Image Processing and Computer Graphics

Riccardo Salvalaggio

19th of April, 2021

Contents

I Computer Graphics **5**

1 Introduction Computer Graphics **6**
1.1 Rendering aspects . . . 6
1.2 Light . . . 6

2 Ray Casting **8**
2.1 Implicit surfaces . . . 8
2.2 Parametric surfaces . . . 9
2.3 Combined objects . . . 9
2.4 Triangles . . . 10
2.5 Axis-aligned boxes . . . 10
2.5.1 Axis-Aligned Bounding Box (AABB) . . . 10
2.5.2 Bounding Volume Hierarchies (BVH) . . . 10
2.6 Iso-surfaces in grids . . . 11

3 Shading **11**
3.1 Phong Illumination Model . . . 11
3.1.1 Diffuse Reflection . . . 12
3.1.2 Specular Reflection . . . 12
3.1.3 Reflection from Ambient Illumination . . . 12
3.2 Extensions . . . 12
3.3 Shading Models . . . 13
3.3.1 Well-knows Models . . . 13

4 Homogeneous Notation **13**
4.1 Transformations . . . 14
4.1.1 Translation . . . 14
4.1.2 Rotation . . . 14
4.1.3 Mirroring . . . 15
4.1.4 Scale . . . 15
4.1.5 Shear . . . 15
4.1.6 Planes and normals . . . 15
4.1.7 Basis Transform . . . 15

5 Projection **16**
5.1 2D projections . . . 16
5.2 3D projections . . . 17
5.3 Projection Transform . . . 17
5.3.1 Modelview Transformation . . . 17
5.3.2 Projection Transformation . . . 18
5.3.3 Perspective Projection Transformation . . . 18
5.3.4 Orthographic Projection . . . 19
5.4 Typical vertex transformations . . . 19

6 Rasterization **20**
6.1 Rasterization-based rendering . . . 21
6.1.1 1. Vertex processing. . . . 21
6.1.2 2. Rasterization. . . . 22
6.1.3 3. Fragment processing. . . . 23
6.1.4 4. Fragment update . . . 24

7 Parametric Curves **24**
7.1 Polynomial curves . . . 24
7.2 Bézier Curves . . . 24
7.3 Bernstein Polynomials - Matrix Notation . . . 25
7.3.1 Conversion from Canonical to Bézier . . . 26
7.4 Curve subdivision . . . 26

8 Differential Curves **26**
8.1 C^k Continuity . . . 27
8.2 Piecewise polynomial curves . . . 27
8.2.1 Cubic Bézier Spline . . . 27
8.2.2 Cubic Polynomial in Canonical Form . . . 27
8.2.3 Cubic Hermite . . . 28
8.2.4 Catmull-Rom Spline . . . 28

9 Particle Fluids **28**
9.1 Particle simulation . . . 28
9.2 Particle motion . . . 29
9.3 Particle forces in a fluid . . . 29
9.4 Smoothed Particle Hydrodynamics SPH . . . 29
9.5 SPH for particle fluids . . . 30
9.5.1 Density . . . 30
9.5.2 Pressure . . . 30
9.6 Neighbor search . . . 30
9.7 Boundary handling . . . 31
9.8 Visualization . . . 31
9.8.1 ISO-surface reconstruction . . . 31

II Image Processing **32**

10 Introduction and Image basics **33**
10.1 Imaging model . . . 33
10.2 Image representation . . . 33
10.3 Quantization . . . 34
10.4 Types of images . . . 34

11 Noise, basic operations and filters **35**
11.1 Gaussian filtering . . . 37
11.1.1 Why to prefer Gaussian to box filter? . . . 37
11.2 Box filter . . . 38
11.3 Recursive filter . . . 38

12 Energy minimization **39**
12.1 Convexity . . . 40
12.2 Jacobi Method . . . 40
12.3 Gauss-Seidel Method . . . 40
12.4 Successive over-relaxation (SOR) . . . 41
12.5 Conjugate gradient (CG) . . . 41
12.6 Multigrid methods . . . 41
12.6.1 Unidirectional (cascadic) . . . 41
12.6.2 Correcting (bidirectional) . . . 41
12.6.3 Full multigrid . . . 42

13 Variational Methods **42**
13.1 Discrete/Continuous energies . . . 42
13.2 Consistency . . . 42

14 Motion Estimation **46**
14.1 Lucas-Kanade Method 46
14.2 Horn-Schunck model 47

15 **49**

Part I
Computer Graphics

1 Introduction Computer Graphics

Modeling: generate, represent geometry.
Rendering: light transposing, delete objects etc.
Simulation: animation, dynamic representation.

Light: energy or photons generated by a source, transported along lines, interacting at surfaces (reflection).

Task

$\boldsymbol{v}_i^t$ at $\boldsymbol{x}_i^t$ → $\boldsymbol{v}_i^{t+\Delta t}$ at $\boldsymbol{x}_i^{t+\Delta t}$

Governing equations

$$\frac{\mathrm{d}\boldsymbol{v}_i^t}{\mathrm{d}t} = \boldsymbol{a}_i^t = -\frac{1}{\rho_i^t}\nabla p_i^t + \nu\nabla^2 \boldsymbol{v}_i^t + \boldsymbol{g} \qquad \frac{\mathrm{d}\boldsymbol{x}_i^t}{\mathrm{d}t} = \boldsymbol{v}_i^t$$

$$\frac{\mathrm{d}\rho_i^t}{\mathrm{d}t} = -\rho_i^t \nabla \cdot \boldsymbol{v}_i^t = 0$$

Numerics

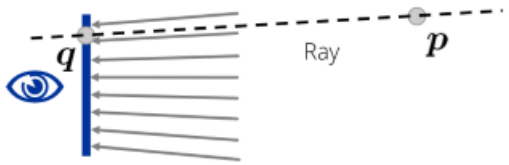
$$\nabla p_i^t \approx \sum_j \frac{m_j}{\rho_j^t} p_j^t \nabla W_{ij}^t \qquad \nabla^2 \boldsymbol{v}_i^t \approx \sum_j \dots$$

$$\boldsymbol{v}_i^{t+\Delta t} = \dots \qquad \boldsymbol{x}_i^{t+\Delta t} = \dots$$

Pressure is computed by solving a pressure Poisson equation.

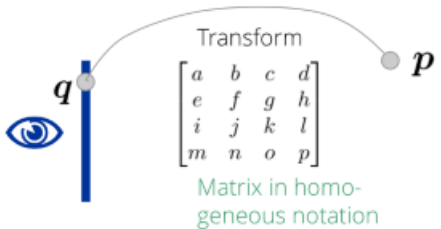
1.1 Rendering aspects

- Ray Tracing:



Ray Tracers compute ray-scene intersections to estimate q from p.

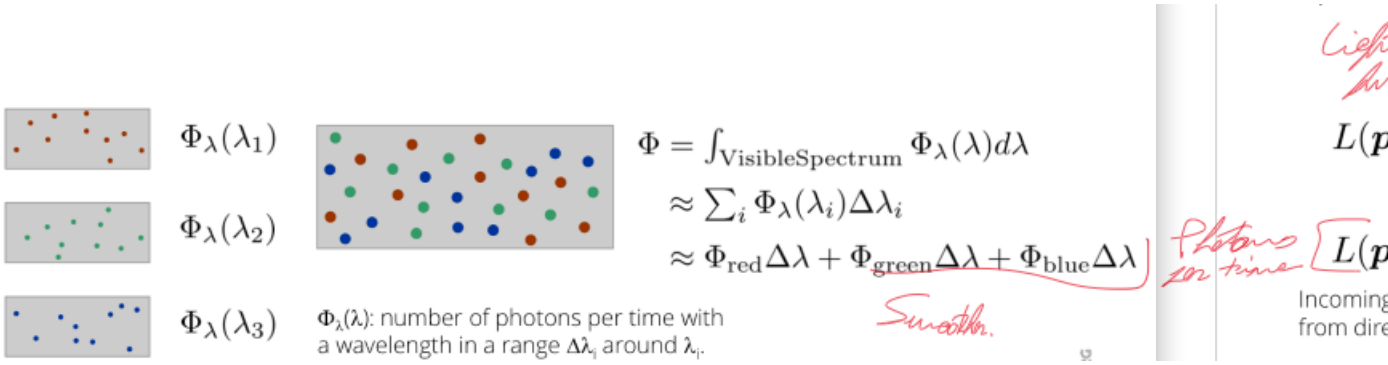
- Rasterization:



Rasterizers apply transformations to p in order to estimate q. p is projected onto the sensor plane.

1.2 Light

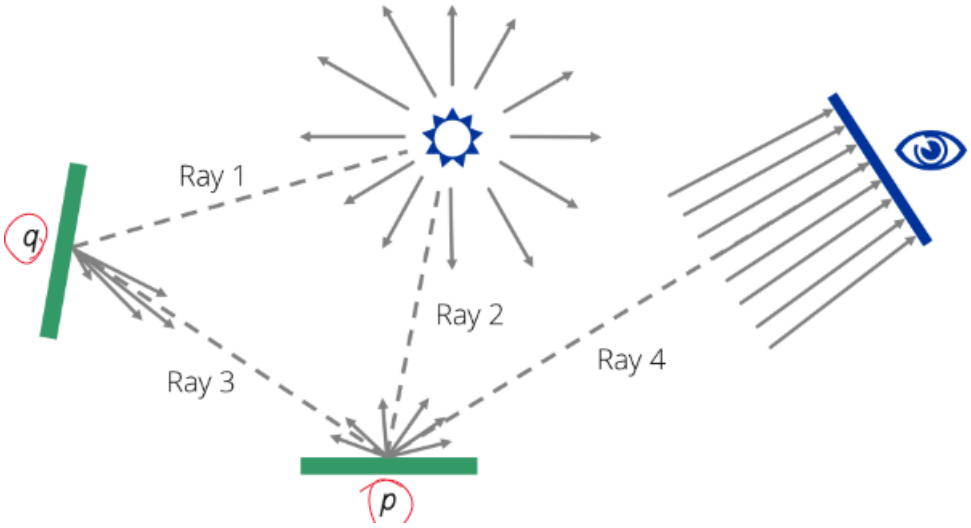
Photons are characterized by a wavelength within the visible spectrum =¿ color.



Rendering -¿ lookup light transported along rays casted into the scene.

2 Ray Casting

Rendering problem: visibility/hidden surface problem, object projection onto sensor plane. (Rat-object intersections with ray casting/tracing).



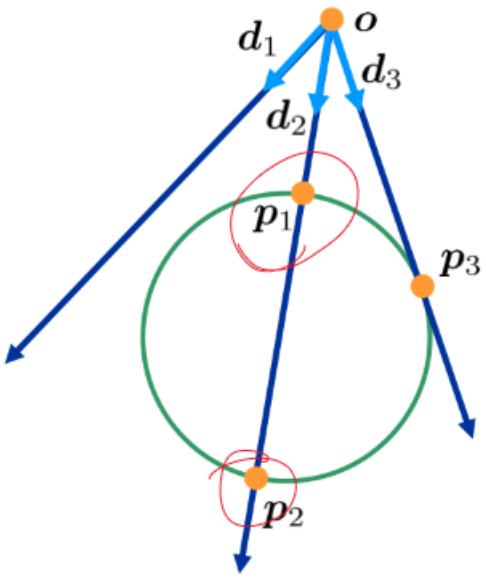
Goal: to compute ray 4 (incoming light at the sensor considering every riflection).
Increasing the number of rays we are considering, we obtain an higher precision (great computationally effort).
Primary rays: start/end at sensors
Secondary rays: don't start/end at sensors
Shadow rays: start/end at sources
Primary solve visibility problem, secondary are useful in order to compute light transport.
Ray casting: computation of position p (what is visible).
Ray tracing: computaiton of light transportd (which colors).
Rendering equations to compue incoming light. **Concept:** a ray is a half-line specified by an origin o and a direction d =¿ r(t) = o+td with t¿=0. We have to compute nearest intersection with all objects.

2.1 Implicit surfaces

If p = x,y,z is a surface point =¿ f(x,y,z) = 0.
Intersection: f(x,y,z) = f(r(t)) = f(o+td) = 0. All point on a normal surface with offset r satisfy that intersection =¿ n*(p-r) = 0. If d is not orthogonal to n, the intersection can be computed: n*(o+td-r)=0, t=[(r-o)*n]/[n*d].
n is given by gradient (function that compute the variation for unit) of the implicit function (n is the direction with the greater value of gradient).

$$\boldsymbol{n} = \nabla f(\boldsymbol{p}) = (\frac{\partial}{\partial x} n_x(x - r_x), \frac{\partial}{\partial y} n_y(y - r_y), \frac{\partial}{\partial z} n_z(z - r_z)) = (n_x, n_y, n_z)$$

For quadrics we use the same knowledge expanded to three dimensions (quadratic equations).



Ray 1: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}_1$

$B^2 - 4AC < 0$

Ray 2: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}_2$

$t_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$

$\boldsymbol{p}_{1,2} = \boldsymbol{o} + t_{1,2}\boldsymbol{d}_2$

Ray 3: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}_3$

$t_3 = \frac{-B}{2A}$

$\boldsymbol{p}_3 = \boldsymbol{o} + t_3\boldsymbol{d}_3$

2.2 Parametric surfaces

Represented by functions with 2D parameters: x = f(u,v), y = g(u,v), z = h(u,v)
Intersetion can be computer from a non-linear system.

$o_x + td_x = f(u,v) \quad o_y + td_y = g(u,v) \quad o_z + td_z = h(u,v)$

Normal vector

$$\boldsymbol{n}(u,v) = \left(\frac{\partial f}{\partial u}, \frac{\partial g}{\partial u}, \frac{\partial h}{\partial u}\right) \times \left(\frac{\partial f}{\partial v}, \frac{\partial g}{\partial v}, \frac{\partial h}{\partial v}\right)$$

Tangent Tangent

Parametric representations are used to render partial objects.

2.3 Combined objects

Compound objects: union of forms.
Constructive Solid Geomtry CSG: combine simple objects to complex geometry using boolean operators.
Estimate and analyze all intersections considering intervals inside objects, works for closed surfaces.

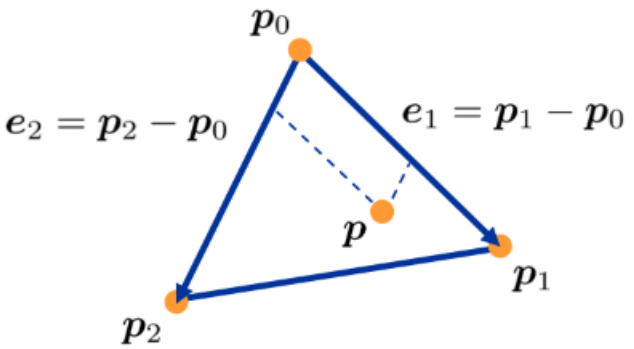
2.4 Triangles

– Parametric representation (Barycentric

$$\boldsymbol{p}(b_1, b_2) = (1 - b_1 - b_2)\boldsymbol{p}_0 + b_1\boldsymbol{p}_1 + b_2\boldsymbol{p}_2$$

$$b_1 \geq 0 \quad b_2 \geq 0 \quad b_1 + b_2 \leq 1$$

Vertices **p** is an ar of the tria



$$\boldsymbol{p} = \boldsymbol{p}_0 + b_1\boldsymbol{e}_1 -$$
$$= \boldsymbol{p}_0 + b_1(\boldsymbol{p}_1$$
$$= (\underbrace{1 - b_1 - b}_{b_0}$$

Popular appoximate surface representation
Intersection: o+td = (1-b1-b2)p0 + b1p1 + b2p2.

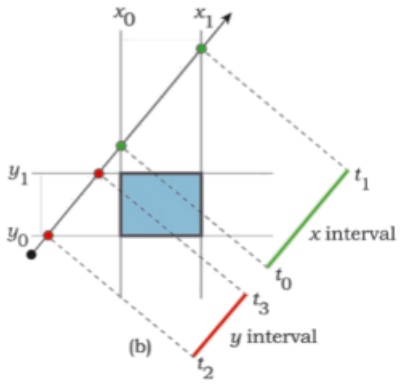
Solution

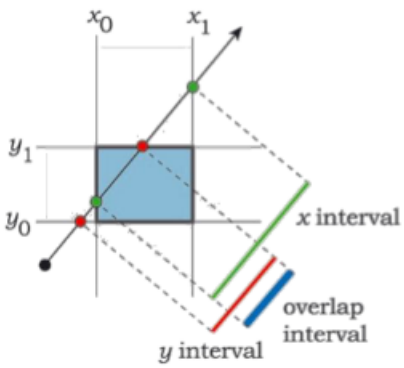
$$\begin{pmatrix} t \\ b_1 \\ b_2 \end{pmatrix} = \frac{1}{(\boldsymbol{d} \times \boldsymbol{e}_2) \cdot \boldsymbol{e}_1} \begin{pmatrix} (\boldsymbol{s} \times \boldsymbol{e}_1) \cdot \boldsymbol{e}_2 \\ (\boldsymbol{d} \times \boldsymbol{e}_2) \cdot \boldsymbol{s} \\ (\boldsymbol{s} \times \boldsymbol{e}_1) \cdot \boldsymbol{d} \end{pmatrix}$$

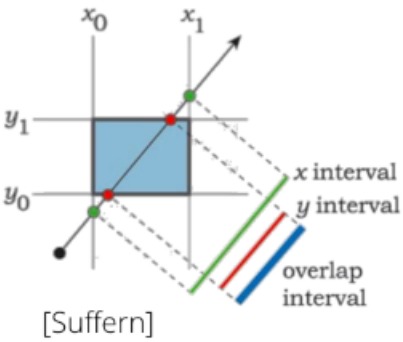
2.5 Axis-aligned boxes

2.5.1 Axis-Aligned Bounding Box (AABB)

Simple geometry outside complex geometry, if a ray doesn't intersecate to the simple, surely doesn't intersect the complex. Boxes are represented by slabs. Intersections of rays are analyzed in order to check for ray-object intersection. Implementations is again done using normal intersection equation. Overlapping ray intervals indicate intrsections.







