order (vs. rasterization, which proceeds in triangle order). No depth buffer needed and natural order for processing transparent surfaces (order they appear along ray), but entire scene must be stored. Can view rasterization as efficient algorithm for visibility queries, given rays with same origin and uniform distribution over projection plane. 19.1) Shadows: For rasterization → shadow mapping: Scene is rendered (depth buffer only) from light location, this buffer is then used when rendering from camera POV (for shadow queries). For raytracing → „shadow rays" are shot towards light source from place where camera rays intersect scene. 19.2) Reflections: For rasterization → Render scene 6 times (all directions) from reflective's object POV, apply renderings as textures to object. For raytracing → Shoot another ray when hitting reflective object (can model surfaces by reflection function) → recursive raytracing. 19.3) Ray Intersection: Ray given by $r(t)=o+td$, o=origin, d=dir. vector. Intersection with implicit surface: Plug in r for x, solve for t. E.g. plane, given by $N^T x=c \rightarrow N^T(o+td)=c \rightarrow t=\frac{c-N^T o}{N^T d}$, i.e. $o+\frac{c-N^T o}{N^T d}$ as intersection point. Ray-triangle intersection: Intersect with plane, ~~compute~~ barycentric coordinates check for inside intersection point. 19.4) Ray-Scene Intersection: Given scene with N primitives, find first hit object. To do better than $O(N)$ → Put axis-aligned boxes around objects, only consider primitive in hit boxes. Bounding Volume Hierarchy: Divides space into set of bounding boxes, organized tree-like. Interior nodes represent subset of primitives, leaf nodes contain list of primitives. 

