TECHNISCHE UNIVERSITÄT DRESDEN

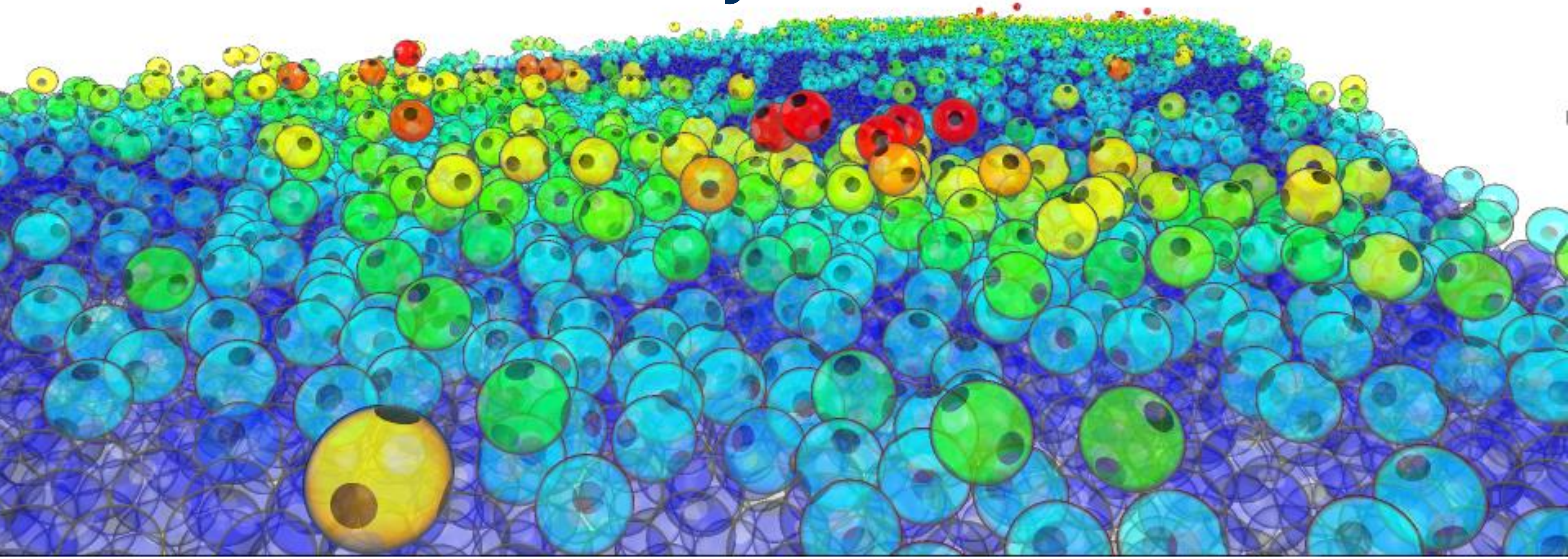
Computer Graphics and Visualization

Scientific Visualization

Part II – Particles

1. Glyphs & Tubes

2. Many & Derived Surface



Content

Part II – Particles

1. Glyphs & Tubes

- Particle Data
  - Sources
  - Basics
  - Clusters
- Shader Based Particle Raycasting
  - Spheres
  - Cylinders
  - Arrows
- Tube Based Visualization