2.5.2 Bounding Volume Hierarchies (BVH)

AABBs combined in a hierarchy way. put smaller boxes recursively. Efficient pruning but memory and pre-processing overhead.

2.6 Iso-surfaces in grids

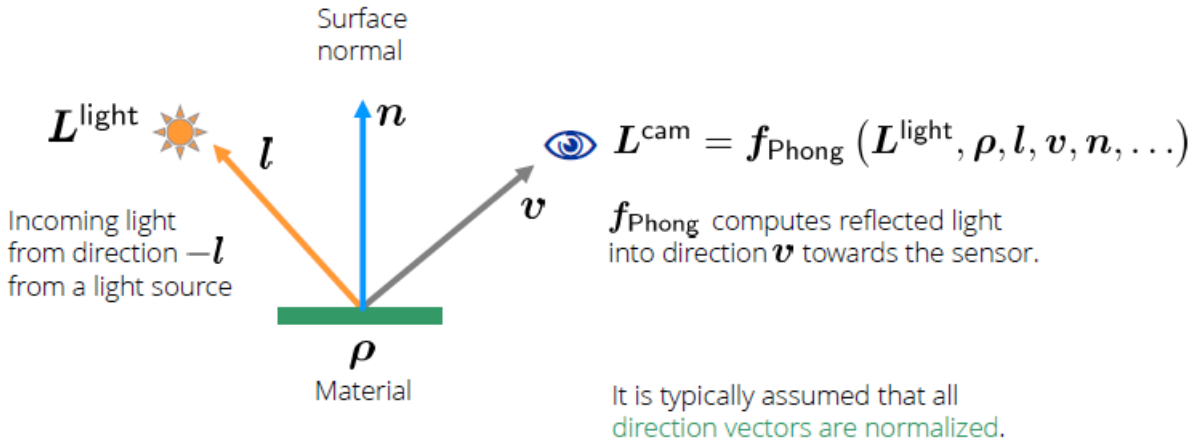
In order to compute intersections on fluid surfaces. Computation is based on density comparison.

Ray casting is very versatile concept to compute what is visible at a sensor but so expensive for complex geometries.

3 Shading

The way to compute color and intensity of what is visible by the sensor. Light is emitted by light sources but is scattered and or absorbed by surface (if dark aborbed, if bright reflected)and media.
Radiance: photons per time * area * solid angle (describe how much light is transported). **Coloured light** is represented by a 3D vector L: (Lred, Lgreen, Lblue). **Coloured objects** are characterized by reflectance coefficient ρ = (ρred, ρgreen, ρblue). Colours that are in the vector are reflected, otherwise absorbed.
We'll use Local Illumination Models that approximately solve the rendering equation.

3.1 Phong Illumination Model



Lsurf is the surface illumination caused by Llight. Lrefl will depend on the object color ρ. Lcam is a portion of Lrefl transported to the sensor (will depend on the material).

Lambert's Cosine Law 1 *Illumination strenght at a surface is proportional to the cosine of the angle between l and n.*

Overall reflected light is

$$L^{refl} = \rho \otimes L^{surf} = \begin{pmatrix} \rho_{red} \cdot L^{surf}_{red} \\ \rho_{green} \cdot L^{surf}_{green} \\ \rho_{blue} \cdot L^{surf}_{blue} \end{pmatrix} = \rho \otimes L^{light} \cdot (n \cdot l)$$

n and *l* have to be normalized. *n*·*l* has to be non-negative.

– Yellow surface under white illumination

$$L^{refl} = \rho \otimes L^{light} \cdot (n \cdot l) = \begin{pmatrix} 1 \cdot 1 \\ 1 \cdot 1 \\ 0 \cdot 1 \end{pmatrix} \cdot (n \cdot l) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \cdot (n \cdot l)$$ Reflects yellow light

– Yellow surface under red illumination

$$L^{refl} = \rho \otimes L^{light} \cdot (n \cdot l) = \begin{pmatrix} 1 \cdot 1 \\ 1 \cdot 0 \\ 0 \cdot 0 \end{pmatrix} \cdot (n \cdot l) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot (n \cdot l)$$ Reflects red light

– Yellow surface under blue illumination

$$L^{refl} = \rho \otimes L^{light} \cdot (n \cdot l) = \begin{pmatrix} 1 \cdot 0 \\ 1 \cdot 0 \\ 0 \cdot 1 \end{pmatrix} \cdot (n \cdot l) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot (n \cdot l)$$ Does not reflect light. Blue is absorbed. Red and green could be reflected, but are not in the light.

Depending on the material, the reflection will be either Diffuse (if Matte) or Specular (if Shiny).

$$L^{\text{cam}} = \alpha \cdot \rho \otimes L^{\text{indirect}} + \sum_i L_i^{\text{light}} \cdot (n \cdot l_i) \otimes \left(\beta \cdot \rho + \gamma \cdot \rho^{\text{white}} \cdot (r_i \cdot v)^m\right)$$

Figure 1: Total model (coefficients are user-defined)

3.1.1 Diffuse Reflection

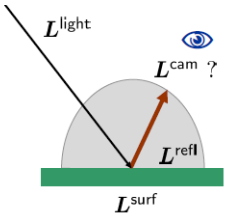
Matte surfaces reflect light equally into all directions.

$L^{\text{refl}} = \int_{2\pi} L(\omega_o) \cos\theta_o \mathrm{d}\omega_o$ Overall reflected light equals reflected light into a direction integrated over all directions

$L(\omega_o) = \text{const} = L^{\text{cam}}$ Definition of diffuse reflection

$L^{\text{refl}} = L^{\text{cam}} \int_{2\pi} \cos\theta_o \mathrm{d}\omega_o = L^{\text{cam}} \cdot \pi$

$L^{\text{cam}} = \frac{1}{\pi} L^{\text{refl}} = \frac{1}{\pi} \cdot \rho \otimes L^{\text{light}} \cdot (n \cdot l)$



3.1.2 Specular Reflection

Shiny surfaces reflect light into a small set of dominant directions.

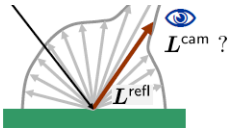
– Light L^{cam} towards sensor

$L^{\text{refl}} = \int_{2\pi} L(\omega_o) \cos\theta_o \mathrm{d}\omega_o$ Overall reflected light equals reflected light into a direction integrated over all directions

$L(\omega_o) \sim (r \cdot \omega_o)^m$ Definition of specular reflection. r is the reflection vector of light direction l with respect to normal n.

$L^{\text{refl}} = \int_{2\pi} k(r \cdot \omega_o)^m \cos\theta_o \mathrm{d}\omega_o$

$L^{\text{cam}} = \rho^{\text{white}} \otimes L^{\text{light}} \cdot (n \cdot l) \cdot (r \cdot v)^m$ *k* is not analyzed in Phong's model. White surface color accounts for the fact that shiny surfaces reflect the entire light spectrum.



Exponent m governs the size of the highlight area. M does not influence the maximal intensity. r can be computed as: r = 2*(l*n)*n-l.
According to the Blinn-Phong (Lrefl from a shiny surface) illumination model r = (l+v)/norma(l+v).

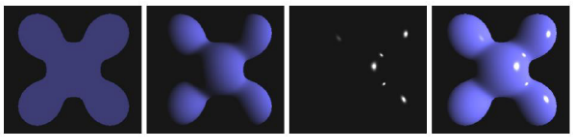
3.1.3 Reflection from Ambient Illumination

Indirect illumination from other surfaces. Reflecte light is: p_*xor*Lindirect.

E.g., red cube in an illuminated room with yellow walls:

$$L^{\text{indirect}} = \begin{pmatrix}1\\1\\0\end{pmatrix} \qquad \rho^{\text{cube}} = \begin{pmatrix}1\\0\\0\end{pmatrix}$$

$$L^{\text{cam}} = \frac{1}{\pi} \cdot \begin{pmatrix}1\\0\\0\end{pmatrix} \otimes \begin{pmatrix}1\\1\\0\end{pmatrix} = \frac{1}{\pi} \cdot \begin{pmatrix}1\\0\\0\end{pmatrix}$$



Lcam = 1/_*pi*p_*xor*Lindirect.
In conclusion Phong is an efficient local computing model and its implementation can be parallelized.Drawbacks: resulting images tend to look less realistic because realistic scenes have more complex illuminations, complex material and non-physical phong parameters cause issues.

3.2 Extensions

Distances must be considered:
1. Between object surface and light source.
- Inverse Square Law: Illumination of a surface decreases quadratically with the distance from a light source =>Lsurf = $(1/r^2) * Llight * (n * l)$.

2. Between object surface and viewer.
- Fog: It is approximated by a combination of Lcam and Cfog. Lcam,fog = f(d)*Lcam+(1-f(d))*Cfog. f(d) describes the visibility.

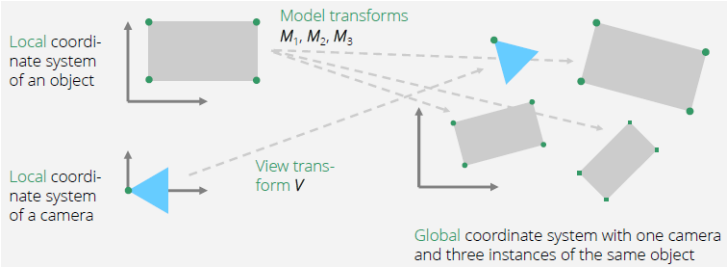
3.3 Shading Models

Illumination models can be evaluated per vertex or per fragment.
Faces/primitives are characterized by vertices.
Projected area of a face onto the sensor is subdivided into fragments.
Shading models specify whether the illumination model is evaluated per vertex or per fragment:
- If evaluated per vertex, the shading model specifies whether the resulting vertex colors are interpolated across a primitive or not.
- If evaluated per fragment, surface normals are interpolated across a primitive.

3.3.1 Well-knows Models

- Flat shading: evaluation per vertex, fragments are colored with the color of one specific vertex.
- Gouraud shading: evaluation per vertex, fragments are interpolated from vertex colors.
- Phong shading: evaluation per fragment, normals have to be interpolated from vertices to fragments.

4 Homogeneous Notation



To transform from view space positions to positions on the camera plane:
- Projection transform
- Viewport transform
Affine trasformatiokns: angles and lengths are not preserved, collinearity, proportions, parallelism are.

3D position $\boldsymbol{p}$: $\boldsymbol{p}' = \boldsymbol{T}(\boldsymbol{p}) = \boldsymbol{A}\boldsymbol{p} + \boldsymbol{t}$

3x3 matrix A represents linear transformations, 3D vector t represents translation. Using the homogeneous notation, all affine transformations are represented with one matrix vector multiplication.
Using the homogeneous notation, transformations of vectors and positions are handled in a unified way.
- From Cartesian to Homogeneous:

$(x, y, z)^\mathsf{T} \rightarrow [x, y, z, 1]^\mathsf{T}$ The most obvious way, but an infinite number of options.

$(x, y, z)^\mathsf{T} \rightarrow [\lambda x, \lambda y, \lambda z, \lambda]^\mathsf{T} \quad \lambda \neq 0$

- From Homogeneous to Cartesian:

$[x, y, z, w]^\mathsf{T} \rightarrow (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^\mathsf{T}$

- General form:

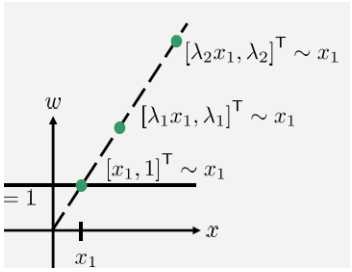


Figure 2: 1D Illustration

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ p_0 & p_1 & p_2 & w \end{bmatrix}$$

m is rotation,scale,shear; t is translation; p for projection; w is the homogeneous component.

4.1 Transformations

4.1.1 Translation

– Of a position

$$\boldsymbol{T}(\boldsymbol{t})\boldsymbol{p} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix}$$

– Of a vector

$$\boldsymbol{T}(\boldsymbol{t})\boldsymbol{v} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

– Inverse transform

$$\boldsymbol{T}^{-1}(\boldsymbol{t}) = \boldsymbol{T}(-\boldsymbol{t})$$

4.1.2 Rotation

Positive (anticlockwise)

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R}_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R}_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.1.3 Mirroring

$$P_x = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad P_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation and reflection matrices are orthogonal: $RR^T = R^T R = I, R^-1 = R^T$

4.1.4 Scale

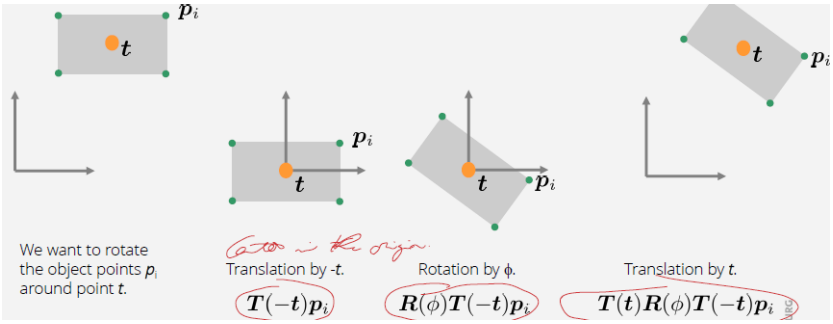
$$S(s_x, s_y, s_z)p = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \\ 1 \end{bmatrix}$$

4.1.5 Shear

Offset of one component wrt another component.

$$H_{xz}(s)p = \begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + sp_z \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

Example:

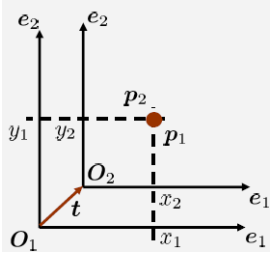


4.1.6 Planes and normals

Planes can be represented by a surface normal n and a point r. All points p with n*(p-r)=0 form a plane.

4.1.7 Basis Transform

- Translation



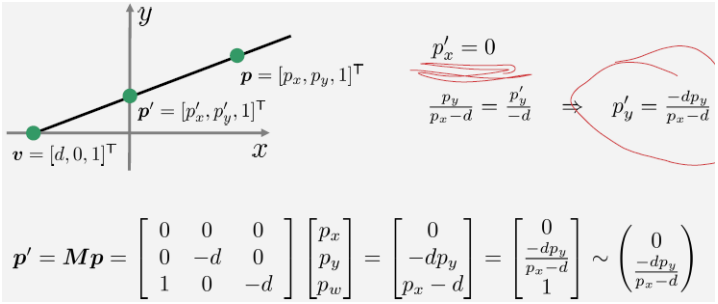


Figure 3: To compute intersection with the x-axis

- Rotation

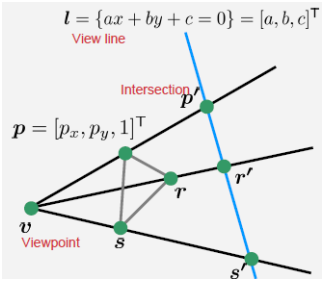
$$\boldsymbol{p}_2 = \begin{pmatrix} \boldsymbol{b}_1^\mathsf{T} \\ \boldsymbol{b}_2^\mathsf{T} \\ \boldsymbol{b}_3^\mathsf{T} \end{pmatrix} \boldsymbol{p}_1 \sim \begin{bmatrix} \boldsymbol{b}_{1,x} & \boldsymbol{b}_{1,y} & \boldsymbol{b}_{1,z} & 0 \\ \boldsymbol{b}_{2,x} & \boldsymbol{b}_{2,y} & \boldsymbol{b}_{2,z} & 0 \\ \boldsymbol{b}_{3,x} & \boldsymbol{b}_{3,y} & \boldsymbol{b}_{3,z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{p}_1$$

The view transform can be seen as a basis transform. Placing and orienting the camera is a transformation v.The basis transform is realized by applying $v^{-}1$ to all objects.
Usage of the homogeneous notation is motivated by a unified processing of affine transformations, perspective projections, points, and vectors.All transformations of points and vectors are represented by a matrix vector multiplication.

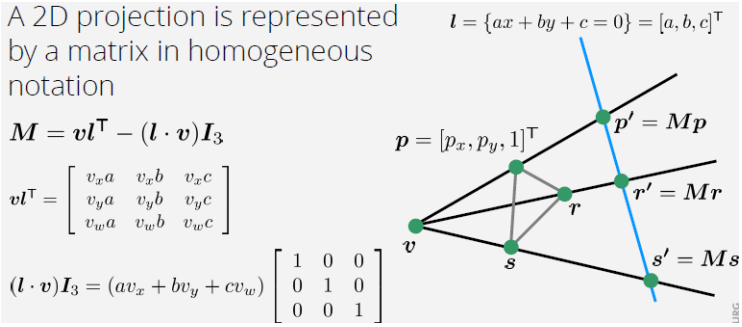
5 Projection

Last matrix M row is dedicated to projections and can be used to realize divisions by a linear combination.