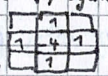
Primitives are partitioned into disjoint sets, but boxes can overlap in space. „front-to-back" traversal works like this: 1.) Check if box not hit or closest point on box farther than curr. min. distance → return 2.) If leaf node → Traverse all primitives 3.) Otherwise, traverse first child where closest point is nearer. To implement partitioning → Can use axis-aligned approach. Split plane/line along axis is chosen, primitives are partitioned according to the side of their centroid → Can evaluate small number of split planes with a heuristic to get plane. Space-Partitioning Acceleration Structures: In contrast to BVH, can also partition space instead primitives into disjoint sets → Primitive can be contained in multiple regions. E.g. uniform grid where each cell contains primitives that overlap it → Allows efficient walking through (similar to line rasterization). K-D trees partition space via axis-aligned planes (partitions can have different size → useful for non-uniform primitive distribution). Unlike BVH, can terminate search after first hit. Quad-trees (2D) / octrees (3D): Uniform grid, but cells can be further divided (by 4/8) if they have too many primitives. 20: Animation: Creating illusion of motion, e.g. by rapidly displaying minimally changing images. Some principles: 1.) Squash and stretch (shape distortion) 2.) Anticipation (action preparation) 3.) Follow through (termination of action) 4.) Slow in, slow out (timing). Hand-drawn motion: Keyframes by senior artist, inbetweens by assistant. Techniques in computer animation: 1.) Artist-directed (e.g. keyframing 2.) Data-driven (e.g. motion capture) 3.) Procedural (e.g. simulation) 20.1) Keyframing: Specify important events (position, color, lights, camera configuration, ...) only, interpolate rest. Splines: Any piecewise polynomial. 1D: (t_i, f_i), $f(t_i)=f_i \; \forall i$. Has to be polynomial between knots: $t_i \le t \le t_{i+1}$, $f(t)=\sum_{j=0}^{d} c_i t^j =: p_i(t)$. Most common: $d=3$ (min. norm of 2nd derivative for small displacements). Runge's phenomenom: Higher degree can lead to worse approx. (oscillation). Natural Splines: p_i cubic. 1.) Interpolation: $p_i(t_i)=f_i$, $p_i(t_{i+1})=f_{i+1}$, $i=0,\dots,n-1$ 2.) C^1 continuity: $p_i'(t_{i+1})=p_{i+1}'(t_{i+1})$ 3.) C^2 continuity: $p_i''(t_{i+1})=p_{i+1}''(t_{i+1})$. $4n$ DOFs, only $4n-2$ eq. → $p_0''(t_0)=p_n''(t_n)=0$. Spline properties: 1.) Interpolation 2.) C^2 continuity 3.) Locality (moving one control point doesn't affect whole curve). Not all possible at the same time (natural: no locality). Hermite/Bézier Splines: Tangents are also specified (don't have to match), i.e. $p_i(t_{i+1})=u_i$, $p_i'(t_{i+1})=u_{i+1}$, $i=0,\dots,n-1$ → No C^2 continuity. Catmull-Rom Splines: Like Bézier, but tangent is calculated as diff. of neighbors: $u_i=\frac{f_{i+1}-f_{i-1}}{t_{i+1}-t_{i-1}}$. B-Spline: No. (exact) interpolation. $B_{i,1}(t)=1$ for $t_i \le t \le t_{i+1}$, 0 oth. $B_{i,k}(t):=\frac{t-t_i}{t_{i+k-1}-t_i} B_{i,k-1}(t)+\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} B_{i+1,k-1}(t)$ $f(t)=\sum_i a_i B_{i,d}$ 20: Rigging: Character animation pipeline = Modeling, rigging, animation. Rig = User-defined mapping between params and deformations of high-res mesh. Rigging = Creating a rig. Animation then defined through set of (spline) curves, determining rig params over time. 21.1) Blend Shape Rigs: Input: Meshes M_i with vertices x_i^j, blending weights $\alpha=(\alpha_1,\dots,\alpha_n)$. Output: Blended mesh M through linear combination: $M=\sum_i \alpha_i M_i$, $x^j=\sum_i \alpha_i x_i^j$. Keyframes: Blending weights $\alpha(t_i)$. 21.2) Cage-based Deformers: High-res model embedded in a coarse mesh (cage), cage is deformed and deformations are applied to high-res model. For a point x_i in the high-res model: $x_i=\sum_j^m w_j(x_i) c_j$, c_j = points of cage, w_j = weights. 21.3) Skeletal Animation: Animate only skeleton, simulate muscles, fatty tissues, skin, environment interactions ... (computationally expensive). 21.4) Linear Blend Skinning: Deformations of surface mesh are coupled to skeletal pose. Forward Kinematics: Joints: Local coord. frames,

⑧ Bones = vectors between consecutive joints. Each non-root bone is defined in frame of unique parent, changes to parent frame affect all descendant bones. $p(j)$ = Parent of joint j, frame of joint j expressed w.r.t. its parent: $^{p(j)}R_j = \begin{pmatrix} Rot(j) & t(j) \\ 0 & 1 \end{pmatrix}$, i.e. rotation + translation. $t(j)$ typically constant during animation, $Rot(j)$ varies. Transformation from frame j to world: $^wR_j = {}^wR_0 \dots {}^{p(p(j))}R_{p(p(j))}{}^{p(j)}R_{p(j)}{}^jR_j = t(0)Rot(0)\dots t(p(j))Rot(p(j))t(j)Rot(j)$ **Skinning**: Given joint angles (i.e. rotation matrices), skeleton conf. can be computed using FK. How to adjust surface mesh? **Rigid skinning**: Assign each vertex to closest bone, compute world coord. according to bone's transformation: $v = {}^wR_j{}^wR_j^{-1}v'$, v' = vertex coord. in rest pose (first transformed in coord. frame of bone j, so ^wR_j can be applied). Works fine if vertices close to bone/small deformations. **Linear blend skinning**: Vertex assigned to multiple bones with weights (manually painted or computed), world coord. are convex combination: $v = \sum_j \alpha_j {}^wR_j{}^wR_j^{-1}v'$. → Smoother deformations, but „candy-wrapper" effects can happen: → Use better weights, auxiliary joints/bones, better interpolation scheme.