5.1 2D projections



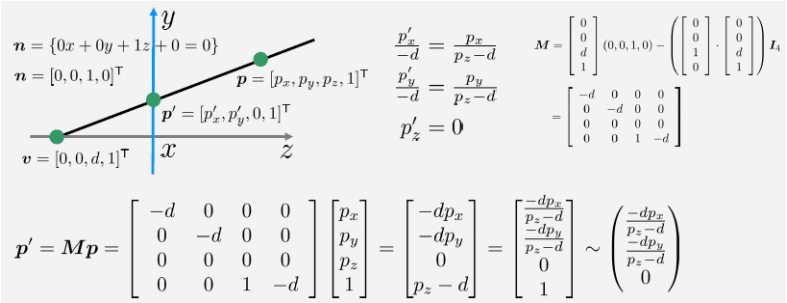
If the homogeneous component of v is not equal to zero, we have a perspective projection; if v is at infinity, we have a parallel projection.
Location of viewpoint and orientation of the viewline determine the type of projection:
- Parallel (viewpoint at infinity, parallel projectors).
- Orthographic (viewline orthogonal to the projectors).
- Oblique (viewline not orthogonal to the projectors).
- Perspective (non parallel projectors):
- One point (viewline intersects one principal axis).
- Two point (viewline intersects two principal axes, two vanishing points).



See other examples on slides.

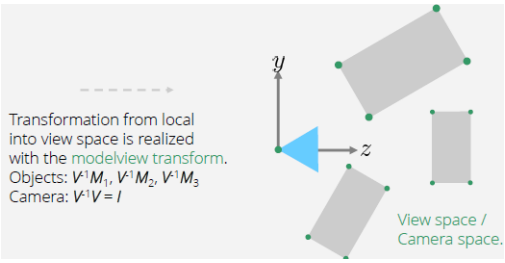
5.2 3D projections

Same thing expanded to 3D.

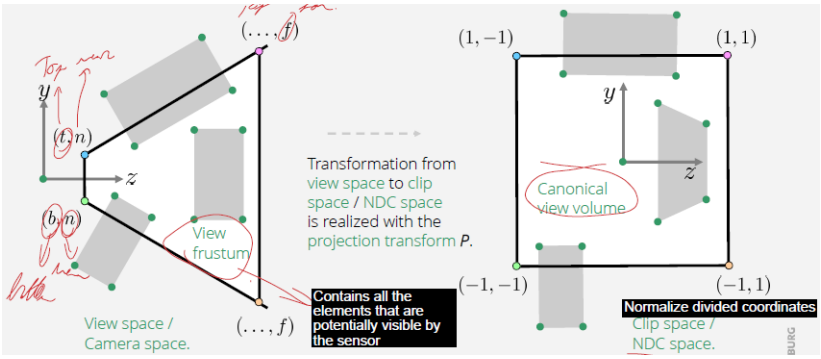


5.3 Projection Transform

5.3.1 Modelview Transformation



5.3.2 Projection Transformation

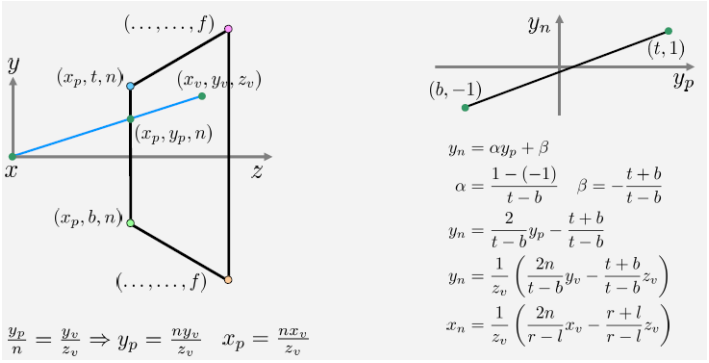


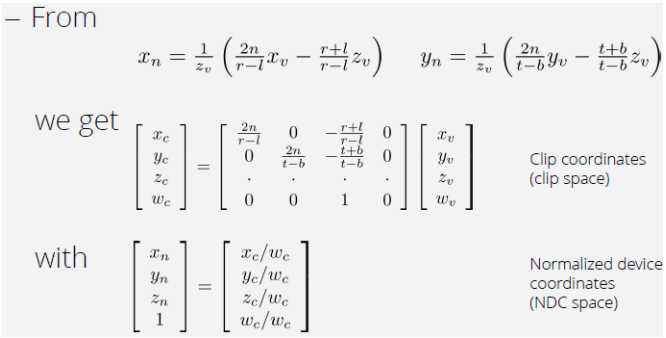
Why to transform to Clip/NDC? It allows simplified and unified implementations:
Culling (exclude non-visible elements), Clipping (exclude non-visible parts of visible elements), Visibility (useful for ray tracing).

5.3.3 Perspective Projection Transformation

Maps a view volume/pyramidal frustum (l,r,t,b) to a canonical view volume (-1,1). It is applied to vertices. Maps coordinates to (-1,1) maintaining coherence.

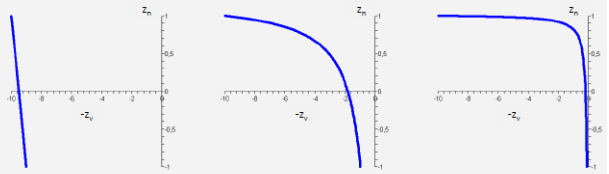
- Derivation





– $z_n = \frac{f+n}{f-n} - \frac{1}{z_v}\frac{2fn}{f-n}$

– Near plane should not be too close to zero



$n = 9 \quad f = 10$ $n = 1 \quad f = 10$ $n = 0.1 \quad f = 10$

Figure 4: Non-linear mapping of depth values

– General form

$$\boldsymbol{P} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– Simplified form for a symmetric view volume

$$r + l = 0$$
$$r - l = 2r$$
$$t + b = 0$$
$$t - b = 2t$$

$$\Rightarrow \boldsymbol{P} = \begin{bmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

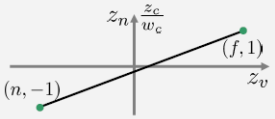
Figure 5: Matrix general form

– z_v is mapped from (near, far) or (n, f) to (-1, 1)

– The transform does not depend on x_v and y_v

– So, we have to solve for A and B in

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_v \\ y_v \\ z_v \\ w_v \end{bmatrix}$$

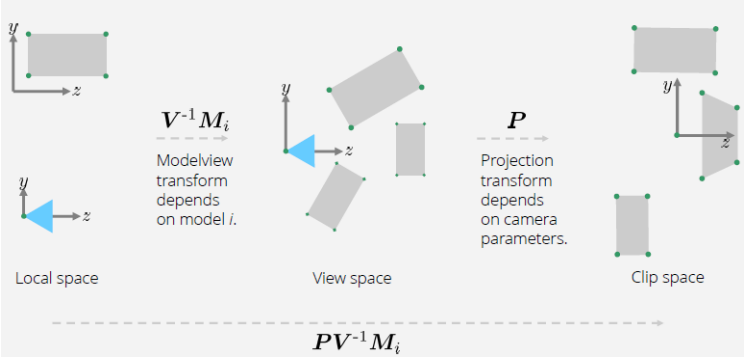


$$z_n = \frac{z_c}{w_c} = \frac{Az_v + Bw_v}{z_v}$$

Then, we can solve A and B and get the complete Projection Matrix

5.3.4 Orthographic Projection

5.4 Typical vertex transformations



Model transform:	Local space	⇨	Global space
View transform:	Local space	⇨	Global space
Inverse view transform:	Global space	⇨	View space
Modelview transform:	Local space	⇨	View space
Projection transform:	View space	⇨	Clip space

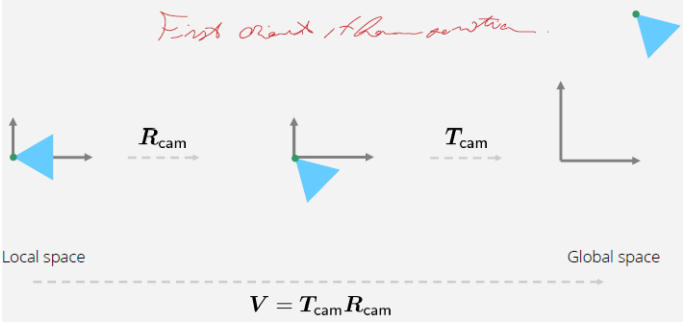


Figure 6: Camera placement

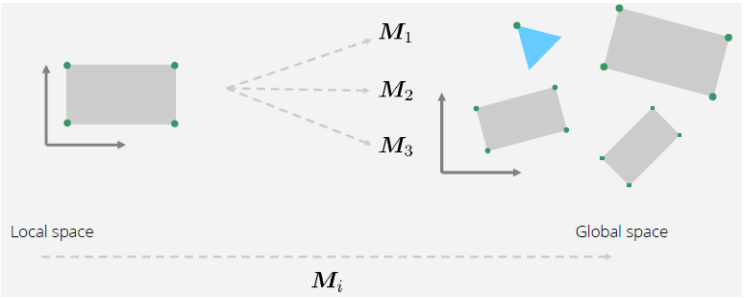


Figure 7: Object Placement

6 Rasterization

As we previously said, visibility can be resolved by ray casting or by rasterization. In particular, the second represent the computation of pixel positions in an image plane that represent a projected primitive. After that, we can solve visibility problem between various objects thanks to pixel positions.

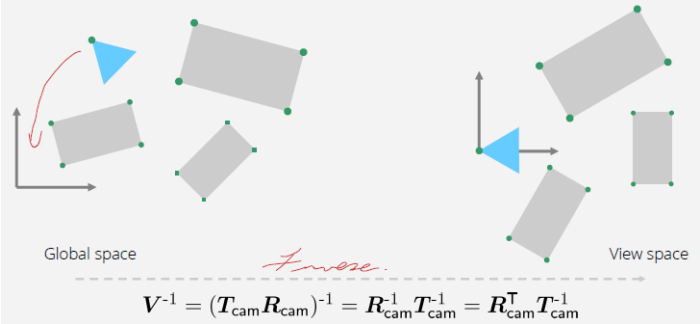


Figure 8: View Transform

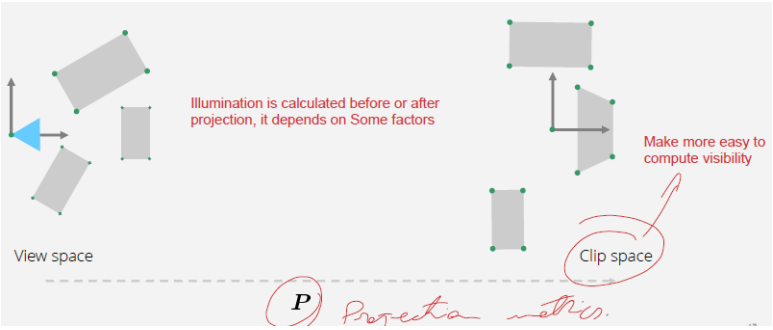
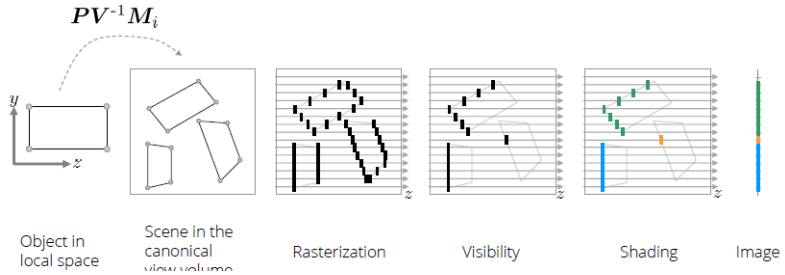


Figure 9: Projection transform



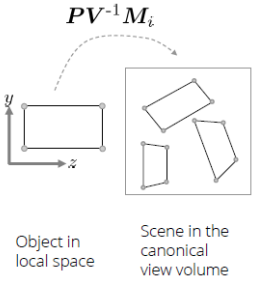
Rasterization is usually composed by three steps: **Rendering pipeline, Rasterization-based rendering, Rasterization.**

6.1 Rasterization-based rendering

- Main stages:

6.1.1 1. Vertex processing.

Scene modelling, placements.

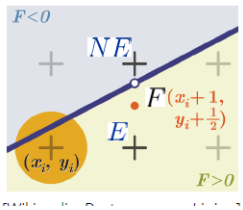


GPU rasterizers assume that all vertex positions are in clip / NDC space.
Only vertices inside the canonical view volume are processed, or the scene can be setup within the canonical view volume and rendered with parallel projection.
Position: Z component in NDC space is referred to as depth value, Color: Phong illumination model can be used, A parameer for rendering effects.

6.1.2 2. Rasterization.

Fragments with attributes from primitives. In the Bresenham Line algorithm for Line rasterization, lines are represented as: F(x,y) = ax+by+c=0.

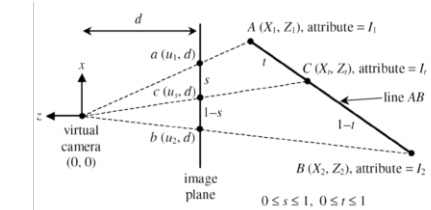
- F is evaluated at the midpoint between (x_i+1, y_i) and (x_i+1, y_i+1)
- $F(x_i+1, y_i+\frac{1}{2}) > 0$ choose NE, i.e. (x_i+1, y_i+1)
- $F(x_i+1, y_i+\frac{1}{2}) \leq 0$ choose E, i.e. (x_i+1, y_i)




```
void BresenhamLine(int xb, int yb, int xe, int ye) {

    int dx, dy, incE, incNE, d, x, y;

    dx = xe - xb; dy = ye - yb;
    d = 2*dy - dx; incE = 2*dy; incNE = 2*(dy - dx);
    x = xb; y = yb;

    GenerateFragment(x, y);

    while (x < xe) {
        x++;
        if (d <= 0)    d += incE;        /* choose E */
        else {d += incNE; y++; }         /* choose NE */
        GenerateFragment(x, y);
    }
}
```

For what concern Polygons, compute intersections of non-horizontal edges with horizontal scanlines: y = y_1 + 0.5. Fill pixel positions in between two intersections with fragments.

Attribute interpolation: from vertices to fragments, anyway linear interpolation in view space cannot be realized by linear interpolation in clip space.



$$I_t = I_1 + t(I_2 - I_1)$$

Linear interpolation in view space

$$I_s = I_1 + \frac{sZ_1}{sZ_1+(1-s)Z_2}(I_2 - I_1)$$

Non-linear interpolation in clip space

$$I_s = \frac{\frac{I_1}{Z_1}+s\left(\frac{I_2}{Z_2}-\frac{I_1}{Z_1}\right)}{\frac{1}{Z_1}+s\left(\frac{1}{Z_2}-\frac{1}{Z_1}\right)}$$

Linear interpolation of I / Z and $1 / Z$ in clip space

w_clip = Z_view

6.1.3 3. Fragment processing.

Fragments attributes are processed and tested (making use of position, color, textures FB data etc.), then can be discarded or pass the test and used to update framebuffer attributes.

- Tests:

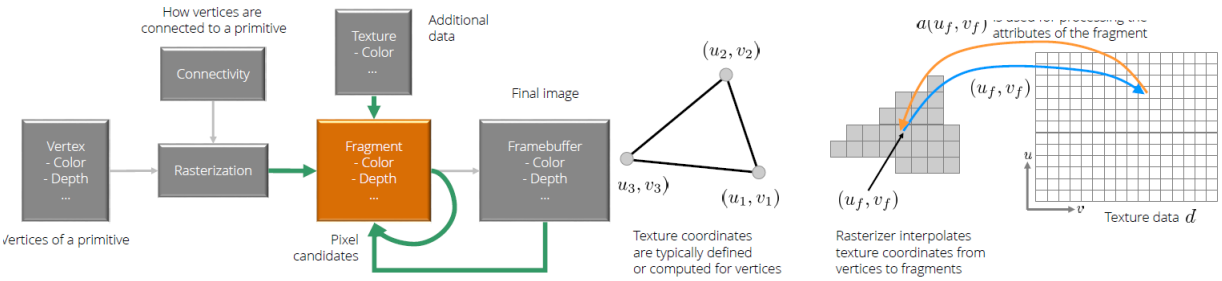
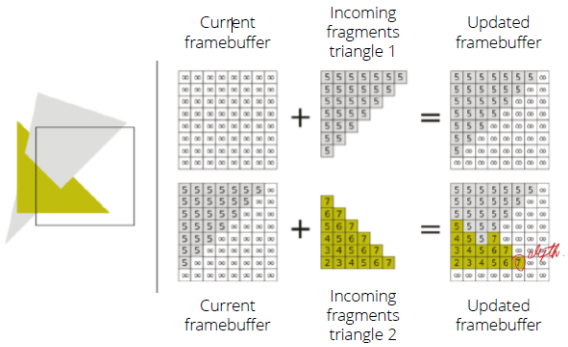


Figure 10: Example of texturing (e.g. Snake skin)

1. Scissor Test: check if fragment position is inside a specified rectangle.
2. Alpha Test: check range of the fragment alpha value. (e.g. for trasparency)
3. Stencil Test: check if FB stencil vlue fulfills a certain requirement.

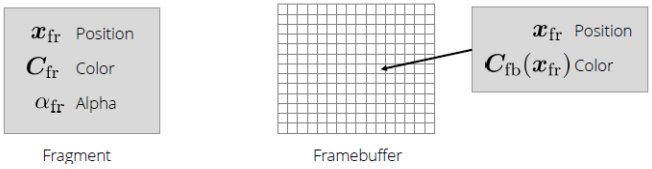
Example: Depth test
Compare fragment depth value with the framebuffer depth value at the fragment position.



6.1.4 4. Fragment update

Combine fragment color with framebuffer color (alpha value is used to set transparency level of images in order to make them mixing):

$$\boldsymbol{C}_{\mathrm{fb}}(\boldsymbol{x}_{\mathrm{fr}}) = \alpha_{\mathrm{fr}} \cdot \boldsymbol{C}_{\mathrm{fr}} + (1 - \alpha_{\mathrm{fr}}) \cdot \boldsymbol{C}_{\mathrm{fb}}(\boldsymbol{x}_{\mathrm{fr}})$$



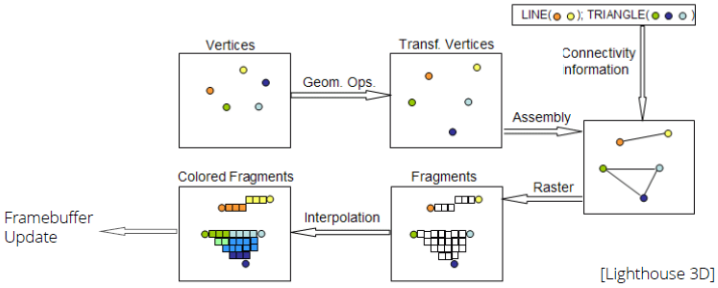
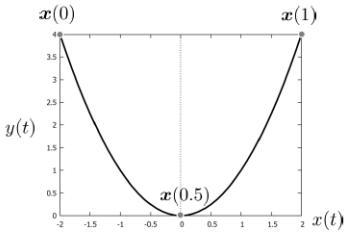


Figure 11: Complete overview.

7 Parametric Curves

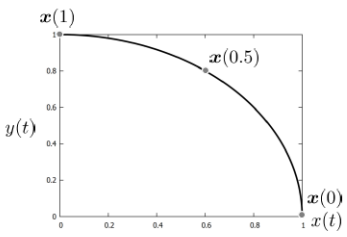
In modelling, Parametric curves are used. Idea: specifying the curve with a small number of control points. It is computed as weighted sum of them (p_i).

7.1 Polynomial curves



$\boldsymbol{x}(t) = (4t - 2, (4t - 2)^2)^\mathsf{T}$

Polynomial curve of degree 2



$\boldsymbol{x}(t) = (\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2})^\mathsf{T}$

Rational curve of degree 2

7.2 Bézier Curves

n+1 control points for degree n. First and last control point are interpolated, others are approximated.

Linear Bézier curve

- Control points: $\boldsymbol{p}_0 = (1,2)^\mathsf{T}$ $\boldsymbol{p}_1 = (3,4)^\mathsf{T}$
- Curve: $\boldsymbol{x}(t) = (1-t)\boldsymbol{p}_0 + t\boldsymbol{p}_1$
  $= (1-t+3t, 2(1-t)+4t)^\mathsf{T} = (1+2t, 2+2t)^\mathsf{T}$

Quadratic Bézier curve

- Control points: $\boldsymbol{p}_0 = (1,2)^\mathsf{T}$ $\boldsymbol{p}_1 = (4,-1)^\mathsf{T}$ $\boldsymbol{p}_2 = (8,6)^\mathsf{T}$
- Curve: $\boldsymbol{x}(t) = (1-t)^2\boldsymbol{p}_0 + 2(1-t)t\boldsymbol{p}_1 + t^2\boldsymbol{p}_2$
  $= (1+6t+t^2, 2-6t+10t^2)^\mathsf{T}$

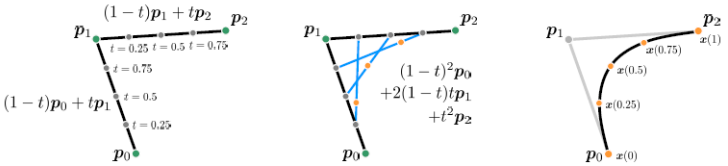
$\boldsymbol{x}(t) = \sum_{i=0}^{n} B_{i,n}(t)\boldsymbol{p}_i \quad t \in [0,1]$

$B_{i,n}(t) = \frac{n!}{(n-i)!i!}(1-t)^{n-i}t^i \qquad 0 \le i \le n$

- Binomial coefficients: $\frac{n!}{(n-i)!i!} = \binom{n}{i}$

$$\begin{array}{ccccccccc}
\binom{0}{0} & & & & & 1 & & & \\
\binom{1}{0} & \binom{1}{1} & & & & 1 & 1 & & \\
\binom{2}{0} & \binom{2}{1} & \binom{2}{2} & & = & 1 & 2 & 1 & \\
\binom{3}{0} & \binom{3}{1} & \binom{3}{2} & \binom{3}{3} & & 1 & 3 & 3 & 1 \\
\binom{4}{0} & \binom{4}{1} & \binom{4}{2} & \binom{4}{3} & \binom{4}{4} & 1 & 4 & 6 & 4 & 1
\end{array}$$

- Interpolation between the interpolation results of two points



- Property:

1. Partition of unity: sum of t is 1.
2. Invariance under affine transformations:

$\boldsymbol{M}\left(\sum_{i=0}^{n} B_{i,n}(t)\boldsymbol{p}_i\right) = \sum_{i=0}^{n} B_{i,n}(t)\boldsymbol{M}\boldsymbol{p}_i$

$\boldsymbol{M}$ is a transformation matrix

$\boldsymbol{p}_i$ are the control points

Endpoint interpolation:

$\boldsymbol{x}(0) = \sum_{i=0}^{n} B_{i,n}(0)\boldsymbol{p}_i = \boldsymbol{p}_0 \qquad \boldsymbol{x}(1) = \sum_{i=0}^{n} B_{i,n}(1)\boldsymbol{p}_i = \boldsymbol{p}_n$

Endpoint tangent:

$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}(0) = n(\boldsymbol{p}_1 - \boldsymbol{p}_0) \qquad \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}(1) = n(\boldsymbol{p}_n - \boldsymbol{p}_{n-1})$

Convex hull:

$\boldsymbol{x}(t) \in \mathrm{CH}(\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n) \quad t \in [0,1]$

$\mathrm{CH}(\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n) = \left\{ \sum_{i=0}^{n} a_i\boldsymbol{p}_i \mid \sum_{i=0}^{n} a_i = 1, a_i \ge 0 \right\}$

7.3 Bernstein Polynomials - Matrix Notation

Quadratic

$$\begin{aligned} B_{0,2}(t) &= (1-t)^2 \\ B_{1,2}(t) &= 2(1-t)t \\ B_{2,2}(t) &= t^2 \end{aligned} \qquad \begin{pmatrix} B_{0,2}(t) \\ B_{1,2}(t) \\ B_{2,2}(t) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{S}_2^{\mathrm{Bez}}} \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix}$$

Cubic

$$\begin{aligned} B_{0,3}(t) &= (1-t)^3 \\ B_{1,3}(t) &= 3(1-t)^2t \\ B_{2,3}(t) &= 3(1-t)t^2 \\ B_{3,3}(t) &= t^3 \end{aligned} \qquad \begin{pmatrix} B_{0,3}(t) \\ B_{1,3}(t) \\ B_{2,3}(t) \\ B_{3,3}(t) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{S}_3^{\mathrm{Bez}}} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

Polynomial bases: canonical: 1,t,t^2,t^3, elements must be linearly independent.

$$(\boldsymbol{S}_3^{\mathrm{Bez}})^{-1} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 \\ 0 & 0 & \frac{1}{3} & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{aligned} 1 &= B_{0,3}(t) + B_{1,3}(t) + B_{2,3}(t) + B_{3,3}(t) \\ t &= \tfrac{1}{3}B_{1,3}(t) + \tfrac{2}{3}B_{2,3}(t) \\ t^2 &= \tfrac{1}{3}B_{2,3}(t) + B_{3,3}(t) \\ t^3 &= B_{3,3}(t) \end{aligned}$$

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} p_0 & p_1 & p_2 & p_3 \\ q_0 & q_1 & q_2 & q_3 \end{pmatrix} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

Curve | Geometry matrix | Spline matrix (Bernstein) | Basis (canonical)

General spline formulation: piecewise polynomial function, x(t) = **GST**(t)

7.3.1 Conversion from Canonical to Bézier

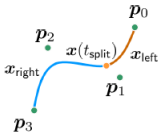
$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix} = \begin{pmatrix} p_0 & p_1 & p_2 & p_3 \\ q_0 & q_1 & q_2 & q_3 \end{pmatrix} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

7.4 Curve subdivision

Given a curve from p_0 to p_3, generate two curves from p_0 to $x(t_{split})$ and from $x(t_{split})$ to p_3 given a value $0 \leq t_{split} \leq 1$.
- Applications: Rendering: *subdivide a curve toward almost linear segments*, Modeling: *modify a part of a curve.*

Use de Casteljau algorithm

$$\begin{aligned} &\boldsymbol{p}_i^0 = \boldsymbol{p}_i \quad i = 0,1,2,3 \\ &\boldsymbol{p}_i^j = (1-t_{\text{split}})\boldsymbol{p}_i^{j-1} + t_{\text{split}}\boldsymbol{p}_{i+1}^{j-1} \quad \boldsymbol{x}(t_{\text{split}}) = \boldsymbol{p}_0^3 \\ &j = 1,2,3 \quad i = 0,\ldots,3-j \end{aligned}$$



$$\boldsymbol{x}_{\text{left}}(t) = B_{0,2}(t)\boldsymbol{p}_0 + B_{1,2}(t)\boldsymbol{p}_1 + B_{2,2}(t)\boldsymbol{p}_2 \qquad t \in [0, t_{\text{split}}]$$

$$\boldsymbol{x}_{\text{left}}(t_l) = B_{0,2}(t_l \cdot t_{\text{split}})\boldsymbol{p}_0 + B_{1,2}(t_l \cdot t_{\text{split}})\boldsymbol{p}_1 + B_{2,2}(t_l \cdot t_{\text{split}})\boldsymbol{p}_2 \qquad t_l \in [0,1]$$

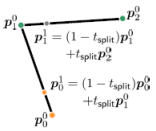
In matrix notation

$$\boldsymbol{x}_{\text{left}}(t_l) = (\boldsymbol{p}_0 \quad \boldsymbol{p}_1 \quad \boldsymbol{p}_2)\, \boldsymbol{S}_2^{\text{Bez}} \begin{pmatrix} 1 \\ t_l \cdot t_{\text{split}} \\ (t_l \cdot t_{\text{split}})^2 \end{pmatrix}$$

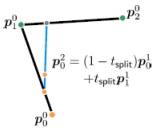
$$\boldsymbol{x}_{\text{left}}(t_l) = \underbrace{(\boldsymbol{p}_0 \quad \boldsymbol{p}_1 \quad \boldsymbol{p}_2)\, \boldsymbol{S}_2^{\text{Bez}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & t_{\text{split}} & 0 \\ 0 & 0 & t_{\text{split}}^2 \end{pmatrix} (\boldsymbol{S}_2^{\text{Bez}})^{-1}}_{(\boldsymbol{p}_{l,0} \quad \boldsymbol{p}_{l,1} \quad \boldsymbol{p}_{l,2})} \boldsymbol{S}_2^{\text{Bez}} \begin{pmatrix} 1 \\ t_l \\ t_l^2 \end{pmatrix}$$

Rewriting the curve with the Bernstein basis functions

$$\begin{aligned} \boldsymbol{p}_{l,0} &= \boldsymbol{p}_0 \\ \boldsymbol{p}_{l,1} &= (1-t_{\text{split}})\boldsymbol{p}_0 + t_{\text{split}}\boldsymbol{p}_1 \\ \boldsymbol{p}_{l,2} &= (1-t_{\text{split}})^2\boldsymbol{p}_0 + 2t_{\text{split}}(1-t_{\text{split}})\boldsymbol{p}_1 + t_{\text{split}}^2\boldsymbol{p}_2 \\ &= (1-t_{\text{split}})\big[(1-t_{\text{split}})\boldsymbol{p}_0 + t_{\text{split}}\boldsymbol{p}_1\big] \\ &\quad + t\big[(1-t_{\text{split}})\boldsymbol{p}_1 + t_{\text{split}}\boldsymbol{p}_2\big] \end{aligned}$$



$$\boldsymbol{p}_{l,0} = \boldsymbol{p}_0^0 \quad \boldsymbol{p}_{l,1} = \boldsymbol{p}_0^1 \quad \boldsymbol{p}_{l,2} = \boldsymbol{p}_0^2$$



$$\begin{aligned} \boldsymbol{x}_{\text{left}}(t) &= (\boldsymbol{p}_0^0 \quad \boldsymbol{p}_0^1 \quad \boldsymbol{p}_0^2)\, \boldsymbol{S}_2^{\text{Bez}} \boldsymbol{T}_2(t) \\ \boldsymbol{x}_{\text{right}}(t) &= (\boldsymbol{p}_0^2 \quad \boldsymbol{p}_1^1 \quad \boldsymbol{p}_2^0)\, \boldsymbol{S}_2^{\text{Bez}} \boldsymbol{T}_2(t) \end{aligned}$$

Right sub-curve derived in the same way.

8 Differential Curves

- Properties: Derivatives, can be considered when connecting polynomials to splines.
- Tangent: Direction of a curve in that point.

Tangents $\boldsymbol{t}(0)$ and $\boldsymbol{t}(1)$

- Linear: $\boldsymbol{t}(0) = \boldsymbol{p}_1 - \boldsymbol{p}_0 \quad \boldsymbol{t}(1) = \boldsymbol{p}_1 - \boldsymbol{p}_0$
- Quadratic: $\boldsymbol{t}(0) = 2(\boldsymbol{p}_1 - \boldsymbol{p}_0) \quad \boldsymbol{t}(1) = 2(\boldsymbol{p}_2 - \boldsymbol{p}_1)$
- Cubic: $\boldsymbol{t}(0) = 3(\boldsymbol{p}_1 - \boldsymbol{p}_0) \quad \boldsymbol{t}(1) = 3(\boldsymbol{p}_3 - \boldsymbol{p}_2)$
- Degree n: $\boldsymbol{t}(0) = n(\boldsymbol{p}_1 - \boldsymbol{p}_0) \quad \boldsymbol{t}(1) = n(\boldsymbol{p}_n - \boldsymbol{p}_{n-1})$

- Velocity: if t is interpreted as time, the derivative is the velocity.
- Acceleration: same but for the second derivative.

General forms

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}(t) = \sum_{i=0}^{n-1} n(\boldsymbol{p}_{i+1} - \boldsymbol{p}_i) B_{i,n-1}(t)$$

$$\frac{\mathrm{d}^2\boldsymbol{x}}{\mathrm{d}t^2}(t) = \sum_{i=0}^{n-2} n(n-1)(\boldsymbol{p}_{i+2} - 2\boldsymbol{p}_{i+1} + \boldsymbol{p}_i) B_{i,n-2}(t)$$

8.1 C^k Continuity

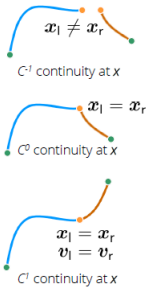
A parametric curve is C^k Continuous if the first k derivatives exists and are continuous.

C^{-1} – continuity
- Curve endpoint positions are not equal

C^0 – continuity
- Curve endpoint positions are *equal*

C^1 – continuity
- Tangent continuity
- C^0 and first derivatives at endpoints are *equal*

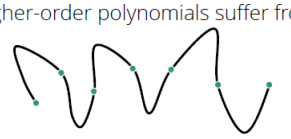
C^2 – continuity
- Curvature continuity
- C^1 and second derivatives at endpoints are *equal*



8.2 Piecewise polynomial curves

Why to connect polynomial curves of lower degree to larger piecewise ones?

Higher-order polynomials suffer from oscillations

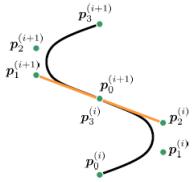


Connect *n*-1 polynomials of lower degree instead

G^k **continuity: (Geometry continuity)** same velocity direction, proportional magnitude.

8.2.1 Cubic Bézier Spline

C^0 continuity: $p_3^{(i)} = p_0^{(i+1)}$. Intermediate control points $p_1^{(i)}$, $p_2^{(i)}$ and $p_1^{(i+1)}$, $p_2^{(i+1)}$ can be used to obtain C^1 continuity.