Inverse Kinematics: Given goal position for „end effector" (e.g. hand), compute joint angles to achieve position. → Analytic formulation, Energy-based methods, ... **22.3) Motion Capture**: We get joints target by MC. Becomes optimization-based IK: $\min_\theta f_0(\theta) = \frac{1}{2}(p(\theta)-\vec{p})^T(p(\theta)-\vec{p})$, $\vec{p}$ = target, $p(\theta)$ = FK-transformed point. Side constraints (e.g. max. joint angle) possible.

22: Physics-Based Animation Newton's 2nd law: $F = ma$, F = force, m = mass, a = acceleration. More general: System has config. $q(t)$, velocity $\dot{q} = \frac{d}{dt}q$, mass M, forces F are acting on it and potentially constraints $g(q,\dot{q},t) = 0$. Then, $\ddot{q} = \frac{F}{m}$, want to solve for $q(t)$ **22.1) Particle systems**: Complex phenomena modeled as collection of particles with attached forces acting on them **22.2) Numerical Integration**: Given $q(0), \dot{q}(0)$, find $q(t)$. Function $q(t)$ replaced by discrete samples q_i: $q_{i+1} = q_i + \Delta q_i \approx q_i + \int_t^{t+\Delta t} \dot{q}(t)dt$ (approx. ∫). 1.) Rectangle rule: $\Delta q_i = \dot{q}(t)\cdot h$ 2.) Midpoint rule: $\Delta q_i = \dot{q}(t+\frac{h}{2})\cdot h$ 3.) Trapezoid rule: $\Delta q_i = \frac{\dot{q}(t)+\dot{q}(t+h)}{2}$ **Forward Euler**: $q_{i+1} = q_i + h\cdot\dot{q}_i$ → Only stable for small timesteps **Backward Euler**: $q_{k+1} = q_k + T f(q_{k+1})$ → Unconditionally stable for linear ODEs. **Symplectic Euler**: Only for 2nd order, no numerical damping/energy is preserved. Velocity updated using curr. config., config. using new velocity.

23: PDEs: $u_{tt} = \frac{\partial^2}{\partial t^2}u(t,x)$, $u_{xy} = \frac{\partial^2}{\partial x \partial y}u$, $\nabla = \left(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n}\right)^T$, $\Delta = \nabla^T\cdot\nabla = \sum_{k=1}^{d}\frac{\partial^2}{\partial x_k^2}$ **23.1) Classification**: Order = order of highest partial derivative, linear: function u and its derivatives only occur linearly (e.g. $u_t + c\cdot u_x = 0$ vs. non-linear: $u_t + u\cdot u_x = 0$) → coeffs can be non-linear function. 2nd order linear PDEs have form: $Au_{xx} + 2Bu_{xy} + Cu_{yy} = F(x,y,u,u_x,u_y)$. **1.) Hyperbolic** (wave eq.): $B^2 - AC > 0$ (wave front hard to solve/no steady state) **2.) Parabolic**: $B^2 - AC = 0$ → Heat eq. (spread of heat over time; solution same as Laplace after long time) **3.) Elliptic**: $B^2 - AC < 0$ → Laplace eq. (smoothest function interpolating boundary data; easy to solve, robust to errors)