8.2.2 Cubic Polynomial in Canonical Form

$$
\begin{aligned}
&\boldsymbol{x}^{(i)}(0) = \boldsymbol{p}_i && \frac{\mathrm{d}\boldsymbol{x}^{(i)}}{\mathrm{d}t}(1) = \frac{\mathrm{d}\boldsymbol{x}^{(i+1)}}{\mathrm{d}t}(0)\\
&\boldsymbol{x}^{(i)}(1) = \boldsymbol{p}_{i+1} && \frac{\mathrm{d}^2\boldsymbol{x}^{(i)}}{\mathrm{d}t^2}(1) = \frac{\mathrm{d}^2\boldsymbol{x}^{(i+1)}}{\mathrm{d}t^2}(0)\\
&\boldsymbol{x}^{(i+1)}(0) = \boldsymbol{p}_{i+1} && \frac{\mathrm{d}^2\boldsymbol{x}^{(i)}}{\mathrm{d}t^2}(0) = \boldsymbol{0}\\
&\boldsymbol{x}^{(i+1)}(1) = \boldsymbol{p}_{i+2} && \frac{\mathrm{d}^2\boldsymbol{x}^{(i+1)}}{\mathrm{d}t^2}(1) = \boldsymbol{0}
\end{aligned}
$$

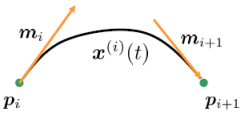
Linear system for unknown coefficients

$$
\begin{pmatrix}
\boldsymbol{I}_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\boldsymbol{I}_2 & \boldsymbol{I}_2 & \boldsymbol{I}_2 & \boldsymbol{I}_2 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & \boldsymbol{I}_2 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & \boldsymbol{I}_2 & \boldsymbol{I}_2 & \boldsymbol{I}_2 & \boldsymbol{I}_2\\
0 & \boldsymbol{I}_2 & 2\boldsymbol{I}_2 & 3\boldsymbol{I}_2 & 0 & -\boldsymbol{I}_2 & 0 & 0\\
0 & 0 & 2\boldsymbol{I}_2 & 6\boldsymbol{I}_2 & 0 & 0 & -2\boldsymbol{I}_2 & 0\\
0 & 0 & 2\boldsymbol{I}_2 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 2\boldsymbol{I}_2 & 6\boldsymbol{I}_2
\end{pmatrix}
\begin{pmatrix}
\boldsymbol{a}_i\\ \boldsymbol{b}_i\\ \boldsymbol{c}_i\\ \boldsymbol{d}_i\\ \boldsymbol{a}_{i+1}\\ \boldsymbol{b}_{i+1}\\ \boldsymbol{c}_{i+1}\\ \boldsymbol{d}_{i+1}
\end{pmatrix}
=
\begin{pmatrix}
\boldsymbol{p}_i\\ \boldsymbol{p}_{i+1}\\ \boldsymbol{p}_{i+1}\\ \boldsymbol{p}_{i+2}\\ \boldsymbol{0}\\ \boldsymbol{0}\\ \boldsymbol{0}\\ \boldsymbol{0}
\end{pmatrix}
$$

8.2.3 Cubic Hermite

Given: $\boldsymbol{x}^{(i)}(0) = \boldsymbol{p}_i$ $\boldsymbol{x}^{(i)}(1) = \boldsymbol{p}_{i+1}$

$\frac{d\boldsymbol{x}^{(i)}}{dt}(0) = \boldsymbol{m}_i$ $\frac{d\boldsymbol{x}^{(i)}}{dt}(1) = \boldsymbol{m}_{i+1}$



How do basis functions H look like that use $\boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \boldsymbol{m}_i, \boldsymbol{m}_{i+1}$ as coefficients?

$$\boldsymbol{x}^{(i)}(t) = \boldsymbol{p}_i H_{0,3}(t) + \boldsymbol{p}_{i+1} H_{1,3}(t) + \boldsymbol{m}_i H_{2,3}(t) + \boldsymbol{m}_{i+1} H_{3,3}(t)$$

One coefficient: $x^{(i)}(t) = a^{(i)} + b^{(i)}t + c^{(i)}t^2 + d^{(i)}t^3$

$\frac{dx^{(i)}}{dt}(t) = b^{(i)} + 2c^{(i)}t + 3d^{(i)}t^2$

Constraints:

$$\begin{aligned}
x^{(i)}(0) = p_i &\Rightarrow a^{(i)} = p_i \\
x^{(i)}(1) = p_{i+1} &\Rightarrow a^{(i)} + b^{(i)} + c^{(i)} + d^{(i)} = p_{i+1} \\
\frac{dx^{(i)}}{dt}(0) = m_i &\Rightarrow b^{(i)} = m_i \\
\frac{dx^{(i)}}{dt}(1) = m_{i+1} &\Rightarrow b^{(i)} + 2c^{(i)} + 3d^{(i)} = m_{i+1}
\end{aligned}$$

$H_{0,3}(t) = 1 - 3t^2 + 2t^3$ $\boldsymbol{p}_0 = (0,0)^\mathsf{T}$

$H_{1,3}(t) = 3t^2 - 2t^3$ $\boldsymbol{p}_1 = (1,0)^\mathsf{T}$

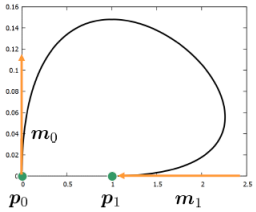
$H_{2,3}(t) = t - 2t^2 + t^3$ $\boldsymbol{m}_0 = (0,1)^\mathsf{T}$

$H_{3,3}(t) = -t^2 + t^3$ $\boldsymbol{m}_1 = (-10,0)^\mathsf{T}$

Basis functions Geometry

$$\boldsymbol{x}^{(i)}(t) = (0,0)^\mathsf{T} + (3t^2 - 2t^3, 0)^\mathsf{T} + (0, t - 2t^2 + t^3)^\mathsf{T} + (10t^2 - 10t^3, 0)^\mathsf{T}$$

Curve



General spline formulation (arbitrary dimension)

$$\boldsymbol{x}^{(i)}(t) = \begin{pmatrix} \boldsymbol{p}_i & \boldsymbol{p}_{i+1} & \boldsymbol{m}_i & \boldsymbol{m}_{i+1} \end{pmatrix} \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

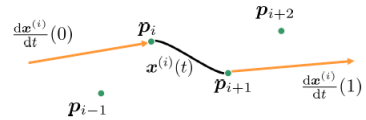
Curve · Geometry matrix · Spline matrix (Hermite) · Basis (canonical)

8.2.4 Catmull-Rom Spline

Variant of the Hermite Spline: formulate derivatives with control points.

Given $\boldsymbol{x}^{(i)}(0) = \boldsymbol{p}_i$ $\boldsymbol{x}^{(i)}(1) = \boldsymbol{p}_{i+1}$

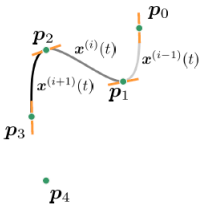
$\frac{d\boldsymbol{x}^{(i)}}{dt}(0) = \frac{1}{2}(\boldsymbol{p}_{i+1} - \boldsymbol{p}_{i-1})$ $\frac{d\boldsymbol{x}^{(i)}}{dt}(1) = \frac{1}{2}(\boldsymbol{p}_{i+2} - \boldsymbol{p}_i)$



– First derivatives are equal at connections

- $\boldsymbol{x}^{(i-1)}(t) = (\boldsymbol{p}_{-1} \quad \boldsymbol{p}_0 \quad \boldsymbol{p}_1 \quad \boldsymbol{p}_2)\, \boldsymbol{S}_3^{\mathrm{CR}} \boldsymbol{T}_3(t)$
- $\boldsymbol{x}^{(i)}(t) = (\boldsymbol{p}_0 \quad \boldsymbol{p}_1 \quad \boldsymbol{p}_2 \quad \boldsymbol{p}_3)\, \boldsymbol{S}_3^{\mathrm{CR}} \boldsymbol{T}_3(t)$
- $\boldsymbol{x}^{(i+1)}(t) = (\boldsymbol{p}_1 \quad \boldsymbol{p}_2 \quad \boldsymbol{p}_3 \quad \boldsymbol{p}_4)\, \boldsymbol{S}_3^{\mathrm{CR}} \boldsymbol{T}_3(t)$

Each curve interpolates between two control points using four control points



Spline formulation

$$\boldsymbol{x}^{(i)}(t) = \begin{pmatrix} \boldsymbol{p}_i & \boldsymbol{p}_{i+1} & \frac{1}{2}(\boldsymbol{p}_{i+1} - \boldsymbol{p}_{i-1}) & \frac{1}{2}(\boldsymbol{p}_{i+2} - \boldsymbol{p}_i) \end{pmatrix} \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

Hermite geometry matrix · Hermite spline matrix

$$\boldsymbol{x}^{(i)}(t) = \begin{pmatrix} \boldsymbol{p}_{i-1} & \boldsymbol{p}_i & \boldsymbol{p}_{i+1} & \boldsymbol{p}_{i+2} \end{pmatrix} \begin{pmatrix} 0 & 0 & -\frac{1}{2} & 0 \\ 1 & 0 & 0 & -\frac{1}{2} \\ 0 & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

Catmull-Rom geometry matrix · Catmull-Rom spline matrix

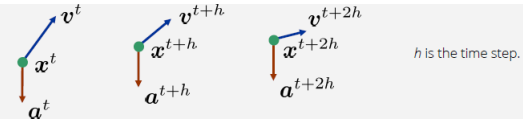
9 Particle Fluids

It is a topic about simulation.

9.1 Particle simulation

Estimate particles properties: x_i^t and v_i^t

9.2 Particle motion



Computation of unknown future particle quantities from known current information.

- Governing Equations for a particle:
Newton's Second Law: describe behavior of x^t and v^t in terms of their time derivative.
Given initial position and velocity, the problem is to estimate the next step (H can be seen as depending on frame rate. Smaller the step, smoother the animation.):

- **Explicit Euler Update:**

$$\boldsymbol{x}^{t+h} = \boldsymbol{x}^t + h\frac{\mathrm{d}\boldsymbol{x}^t}{\mathrm{d}t} + O(h^2) = \boldsymbol{x}^t + h\boldsymbol{v}^t + O(h^2)$$
$$\boldsymbol{v}^{t+h} = \boldsymbol{v}^t + h\frac{\mathrm{d}\boldsymbol{v}^t}{\mathrm{d}t} + O(h^2) = \boldsymbol{v}^t + h\boldsymbol{a}^t + O(h^2)$$

Taylor approximation

- **Verlet (Taylor approximations):**

$$\boldsymbol{x}^{t+h} = \boldsymbol{x}^t + h\boldsymbol{v}^t + \frac{h^2}{2}\boldsymbol{a}^t + \frac{h^3}{6}\frac{\mathrm{d}^3\boldsymbol{x}^t}{\mathrm{d}t^3} + O(h^4)$$
$$\boldsymbol{x}^{t-h} = \boldsymbol{x}^t - h\boldsymbol{v}^t + \frac{h^2}{2}\boldsymbol{a}^t - \frac{h^3}{6}\frac{\mathrm{d}^3\boldsymbol{x}^t}{\mathrm{d}t^3} + O(h^4)$$

9.3 Particle forces in a fluid

- Governing Equations for Fluid:
Particle positions x_i^t and the respective attributes are *transported* with the local fluid velocity v_i^t, $\frac{dx_i^t}{dt} = v_i^t$. Time rate of change of the velocity is governed by the Lagrange form of the Navier-Stokes equation (set of accelerations):

$$\frac{\mathrm{d}\boldsymbol{v}_i^t}{\mathrm{d}t} = -\frac{1}{\rho_i^t}\nabla p_i^t + \nu\nabla^2\boldsymbol{v}_i^t + \frac{\boldsymbol{F}_i^{t,\mathrm{other}}}{m_i}$$

Accelerations

- $-\frac{1}{p_i^t} * gradient * p_i^t$: Acceleration due to pressure differences, proportional to compression, particles are accelerated from areas with high pressure to lower pressure.

– Incompressibility

$$-\frac{1}{\rho}\nabla p = -\frac{1}{\rho}\begin{pmatrix}\frac{\partial p}{\partial x_x}\\ \frac{\partial p}{\partial x_y}\\ \frac{\partial p}{\partial x_z}\end{pmatrix}$$

– Viscosity

$$\nu\nabla^2\boldsymbol{v} = \nu\nabla\cdot(\nabla\boldsymbol{v}) = \nu\nabla\cdot\begin{pmatrix}\frac{\partial v_x}{\partial x_x} & \frac{\partial v_x}{\partial x_y} & \frac{\partial v_x}{\partial x_z}\\ \frac{\partial v_y}{\partial x_x} & \frac{\partial v_y}{\partial x_y} & \frac{\partial v_y}{\partial x_z}\\ \frac{\partial v_z}{\partial x_x} & \frac{\partial v_z}{\partial x_y} & \frac{\partial v_z}{\partial x_z}\end{pmatrix}$$

- $v{*}gradient^2{*}v_i^t$**:** Acceleration due to friction forces between particles with different velocities.

9.4 Smoothed Particle Hydrodynamics SPH

Interpolates quantities at arbitrary positions and approximates the spatial derivatives with a finite number of samples.

Particle positions and velocities are governed by $\frac{\mathrm{d}\boldsymbol{x}_i^t}{\mathrm{d}t} = \boldsymbol{v}_i^t$ and $\frac{\mathrm{d}\boldsymbol{v}_i^t}{\mathrm{d}t} = -\frac{1}{\rho_i^t}\nabla p_i^t + \nu\nabla^2\boldsymbol{v}_i^t + \frac{\boldsymbol{F}_i^{t,\mathrm{other}}}{m_i}$
ρ_i^t, $-\frac{1}{\rho_i^t}\nabla p_i^t$, $\nu\nabla^2\boldsymbol{v}_i^t$ and $\frac{\boldsymbol{F}_i^{t,\mathrm{other}}}{m_i}$ are computed with SPH

Quantity A_i is approximated with a set of known quantities A_j at sample positions x_j: $A_i = \sum_j A_j \frac{m_j}{\rho_j} W_{ij}$

Density computation $\rho_i = \sum_j m_j W_{ij}$

Pressure acceleration $-\frac{1}{\rho_i}\nabla p_i = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W_{ij}$

Viscosity acceleration $\nu\nabla^2 \boldsymbol{v}_i = 2\nu \sum_j \frac{m_j}{\rho_j} \frac{\boldsymbol{v}_{ij}\cdot\boldsymbol{x}_{ij}}{\boldsymbol{x}_{ij}\cdot\boldsymbol{x}_{ij}+0.01h^2}\nabla W_{ij}$

Figure 12: Discretizations

for all *particle* i **do**
find neighbors j — Compute adjacent particles for SPH sums

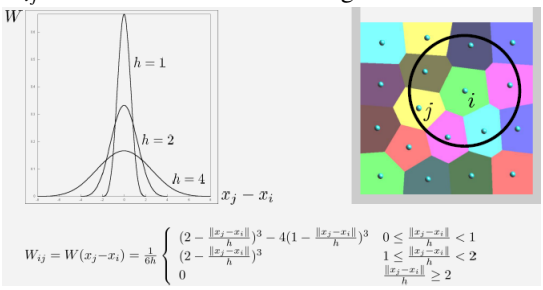
for all *particle* i **do**
$\rho_i = \sum_j m_j W_{ij}$ — Compute density
$p_i = k(\frac{\rho_i}{\rho_0} - 1)$ — Compute pressure

for all *particle* i **do**
$\boldsymbol{a}_i^{\text{nonp}} = \nu\nabla^2 \boldsymbol{v}_i + \boldsymbol{g}$ — Compute non-pressure accelerations
$\boldsymbol{a}_i^{\text{p}} = -\frac{1}{\rho_i}\nabla p_i$ — Compute pressure acceleration
$\boldsymbol{a}_i^t = \boldsymbol{a}_i^{\text{nonp}} + \boldsymbol{a}_i^{\text{p}}$

for all *particle* i **do**
$\boldsymbol{v}_i^{t+\Delta t} = \boldsymbol{v}_i^t + \Delta t \boldsymbol{a}_i^t$ — Explicit Euler for velocity update
$\boldsymbol{x}_i^{t+\Delta t} = \boldsymbol{x}_i^t + \Delta t \boldsymbol{v}_i^{t+\Delta t}$ — Implicit Euler for position update

Figure 13: SPH Fluid Solver

W_{ij} is a kernel function that weights the contributions of sample positions x_j according to their distance to x_i.



$$W_{ij} = W(x_j - x_i) = \frac{1}{6h}\begin{cases} (2 - \frac{\|x_j - x_i\|}{h})^3 - 4(1 - \frac{\|x_j - x_i\|}{h})^3 & 0 \le \frac{\|x_j - x_i\|}{h} < 1 \\ (2 - \frac{\|x_j - x_i\|}{h})^3 & 1 \le \frac{\|x_j - x_i\|}{h} < 2 \\ 0 & \frac{\|x_j - x_i\|}{h} \ge 2 \end{cases}$$

9.5 SPH for particle fluids

9.5.1 Density

Explicit SPH form (Useful for force computation):

$\rho_i = \sum_j \rho_j \frac{m_j}{\rho_j} W_{ij} = \sum_j m_j W_{ij}$

9.5.2 Pressure

Quantifies fluid compression:

$p_i = max(k(\frac{\rho_i}{\rho_0} - 1), 0)$ if the frac is > 1 then there is compression. Pressure acceleration with SPH.

$\boldsymbol{a}_i^{\text{p}} = -\frac{1}{\rho_i}\nabla p_i = -\sum_j m_j \left(\frac{p_i}{\rho^2} + \frac{p_j}{\rho^2}\right)\nabla W_{ij}$

9.6 Neighbor search

Particles are stored in cells, cell size equals the kernel support of a particle. Compute unique cell identifier per particle, sort particles with respect to cell identifier, map cells to a hash table.

for all *particle* i **do**
  find neighbors j — Compute adjacent particles for SPH sums

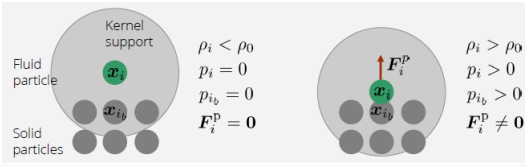
for all *particle* i **do**
  $\rho_i = \sum_j m_j W_{ij}$ — Compute density
  $p_i = k(\frac{\rho_i}{\rho_0} - 1)$ — Compute pressure

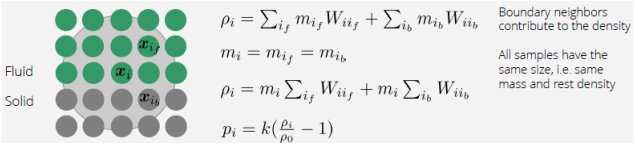
for all *particle* i **do**
  $\boldsymbol{a}_i^{\text{nonp}} = \nu \nabla^2 \boldsymbol{v}_i + \boldsymbol{g}$ — Compute non-pressure accelerations
  $\boldsymbol{a}_i^{\text{p}} = -\frac{1}{\rho_i} \nabla p_i$ — Compute pressure acceleration
  $\boldsymbol{a}_i^t = \boldsymbol{a}_i^{\text{nonp}} + \boldsymbol{a}_i^{\text{p}}$

for all *particle* i **do**
  $\boldsymbol{v}_i^{t+\Delta t} = \boldsymbol{v}_i^t + \Delta t \boldsymbol{a}_i^t$ — Explicit Euler for velocity update
  $\boldsymbol{x}_i^{t+\Delta t} = \boldsymbol{x}_i^t + \Delta t \boldsymbol{v}_i^{t+\Delta t}$ — Implicit Euler for position update

9.7 Boundary handling

Boundaries are sampled with particles that contribute to density, pressure and pressure acceleration.





- Pressure at Boundary Samples:

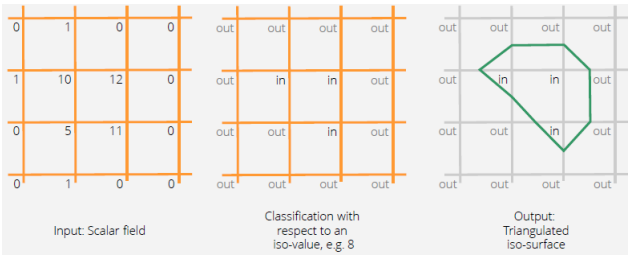
$$\boldsymbol{a}_i^{\text{p}} = -m_i \sum_{i_f} \left(\frac{p_i}{\rho_i^2} + \frac{p_{i_f}}{\rho_{i_f}^2} \right) \nabla W_{ii_f} - m_i \sum_{i_b} \left(\frac{p_i}{\rho_i^2} + \frac{p_{i_b}}{\rho_{i_b}^2} \right) \nabla W_{ii_b}$$

- Mirroring ($p_{ib} = p_i$):

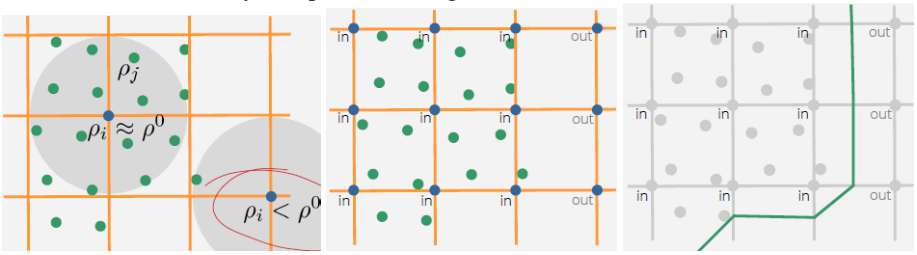
$$\boldsymbol{a}_i^{\text{p}} = -m_i \sum_{i_f} \left(\frac{p_i}{\rho_i^2} + \frac{p_{i_f}}{\rho_{i_f}^2} \right) \nabla W_{ii_f} - m_i \sum_{i_b} \left(\frac{p_i}{\rho_i^2} + \frac{p_i}{\rho_i^2} \right) \nabla W_{ii_b}$$

9.8 Visualization

9.8.1 ISO-surface reconstruction



Initialization: Density computation using SPH.



Classification: $\rho_i \leq 0.5\rho^0$ => **out**, otherwise **in**.

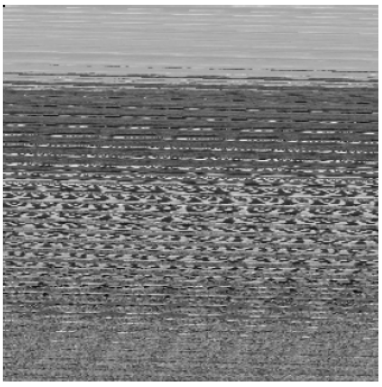
Part II
Image Processing

10 Introduction and Image basics

Digital images: matrix of pixels, defined by an intensity values ($I_{ij} : (\Omega \subset R^2) -> R$).

Matching human visual capabilities means to solve a large part of the AI/ML problem.
Image content is defined by the spatial arrangement of intensities. It is not sufficient to treat images as vectors and to analyze these vectors.

Zebra image

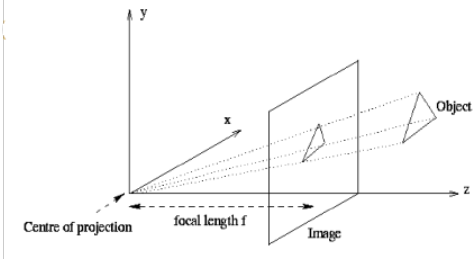


Same image with a different row length

How it works in our brain: The retina significantly reduces the amount of data before it is sent to the brain. In the visual cortex the data is then expanded again. In lower cortical areas (especially V1), neurons respond only to signals in a very small local area (receptive field). In higher cortical areas, the receptive fields get larger and larger due to integration of information from lower areas.

10.1 Imaging model

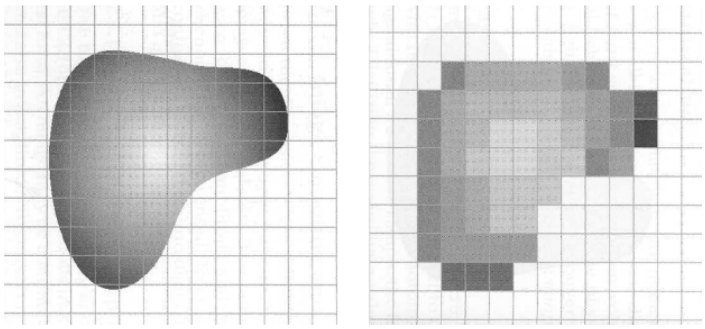
- **Pinhole camera** Objects points (X,Y,Z) are projected to image points (x,y) by $x = f\frac{X}{Z}$, $y = f\frac{Y}{Z}$



- **Optical camera** Large aperture and sharpness possible at a certain depth. Thin lenses: $\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f}$
  Wide lenses: focal length depends on orientation and color.

10.2 Image representation

- **gray value images**
  Image is a continuous function: I: $(\Omega \subset R^2) -> R$. Ω is the image domain (usually rectangular).



Continuous image Discrete, sampled image

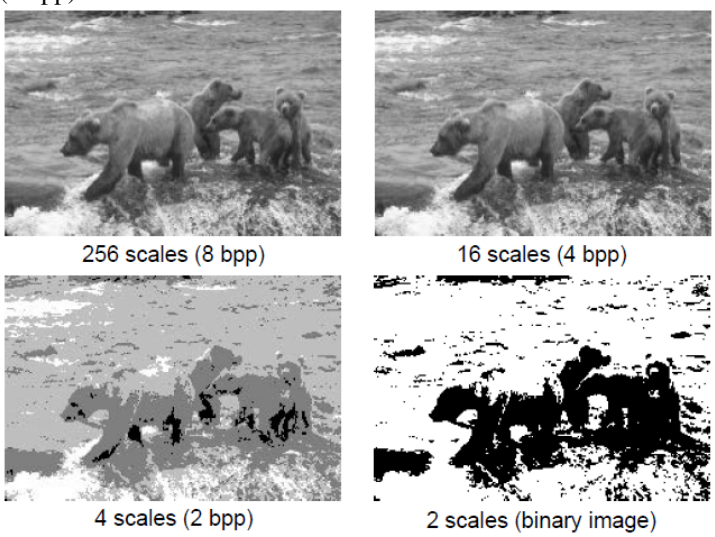
- **Sampling** Intensities are only given on a pixel grid. Grid points are called pixels. If true spacing not known, then grid size of input image set to *h=1*.
  The number of grid points to represent the discrete image is called the **image resolution**. Reducing the number of grid points is called **downsampling**. Increasing the number of grid points is called **upsampling**.
  A discrete signal can only represent frequencies up to a certain limit Nyquist frequency, ignorance of the Nyquist frequency leads to aliasing artifacts (e.g.: straight lines become stepped). **- Nyquist-Shannon theorem**

  > An input signal can be reconstructed from samples in a unique way if the sampling rate is at least two times the bandwidth of the input signal.

- **Downsampling Decanting operator:** ensures minimum smoothing necessary to avoid aliasing. In case of overlap, intensity distribution to both cells according to the overlap ratio.
- **Upsampling Bilinear interpolation:** weighted average of neighboring pixels. Project fine grid point to available coarse grid. Compute weighted average along x-axis: $a_j = (1 - \alpha)I_{i,j} + \alpha I_{i+1,j}$
  Same for y-axis with β. Can be extended to arbitrary dimensions (trilinear interpolation).

10.3 Quantization

To represent a digital image we have to discretize the co-domain: $R- > 1, ..., N$, usual image formats have 256 grey scales (8 bpp).



256 scales (8 bpp) 16 scales (4 bpp)

4 scales (2 bpp) 2 scales (binary image)

10.4 Types of images

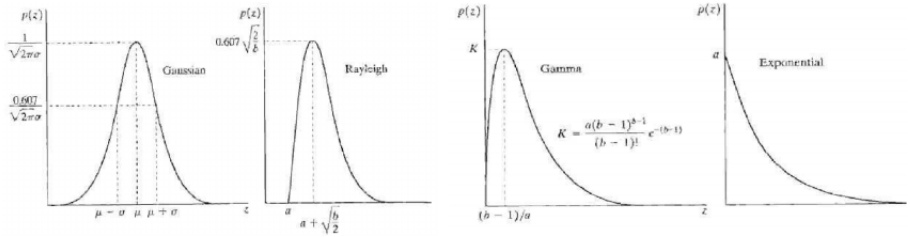
Standard" images: image domain is two-dimensional, co-domain one-dimensional (gray scale).
Matrix-valued images: Diffusion tensor MRI: flow preferences of water molecules, Each voxel (volume element) comprises a 3x3 matrix.

11 Noise, basic operations and filters

Noise is a disturbance of the image data. With actual tools noise anymore exists. Rather than removing noise, make sure the model can deal with noise.

- Types of noise:

- **Additive noise:**
  gray values and noise are independent: $I_{ij} = I_{ij} + n_{ij}$



- **Gaussian noise:**
  Density function: $G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$
  Good approximation in many practical situations, In particular: thermic sensor noise in CCD cameras.

Test image without noise | Gaussian noise with $\sigma = 20$ added

- **Multiplicative noise:**
  Signal dependent: $I_{ij} = I^*_{ij}(1 + n_{ij})$
  More difficult to handle, better transforming the image with log, noise becomes additive noise, apply backtransform to denoised image: $I^*_{ij} = exp \log I^*_{ij}$

- **Impulse noise**
  A certain percentage of pixels is replaced by fixed values, caused, for instance, by pixel defects of CCD chips.

- **Uniform noise**
  A certain percentage of pixels is replaced by uniformly distributed random variables. Very unpleasant noise, no a-priori knowledge in the noise model.

- Signal to Noise Ratio (SNR)

Quantitative measure for the degradation of an image I versus a noise-free version I^* (**ground truth**). Based on the variance of the image versus the variance of the noise.

Signal-to-noise ratio: $\mathrm{SNR} = 10 \log_{10}\left(\frac{\sigma_I^2}{\sigma_n^2}\right) = 10 \log_{10}\left(\frac{\sum_i (I_i^* - \mu)^2}{\sum_i (I_i^* - I_i)^2}\right)$

Peak signal-to-noise ratio: $\mathrm{PSNR} = 10 \log_{10}\left(\frac{N(\max_i I_i^* - \min_i I_i^*)^2}{\sum_i (I_i^* - I_i)^2}\right)$

For small ranges, peak is better. **- Point operations**
The most straightforward way to enhance an image is to treat each pixel independently: $u_{ij} = f(I_{ij})$, Example: changing brightness:
$u(x, y) = I(x, y) + b$, darkening for b¡0.

- Contrast enhancement:
$u(x, y) = aI(x, y)$
$a > 1$, contrast attenuation for $a < 1$, clip values greater than the allowed range.

- Gamma correction:
Cameras have different response curves that human eye: $I \propto$ (is proportional) I^{γ}.
Compensation: $f(I(x, y)) = I_{max}(\frac{I(x,y)}{I_{max}})^{\gamma}$
Dark areas become brighter without leading to oversaturation in the brighter areas.
With γ greater than 1, the image get darker, and viceversa.

The **gray value histogram** contains the number of pixels in the image that have a certain gray value. The equalization is done to have all grey values equally frequent.

- Difference image:
Subtract greyvalues of two subsequent images $\longrightarrow I_{\Delta} = |I_1 - I_2|$. (Can be used for detecting parts of moving objects in static scenes).
- Background substraction:
Take an image of a background, the difference image indicates the object. It can be used for object tracking and segmentation of a person in front of a static background.
$I_{\Delta} = \frac{1}{3}\sum_{k=1}^{3}|I_{k,1} - I_{k,2}|$

Linear filters, convolution theorem
Filters take neighboring pixels into account to improve the signal. Since filters are designed in the Fourier domain we will use **convolution theorem:** $F(f)F(h) = F(f * h)$, with F - Fourier transform. In image processing, filters are diectly designed in Spatial domain because: it is easier to handle and to generalize to nonlinear filters.
x' is the shift, convention of the filter to use -x'.

- **Linear:** $(f * h)(x) = \int h(-x')f(x + x')dx'$
- **2D:** $(I * h)(x, y) = \int h(-x', -y')I(x + x', y + y')dx'dy'$

- Properties:

- **Linearity:** $(\alpha f + \beta g) * h = \alpha(f * h) + \beta(g * h)$
- **Shift invariance:** $f(x) * g(x + \delta) = (f * g)(x + \delta)$
- **Commutativity:** $f * g = g * f$
- **Associativity:** $(f * g) * h = f * (g * h)$
- **Correlation:** $f(x) \star h(x) = \int h(x')f(x + x')dx'$

Discrete convolution in 1D:

$$(f * h)_i = \sum_{k=1}^{n} h_k f_{i-k} \quad i = 1, ..., N$$

Discrete convolution in 2D:

$$(f*h)_{i,j} = \sum_{k=1,l=1}^{n,m} h_{k,l} f_{i-k,j-l} \quad i = 1, ..., N; j = 1, ..., M$$

Separability: A filter is separable if

$$h(x, y) = \delta(x, y) * h_1(x, \cdot) * h_2(\cdot, y)$$

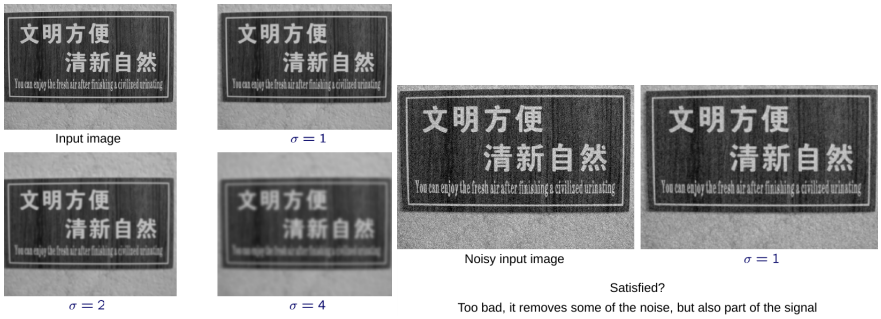
where δ is the Dirac distribution

$$\delta(x) = \begin{cases} \infty & x = 0 \\ 0 & \text{else} \end{cases}, \qquad \int \delta(x)dx = 1$$

Separable filters can be implemented via successive 1D convolutions (associativity of convolution). This reduces the computational complexity of the convolution from $O(NMnm)$ to $O(NM(n + m))$.

11.1 Gaussian filtering

Used for image smoothing, the convolution filter is implemented with a Gaussian kernel of width σ: $G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{x^2}{2\sigma^2})$, the Gaussian filter is separable, the smoothed image can be derived as $I^{\sim}(x, y) = I(x, y) * G_\sigma(x, \dot{}) * G_\sigma(\dot{}, y)$
Complexity: $O(NM\sigma)$, in Fourier domain: $O(NM \log(NM))$



The image domain Ω is generally not an infinite domain but has boundaries $\delta\Omega$.
Boundary types:

- **Dirichlet boundary conditions:** I(x,y)=0
- **Homogeneous Neumann boundary conditions:** $\frac{\delta}{\delta n} I(x, y) = 0$, where n is the outer normal vector on $\delta\Omega$ and $\frac{\delta I}{\delta n} = n^T \nabla I$ is the directional derivative.