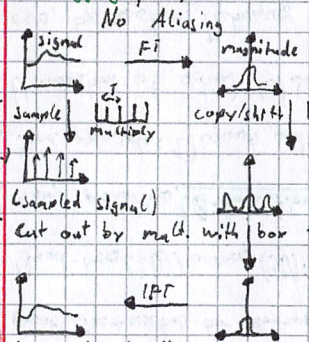
23.2) Numerical Solution: Pick spatial+time discretization, run time-stepping. Spatial: Lagrangian/Particles (conceptually easy, no resolution/domain limitation, good distribution can be hard, finding neighbors can be expensive) or Eulerian/Grid (fast, regular comp., easy representation, „trapped" in grid) **Laplace Eq.**: Need to approx. Laplacian, e.g.  → $\frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2}$. Solve $\Delta u = 0$ by weighted avg. ($^1/_4$) with neighbors until convergence (Jacobi method). Boundary: Dirichlet (fixed; any value can be satisfied) or Neumann (derivative is specified). **Heat Eq.**: $\dot{u} = \Delta u$. E.g. forward Euler, $u^{k+1} = u^k + \Delta u^k$, where Laplacian is approx. as above. **Wave Eq.**: $\ddot{u} = \Delta u \xrightarrow{conv.} \dot{u} = v, \dot{v} = \Delta u$

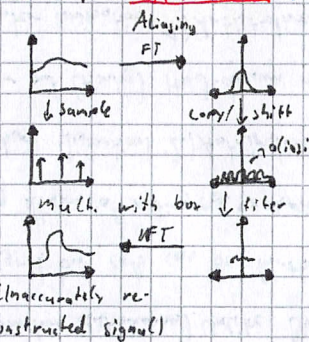
Exercises: BG subtraction: Exemplar set model: Use set of BG pixels, calc. mean μ / temp. covariance Σ. BG if $\sqrt{(x-\mu)^T\Sigma(x-\mu)} > t$. Per-pixel model: For each pixel, mean/cov. matrix calculated over BG frames. Then, above classification rule used. **Filtering**: Creation of Gaussian HP: Extract HP content by subtracting LP filtered img. from original → HP filter constructed by subtracting LP from unit response. **PCA Face Detection**: Lowest compression error → most likely face location. **Transformations**: Rigid-body transformation: „roto-translation" → rotation + translation. Rotations commute only in 2D. **Coord. system change**: Given local coord. system with axes x,y,z and origin t, $F = [x\ y\ z\ t]$ $p_{world} = Fp$, if p is expressed in coord. system. **PVM**: $M = [x\ y\ z\ t]$ is model matrix that expresses model coord. system. C is the camera matrix, typically $C = TR$ (translation/rotation) $\rightarrow p_{cam} = Mp_{obj}$ → $p_{cam} = C^{-1}Mp_{obj}$. V is the view matrix, $V = C^{-1} = R^{-1}T^{-1} = R^TT^{-1}$. Screen projection from camera point with projection matrix P, therefore $p_{cam} = PVMp_{obj}$ **Phong Shading**: Phong reflection model: $I = I_aK_a + I_dK_d(N\cdot L) + I_sK_s(R\cdot V)^\alpha$, a=ambient, d=diffuse, s=specular, K_a=material constant, K_d/K_s=reflection factor, α=exponent for surface character (smooth ≥32, perfect mirror ∞). L=light dir., N=surface normal, V=viewing dir., R=reflection dir. with $R = 2(L\cdot N)N - L$ **Linear Interpolation**: $y_0 + (t-t_0)*(y_1-y_0))/(t_1-t_0)$

Appendix: Assumptions Lucas Kanade: 1.) Brightness constancy 2.) Small motion 3.) Spatial coherence: All points in neighborhood have same motion. **Radians**: $\pi = 180°$ **Trigonometry**: $\sin(0°) = 0$, $\cos(0°) = 1$, $\sin(90°) = 1$, $\cos(90°) = 0$, $\sin(-90°) = -1$, $\cos(-90°) = 0$

Table of Contents:

Appendix II: Fourier Transform: Sampling Visualization





10 Fourier Transform:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi x\xi}\,dx$$

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)e^{2\pi i x\xi}\,d\xi$$

Matrix Multiplication:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \cdot \begin{bmatrix} g & j \\ h & k \\ i & l \end{bmatrix} = \begin{bmatrix} ag+bh+ci & \dots \\ \vdots & \ddots \end{bmatrix}$$

(2×3) (3×2) (2×2)