Neumann are preferred because they have nicer properties. They are obtained by mirroring the image at the boundary.

11.1.1 Why to prefer Gaussian to box filter?

- **a)** The box filter is not separable
- **b)** The Gaussian filter is rotationally invariant, the box filter is not
- **c)** Of course we prefer a box filter

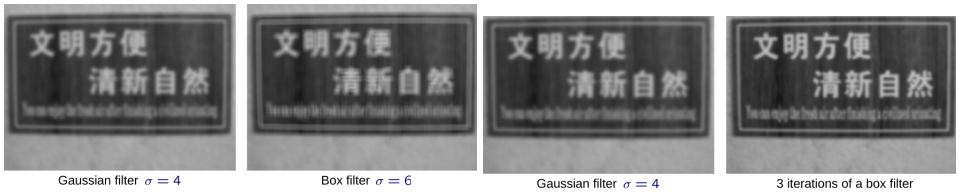
11.2 Box filter

$$B_\sigma(x) = \begin{cases} \frac{1}{2\sigma} & |x| < \sigma \\ 0 & \text{else} \end{cases}$$

Simple averaging of neighbouring values, **but** results are not smooth and not rotationally invariant. The advantage is that convolution with a large kernel is much more efficient, since

$$\tilde{f}_{i+1} = \frac{1}{2\sigma} \sum_{j=i+1-\sigma}^{i+1+\sigma} f_j = \frac{1}{2\sigma} \left(\sum_{j=i-\sigma}^{i+\sigma} f_j + f_{i+1+\sigma} - f_{i-\sigma} \right) = \tilde{f}_i + \frac{1}{2\sigma} \left(f_{i+1+\sigma} - f_{i-\sigma} \right)$$

Complexity O(NM) independent of σ



Gaussian filter $\sigma = 4$ | Box filter $\sigma = 6$ | Gaussian filter $\sigma = 4$ | 3 iterations of a box filter

Anyway, box filters have some relations to Gaussian filter, since iterating the convolutions of the box kernel with itself make the filter similar to gaussian.
- Central limit theorem in statistics: iterated averaging kernels (= positive kernels) converge to Gaussians.

11.3 Recursive filter

Recursively propagate information in both directions of the signal:
α : smoothness parameter

$$f_i = \frac{\left(1 - e^{-\alpha}\right)^2}{1 + 2\alpha e^{-\alpha} - e^{-2\alpha}} \left(I_i + e^{-\alpha}(\alpha - 1) I_{i-1} \right) + 2e^{-\alpha} f_{i-1} - e^{-2\alpha} f_{i-2}$$

$$g_i = \frac{\left(1 - e^{-\alpha}\right)^2}{1 + 2\alpha e^{-\alpha} - e^{-2\alpha}} \left(e^{-\alpha}(\alpha + 1) I_{i+1} - e^{-2\alpha} I_{i+2} \right) + 2e^{-\alpha} g_{i+1} - e^{-2\alpha} g_{i+2}$$

$$\tilde{I}_i = f_i + g_i$$

Recursive filter approximates Gaussian convolution, filter is separable and rotationally invariant. **Relation:** $\alpha\sigma = \frac{5}{2\sqrt{\pi}}$.
Complexity O(NM) is independent of α

Gaussian filter $\sigma = 4$ | Recursive filter with $\sigma = 4$

An important aspect of image processing is the local change of intensities. In continuous functions it is given by the derivative. In multidimensional is the gradient.
Differentiability of images is ensured by combining the derivative with a small amount of Gaussian smoothing $->$ Gaussian derivatives.
The derivative can be applied to the image or to the Gaussian. Gaussian funcions is C^∞, so infinitly differentiable. A common mask for the first derivative is (-1/2, 0, 1/2) (**Central difference**).
Instead for higher order derivatives, use **Laplace filter**, based on second derivatives.

$$\Delta I := \frac{\partial^2 I}{\partial^2 x} + \frac{\partial^2 I}{\partial^2 y} = I_{xx} + I_{yy}$$

Another way to detect edges is by the gradient magnitude (based on first derivatives): $|\nabla I| = \sqrt{I_x^2 + I_y^2}$.

Gradient magnitude

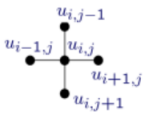
Laplacian
(normalized to a [0,255] range)

12 Energy minimization

- Concept:
1. Formalize your model as an optimization problem: $E(x) = A_1(x)+, ..., +A_n(x)$
2. Solve the optimization problem: $x^* = argmin_x E(x)$
E(x) is called energy (in ML loss funciton).
- Example: image denoising

First step: formulate the model assumptions
- The outcome should be similar to the input image
- The result should be smooth

Second step: formalize these assumptions
- Similarity to the input data (data term):
  $E_D(u_{i,j}) := \sum_{i,j}(u_{i,j} - I_{i,j})^2 \rightarrow \min$
- Similarity to neighboring values (smoothness term):
  $E_S(u_{i,j}) := \sum_{i,j}(u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2 \rightarrow \min$



- Yields an energy minimization problem including a weighting parameter α
  $u^*_{i,j} = \text{argmin}_{u_{i,j}} \left(E_D(u_{i,j}) + \alpha E_S(u_{i,j})\right)$

Third step: solve this optimization problem

- Advantages: transparency, optimality, compatibility.
Global optimization is often hard. Heuristics can obliterate the initial transparency of the model.

$$E(u_{i,j}) = \sum_{i,j}((u_{i,j} - I_{i,j})^2 + \alpha((u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2)) \tag{1}$$

To minimize the function: derivatives must be zero:

$$\frac{\delta E}{\delta u_{i,j}} = 2(u_{i,j} - I_{i,j}) + 2\alpha(u_{i,j} - u_{i-1,j}) - 2\alpha(u_{i+1,j} - u_{i,j}) + 2\alpha(u_{i,j} - u_{i,j-1}) - 2\alpha(u_{i,j+1} - u_{i,j}) = 0 \tag{2}$$

Necessary conditions:

$$\frac{\delta E}{\delta u_{i,j}} = (u_{i,j} - I_{i,j}) + \alpha(u_{i,j} - u_{i-1,j}) - \alpha(u_{i+1,j} - u_{i,j}) + \alpha(u_{i,j} - u_{i,j-1}) - \alpha(u_{i,j+1} - u_{i,j}) = 0 \tag{3}$$

Can be written as:

$$\begin{pmatrix} 1+2\alpha & -\alpha & & -\alpha & & & \\ -\alpha & 1+3\alpha & -\alpha & & -\alpha & & \\ & -\alpha & 1+3\alpha & -\alpha & & -\alpha & \\ -\alpha & & -\alpha & 1+4\alpha & -\alpha & & -\alpha \\ & -\alpha & & -\alpha & 1+3\alpha & -\alpha & \\ & & -\alpha & & -\alpha & 1+3\alpha & -\alpha \\ & & & -\alpha & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ u_{N-1} \\ u_N \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ \cdot \\ \cdot \\ \cdot \\ I_{N-1} \\ I_N \end{pmatrix}$$

$N \times N$ system matrix (for N pixels) is symmetric and positive definite

Since A is Positive definite → the inverse A^{-1} exists and we can solve for u.

12.1 Convexity

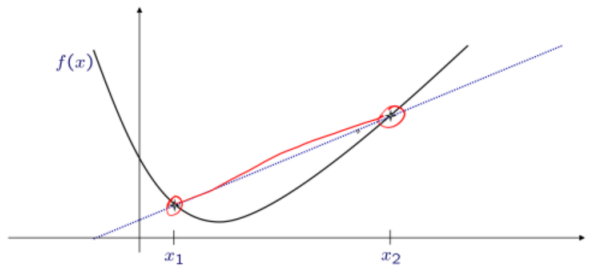
- Convex functions: Positive curvature, global minimum is unique.
- Non-convex functions: Many local minima, global minimum not unique.

A function is **convex** if

$$f((1-\alpha)x_1+\alpha x_2) \leq (1-\alpha)f(x_1)+\alpha f(x_2) \quad \forall x_1, x_2, \forall \alpha \in (0,1)$$

A function is **strictly convex** if

$$f((1-\alpha)x_1+\alpha x_2) < (1-\alpha)f(x_1)+\alpha f(x_2) \quad \forall x_1, x_2, \forall \alpha \in (0,1)$$



Theorem: every convex combination of (strictly) convex functions is again (strictly) convex.
To solve the linear system, an iterative solver is needed

$$\begin{pmatrix} 1+2\alpha & -\alpha & & -\alpha & & & \\ -\alpha & 1+3\alpha & -\alpha & & -\alpha & & \\ & -\alpha & 1+3\alpha & -\alpha & & -\alpha & \\ -\alpha & & -\alpha & 1+4\alpha & -\alpha & & -\alpha \\ & -\alpha & & -\alpha & 1+3\alpha & -\alpha & \\ & & -\alpha & & -\alpha & 1+3\alpha & -\alpha \\ & & & -\alpha & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ \cdot \\ \cdot \\ \cdot \\ I_{N-2} \\ I_{N-1} \end{pmatrix}$$

12.2 Jacobi Method

Decompose A = D+M (D: diagonal part, M: off-diagonal part)
$Ax = b \leftrightarrow (D + M)x = b \leftrightarrow Dx = b - Mx$
D^{-1} can be computed very easily: just replace the diagonal elements by their inverse.
Now compute x iteratively starting by x^0 and: $x^{k+1} = D^{-1}(b - Mx^k)$, until the residual $r^k = Ax^k - b$ is under a threshold.
When is 0 → converged.
- Advantages: simple and parallelizable, **- Disadvantages:** slow, convergence only at ∞, computation not in place.

12.3 Gauss-Seidel Method

Split M into lower triangle and upper:
$Ax = b \leftrightarrow Dx = b - Lx - Ux$
During iteration, use new values with the lower triangle:
$x^{k+1} = D^{-1}(b - Lx^{k+1} - Ux^k)$
In-place computation, recursive propagation of information is faster

12.4 Successive over-relaxation (SOR)

Emphasize the Gauss-Seidel idea by over-relaxing the new solution $x^{k+1} = (1-\omega)x^k + \omega D^{-1}(b - Lx^{k+1} - Ux^k)$
Converges for positive or negative definite matrices, if $\omega \in (0, 2)$. Over relaxation for $\omega > 1$, under for $\omega < 1$.

12.5 Conjugate gradient (CG)

Two vectors are conjugate if they are orthogonal with respect to A: $< u, v >_A = u^T Av = 0$.

$$Ax^* = \alpha_1 Ap_1 +, ..., +\alpha_n Ap_n = b$$

$$p_k^\top Ax^* = p_k^\top \alpha_1 Ap_1 +, ..., +p_k^\top \alpha_n Ap_n = p_k^\top b \quad \text{(expansion with } p_k \text{)}$$

$$\alpha_k = \frac{p_k^\top b}{p_k^\top Ap_k}$$

After n computations we obtain the exact solution x^*

Start with some initial point x^0, Let p_0 be the residual $r_0 = b - Ax^0$. This is the gradient of: $E(x) = \frac{1}{2}x^T Ax - b^T x$ the minimizer of which is x^*
Then, iteratively compute:

$$\alpha_k = \frac{r_k^\top r_k}{p_k^\top Ap_k} \qquad x^{k+1} = x^k + \alpha_k p_k \qquad r_{k+1} = r_k - \alpha_k Ap_k$$

$$\beta_k = \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k} \qquad p_{k+1} = r_{k+1} + \beta_k p_k$$

Stop when residual is small.
A must be symmetric and definite, usually we don't consider exact solution. Sometimes, preconditioners P^{-1} are used to have small condition number for $P^{-1}A$
$Ax = b \leftrightarrow P^{-1}Ax = P^{-1}b$

12.6 Multigrid methods

All previous linear solvers have the drawback that they only act locally due to the sparsity of the matrix. **Idea:** regar the system from a coarser point of view. Faster.

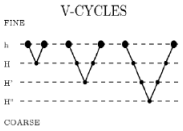
12.6.1 Unidirectional (cascadic)

1. Create downsampled versions of the linear system
2. Compute first approximate solution at the coarse grid (e.g. with SOR)
3. Take upsampled result as initial guess for the next finer grid
4. Refine result there (again with SOR)
Advantages: Fast and simple, **Disadvntages:** Coarse level doesn't approximate so well.

12.6.2 Correcting (bidirectional)

1. Do not downsample the image but the error
2. Compute first solution at fine grid
3. Correct error at coarse grid
4. Refine result at finer grid

1. Presmoothing relaxation step $A^h x^h = b^h$
   - Run some iterations at the fine grid
   - Yields approximate solution $\tilde{x}^h$
   - Remaining error: $e^h = x^h - \tilde{x}^h$

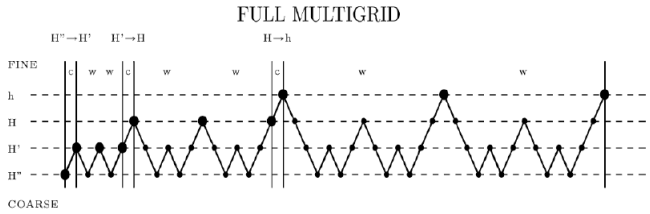


2. Correction step:
   - Goal: compute error $A^h e^h = r^h \qquad r^h = b^h - A^h \tilde{x}^h$
   - Local part of error already removed
     → Solve this system at coarser grid $A^H e^H = r^H$
   - Transfer error to fine grid and correct the solution $\tilde{\tilde{x}}^h = \tilde{x}^h + \tilde{e}^h$

3. Postsmoothing relaxation step
   - Apply some further iterations at fine grid to remove local errors introduced by $\tilde{e}^h$

12.6.3 Full multigrid

1. Combination of cascadic and correcting multigrid
2. Start at coarse grid with downsampled image
3. At each finer level apply a W-cycle



13 Variational Methods

13.1 Discrete/Continuous energies

Continuous formulation: $u^*(x) = argmin_{u(x)} E(u(x))$, with $u(x) : \Omega \rightarrow R$.
The energy function becomes an energy functional, optimization is based on the calculus of variation.
- Advantage of continuous energies:
It is not ensured that discrete energy is consistent with the continuous one.
- Disafvantages: Gradients have to be computed.

13.2 Consistency

Discretizatization of continuous can lead to errors.

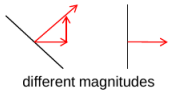
A discretization is called consistent if it converges to the continuous model with finer grid sizes.

Example: discrete approximation of the gradient magnitude

– Inconsistent discretization

$|\nabla u| \approx |u_{i+1,j} - u_{i-1,j}| + |u_{i,j+1} - u_{i,j-1}|$

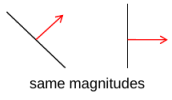
There is an error independent of the grid size



– Consistent discretization

$|\nabla u| \approx \sqrt{(u_{i+1,j} - u_{i-1,j})^2 + (u_{i,j+1} - u_{i,j-1})^2}$

All errors depend on the grid size



In continuous field, to calculate variation, we need Gâteaux derivative.
A functional maps each element u of a vector space to a scalar E(u).
The Gâteaux derivative generalizes the directional derivative to infinite-dimensional vector spaces.

$$\left.\frac{\partial E(u)}{\partial u}\right|_h = \lim_{\epsilon \to 0} \frac{E(u+\epsilon h) - E(u)}{\epsilon} = \left.\frac{dE(u+\epsilon h)}{d\epsilon}\right|_{\epsilon=0}$$

This can be interpreted as the projection of the gradient to the direction h

$$\left.\frac{\partial E(u)}{\partial u}\right|_h = \left\langle \frac{dE(u)}{du}, h \right\rangle = \int \frac{dE(u)}{du}(\mathbf{x}) h(\mathbf{x}) d\mathbf{x}$$

This gradient $\frac{dE(u)}{du}(\mathbf{x})$ is needed to minimize $E(u)$

Back to our Denoising example, in the continuous field, it would be:

$$E(u(\mathbf{x})) = \int_\Omega (u(\mathbf{x}) - I(\mathbf{x}))^2 + \alpha |\nabla u(\mathbf{x})|^2 \, d\mathbf{x}$$

The optimization problem is on the funciton, necessary condition for a minimum:
$\frac{\delta E(u)}{\delta u} h = 0$
Compute the Gâteaux derivative and simplify

$$\begin{aligned}
\left.\frac{\partial E(u)}{\partial u}\right|_h &= \lim_{\epsilon \to 0} \frac{E(u+\epsilon h) - E(u)}{\epsilon} = \left.\frac{dE(u+\epsilon h)}{d\epsilon}\right|_{\epsilon=0} \\
&= \frac{d}{d\epsilon} \int_\Omega (u + \epsilon h - I)^2 + \alpha |\nabla (u + \epsilon h)|^2 \, d\mathbf{x} \Big|_{\epsilon=0} \\
&= \frac{d}{d\epsilon} \int_\Omega (u + \epsilon h - I)^2 + \alpha \left(\frac{d}{dx}(u+\epsilon h)\right)^2 + \alpha \left(\frac{d}{dy}(u+\epsilon h)\right)^2 d\mathbf{x} \Big|_{\epsilon=0} \\
&= \int_\Omega 2(u + \epsilon h - I)h + 2\alpha \left(\frac{d}{dx}(u+\epsilon h)\frac{dh}{dx} + \frac{d}{dy}(u+\epsilon h)\frac{dh}{dy}\right) d\mathbf{x} \Big|_{\epsilon=0} \\
&= \int_\Omega 2(u - I)h + 2\alpha \left(u_x h_x + u_y h_y\right) \, d\mathbf{x}
\end{aligned}$$

Partial integration to turn h_x and h_y into h

$$\begin{aligned}
\left.\frac{\partial E(u)}{\partial u}\right|_h &= \int_\Omega 2(u - I)h - 2\alpha(u_{xx} h + u_{yy} h) \, d\mathbf{x} + (u_x h)_{\partial\Omega_x} + (u_y h)_{\partial\Omega_y} \\
&= \int_\Omega (2(u - I) - 2\alpha(u_{xx} + u_{yy}))h \, d\mathbf{x} + ((\mathbf{n}^\top \nabla u) h)_{\partial\Omega}
\end{aligned}$$

where $\partial\Omega$ denotes the boundary and $\mathbf{n}$ the boundary normal.
In a minimum, the directional derivative must be 0 for all directions

$$\left.\frac{\partial E(u)}{\partial u}\right|_h = \int_\Omega (2(u - I) - 2\alpha(u_{xx} + u_{yy}))h \, d\mathbf{x} + ((\mathbf{n}^\top \nabla u) h)_{\partial\Omega} = 0$$

Yields two conditions:

1. The gradient must be zero → Euler-Lagrange equation:
$$\frac{dE(u)}{du} = 2(u - I) - 2\alpha(u_{xx} + u_{yy}) = 0$$
2. Boundary conditions:
$$(\mathbf{n}^\top \nabla u)_{\partial\Omega} = 0$$

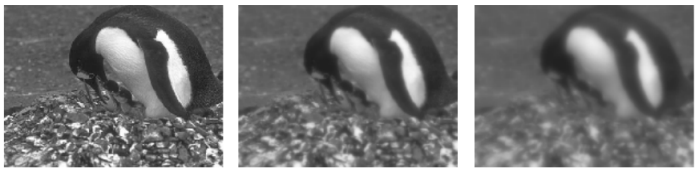
These boundary conditions are called natural boundary conditions as they naturally emanate from the Gâteaux derivative. In the special case here, they coincide with Neumann boundary conditions.
Discretization of

$$\frac{dE(u)}{du} = (u - I) - \alpha(u_{xx} + u_{yy}) = 0$$

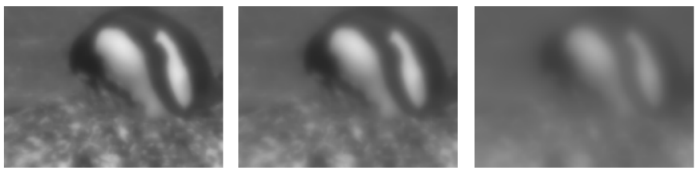
leads to a (in this case) linear system of equations

$$\frac{\partial E}{\partial u_{i,j}} = (u_{i,j} - I_{i,j}) - \alpha(u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1}) = 0$$

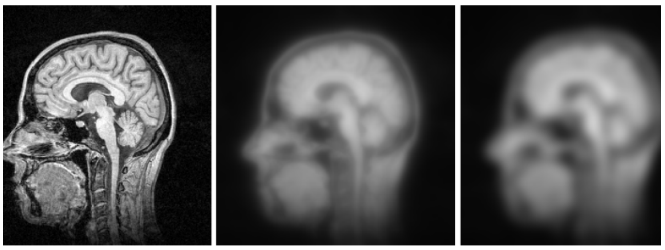
Same linear solvers can be applied. The two different discretization stages do not always lead to the same algorithm.



Input image $\alpha = 1$ $\alpha = 5$



$\alpha = 10$ $\alpha = 20$ $\alpha = 100$



Input image Tikhonov regularization $\alpha = 20$ Gaussian smoothing $\sigma^2 = 40$

A quadratic regularizer leads to blurred edges, so we must do some exceptions (outliers) with non-quadratic regularizers.

- Statistical interpretation

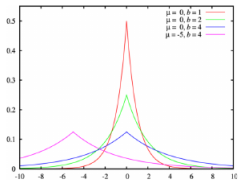
Energy minimization can also emerge from a probabilistic approach taking into account likelihoods and prior probablities: $p(u|I) = \frac{p(I|u)p(u)}{p(I)}$, find u that maximizes (MAP), marginal p(u can be ignored when maximizing).

How is it useful? Turn maximization of the probability into minimization of the negative logarithm → energy minimization problem

$$-\log p(I|u) - \log p(u) = \int_\Omega (u-I)^2 + \alpha|\nabla u|^2 d\mathbf{x} \;\rightarrow\; \min$$

Other noise models lead to different penalizers in the energy functional, excepting outliers we can replace Gaussian by Laplace.

$$p(u) \propto \prod_{\mathbf{x}\in\Omega} \exp\left(-\frac{|\nabla u(\mathbf{x})|}{1/\alpha}\right)$$



Laplace distribution

Leads to an energy functional that preserves image edges

$$E(u) = \int_\Omega (u-I)^2 + \alpha|\nabla u| d\mathbf{x}$$

More general:

$$E(u) = \int_\Omega (u-I)^2 + \alpha\Psi(|\nabla u|^2) d\mathbf{x} \qquad \text{here: } \Psi(s^2) = \sqrt{s^2}$$

(Quadratic) Tikhonov regularizer is another special case: $\Psi(s^2) = s^2$

Corresponding Euler-Lagrange equation (verify at home):

$$\text{div}\left(\Psi'\left(|\nabla u|^2\right)\nabla u\right) - \frac{u-I}{\alpha} = 0$$

where $\Psi'(s^2)$ is the derivative of $\Psi(s^2)$ with respect to its argument.

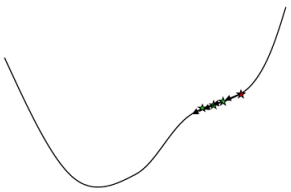
So we get

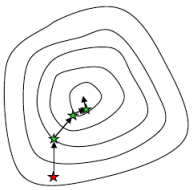
$$\Psi(s^2) = \sqrt{s^2} \quad \Rightarrow \quad \Psi'(s^2) = \frac{1}{2\sqrt{s^2}}$$

$$\text{div}\left(\frac{\nabla u}{2|\nabla u|}\right) - \frac{u-I}{\alpha} = 0$$

The Euler-Lagrange equation is nonlinear in the unknowns, discretization leads to a nonlinear system of equations. A general way to minimize discrete or continuous energies is by gradient descent.

Iterative method: start with an initial value, iteratively move in direction of largest decrease in energy (negative gradient direction).



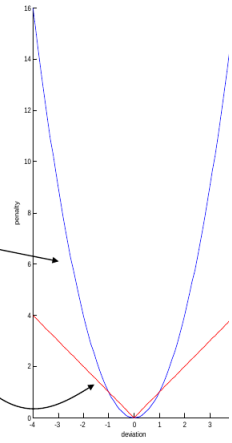


Converges to the next (with regard to the initialization) local minimum

$$E(u) = \int_\Omega (u - I)^2 + \alpha |\nabla u| dx$$

clearly convex

This was close! But yes, convex



Linear combination is convex
→ gradient descent will find a global optimum

- Gradient descent:

Initialize u^0

Negative gradient direction:

$$-\frac{dE(u)}{du} = \text{div}\left(\frac{\nabla u}{2|\nabla u|}\right) - \frac{u - I}{\alpha}$$

Gradient descent: iterative updates with step size τ

$$u^{k+1} = u^k + \tau\left(\text{div}\left(\frac{\nabla u^k}{2|\nabla u^k|}\right) - \frac{u^k - I}{\alpha}\right)$$

Will converge, if τ is "small enough" (more in course Computer Vision)

Set $\Psi' = \frac{1}{\sqrt{|\nabla u|^2 + \epsilon^2}}$, $\epsilon = 0.01$, then $\tau \leq \frac{\epsilon}{4}$

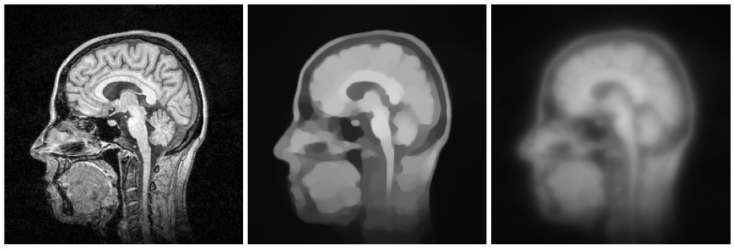
→ slow convergence

$$\text{div}\left(\Psi'\left(|\nabla u|^2\right)\nabla u\right) - \frac{u - I}{\alpha} = 0$$

Keep the nonlinear prefactor fixed (now we have again a linear system) and compute updates in an iterative manner

$$\text{div}\left(\Psi'^k \nabla u^{k+1}\right) - \frac{u^{k+1} - I}{\alpha} = 0$$

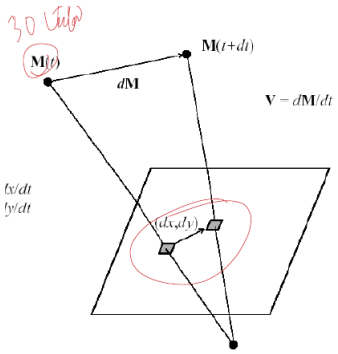
This is the **lagged diffusivity** and has been proven to converge if the linear system in each step is solved exactly. Increased efficiency, Much faster than gradient descent.



Input image | TV regularization $\alpha = 20$ | Tikhonov regularization $\alpha = 20$

14 Motion Estimation

The problem here is to estimate the position of the pixels of an image moving. We seek a vector field (u,v)(x,y,t) **Optical flow field** that describe the motion at t. It can be used for detection or segmentation of objects by comparing the object motion and the background motion → motion segmentation (separate objects from the background), or for 3D motion of objects (**scene flow**). Optical flow can also help estimate the 3D structure of objects. This is called **structure from motion.**



What is perceived, can be different from the reality. The neighborhood (aperture) determines the solution of the optical flow. Therefore, the ambiguity is also called the aperture problem.
Assumption: gray value of a moving pixel is constant: $I(x+u, y+v, t+1) - I(x,y,t) = 0$.
To ease optical flow estimation, we can linearize the first expression with a Taylor expansion:

$$I(x+u, y+v, t+1) = I(x,y,t) + I_x u + I_y v + I_t + O(u^2, v^2),\ I_x \text{ x-derivative of the image.}$$

This leads to the (linearized) optic flow constraint:

$$I_x u + I_y v + I_t = 0$$

14.1 Lucas-Kanade Method

Regarding a single pixel, we cannot uniquely determine its flow vector. There is only one equation, but two unknowns. The Lucas-Kanade method assumes that the motion is constant within a local neighborhood. We then obtain multiple equations for the two unknowns:

$$I_x(x',y',t)u + I_y(x',y',t)v + I_t(x',y',t) = 0,\ \forall x', y' \in R(x,y)$$

It has not a solution anymore, so we just try to minimize the total square error:

$$argmin_{u,v} \sum_{x,y \in R} (I_x(x,y)u + I_y(x,y)v + I_t(x,y))^2$$

The necessary condition for a minimum is:

$$\frac{\partial E}{\partial u} = 2 \sum_{x,y \in \mathcal{R}} (I_x(x,y)u + I_y(x,y)v + I_t(x,y)) I_x(x,y) = 0$$
$$\frac{\partial E}{\partial v} = 2 \sum_{x,y \in \mathcal{R}} (I_x(x,y)u + I_y(x,y)v + I_t(x,y)) I_y(x,y) = 0$$

This leads to a linear system at each pixel

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

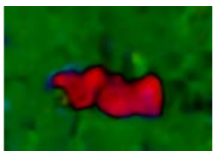
We can also use a Gaussian Window (the bigger the window, the less the reliability). It leads to a convolution:

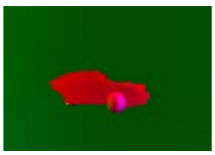
$$\begin{pmatrix} K_\rho * I_x^2 & K_\rho * I_x I_y \\ K_\rho * I_x I_y & K_\rho * I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -K_\rho * I_x I_t \\ -K_\rho * I_y I_t \end{pmatrix}$$

A unique solution will be obtained only if the system is *not singular*(2 non-zero eigenvalues).
- Drawbacks of the Lucas-Kanade method: locally constant motion is not realistic, There are no direct constraints on

the smoothness of the resulting flow field.
- Advantages: fast and simple.


Input sequence


Optical flow estimated with Lucas-Kanade method


Correct flow field

In contrast with Lucas-Kanade, global methods have a global dependency between points due to a global smoothness assumption.

14.2 Horn-Schunck model

A global solution is usually derived with variational methods. Same assumption: gray value constancy leading to the optic flow constraint.

$$I_x u + I_y v + I_z = 0 \tag{4}$$

The second assumption is different. The Horn-Schunck model assumes global smoothness of the flow field:

$$|\nabla u|^2 + |\nabla v|^2 \rightarrow min \tag{5}$$

Both construct the energy functional:

$$E(u,v) = \int_\Omega (I_x u + I_y v + I_z)^2 + \alpha(|\nabla u|^2 + |\nabla v|^2) dxdy \tag{6}$$

That's quadratic so the function is convex (unique optimum). To find the minimizer use calculus of variation.

With the Gâteaux derivatives

$$\frac{d}{d\epsilon} E(u(x) + \epsilon h(x), v(x))\Big|_{\epsilon=0} = 0 \qquad \forall h(x)$$

$$\frac{d}{d\epsilon} E(u(x), v(x) + \epsilon h(x))\Big|_{\epsilon=0} = 0 \qquad \forall h(x)$$

we obtain the Euler-Lagrange equations

$$(I_x u + I_y v + I_z) I_x - \alpha \Delta u = 0$$

$$(I_x u + I_y v + I_z) I_y - \alpha \Delta v = 0$$

Laplacian: $\Delta v = v_{xx} + v_{yy}$

- The discretized versions of these equations read

$$\frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} - \frac{1}{\alpha}(I_x u_{i,j} + I_y v_{i,j} + I_z) I_x = 0$$

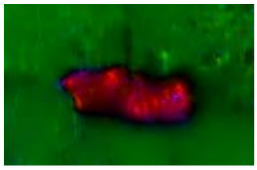
$$\frac{v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{i,j}}{h^2} - \frac{1}{\alpha}(I_x u_{i,j} + I_y v_{i,j} + I_z) I_y = 0$$

This system can be written in matrix-vector notation: $A^h w = b$, where $w = ()u, v$. As in the denoising case, the system matrix is sparse with only few diagonals different from zero.

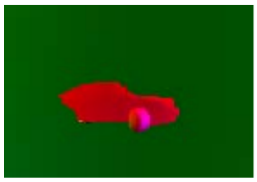
$$A^h = \left(\begin{array}{cc|cc} d & & d & \\ & d & & d \\ \hline d & & d & \\ & d & & d \end{array} \right) + \left(\begin{array}{cc|cc} s & s & s & \\ s & s & s & s \\ \hline s & & s & s \\ & s & s & s \end{array} \right)$$

The data term contributes only to block main diagonals. The smoothness term has four additional block off-diagonals.

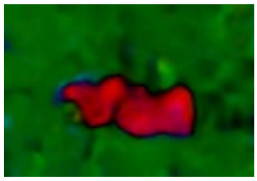
Input sequence



Horn-Schunck (10.4°)



Ground truth



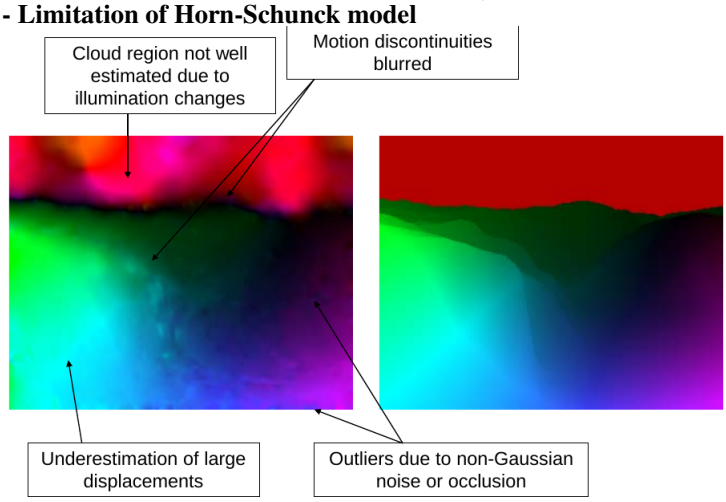
Lucas-Kanade (10.54°)

An established measure of how accurately the optical flow has been estimated is the **average angular error**. The average angular error measures the 3D angle between the correct and the estimated flow vectors:

$$AAE = \frac{1}{n}\sum_{i=1}^{n} arccos\left(\frac{(u_c)_i u_i + (v_c)_i v_i + 1}{\sqrt{((u_c)_i^2 + (v_c)_i^2 + 1)(u_i^2 + v_i^2 + 1)}}\right) \tag{7}$$

An alternative is the end point error: $\mathrm{EPE} = \frac{1}{n}\sum_{i=1}^{n} \sqrt{(u_i - (u_c)_i)^2 + (v_i - (v_c)_i)^2}$

- Limitation of Horn-Schunck model



15