2. Many & Derived Surface

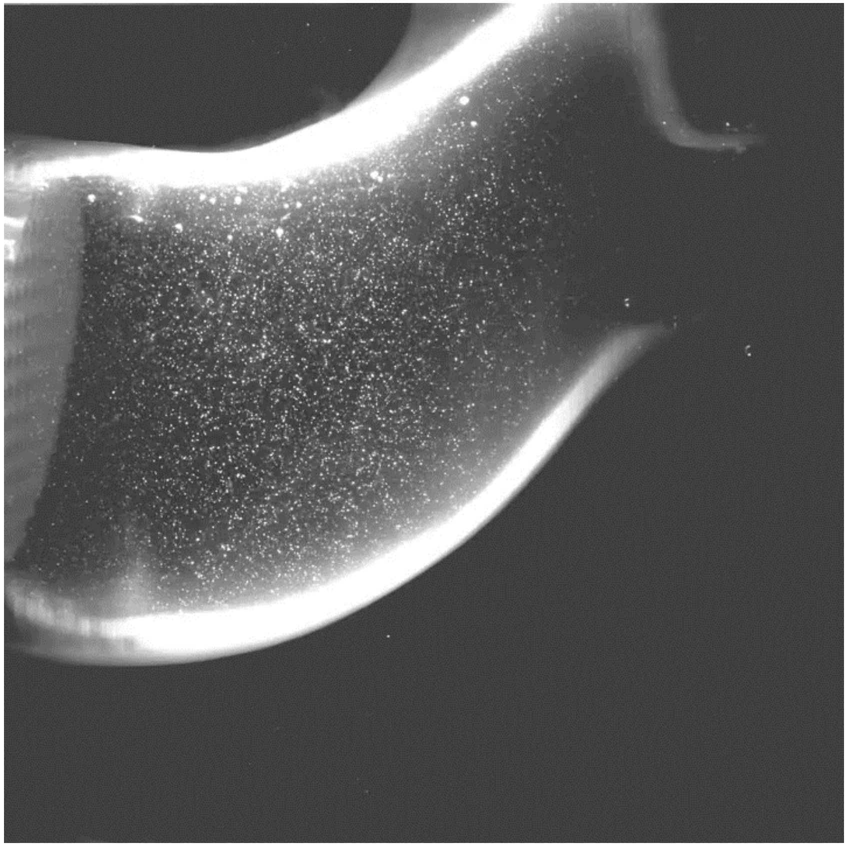
Sources and Types

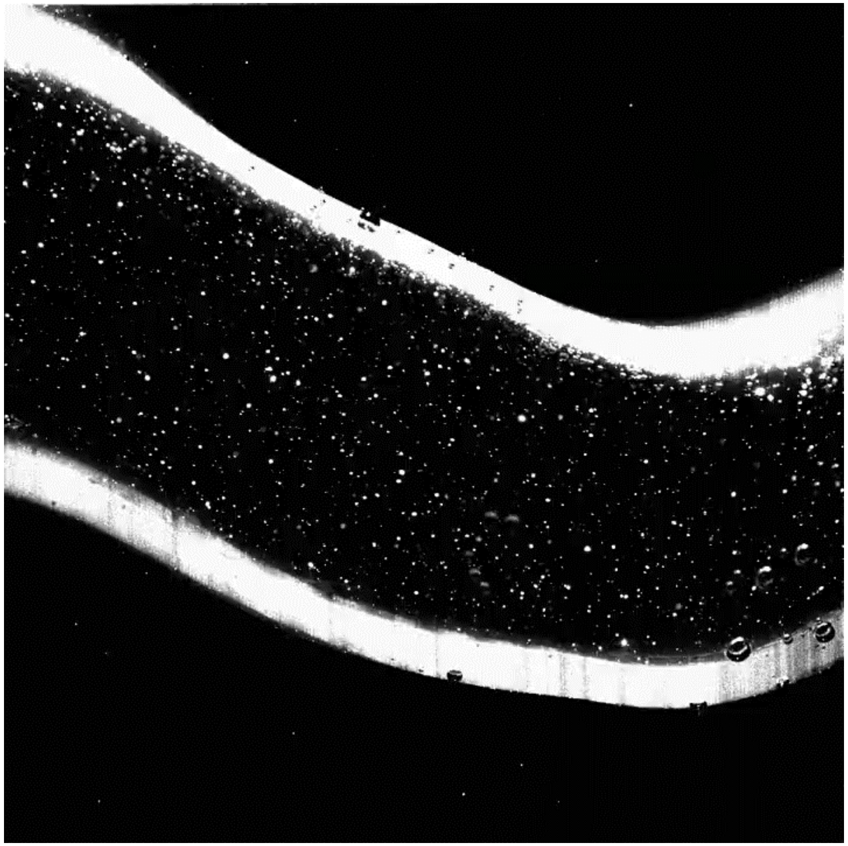
PARTICLE DATA

Particle Data – Sources

Particle tracking velocimetry

- measurement of 3D particle trajectories in fluid with multi-camera setup



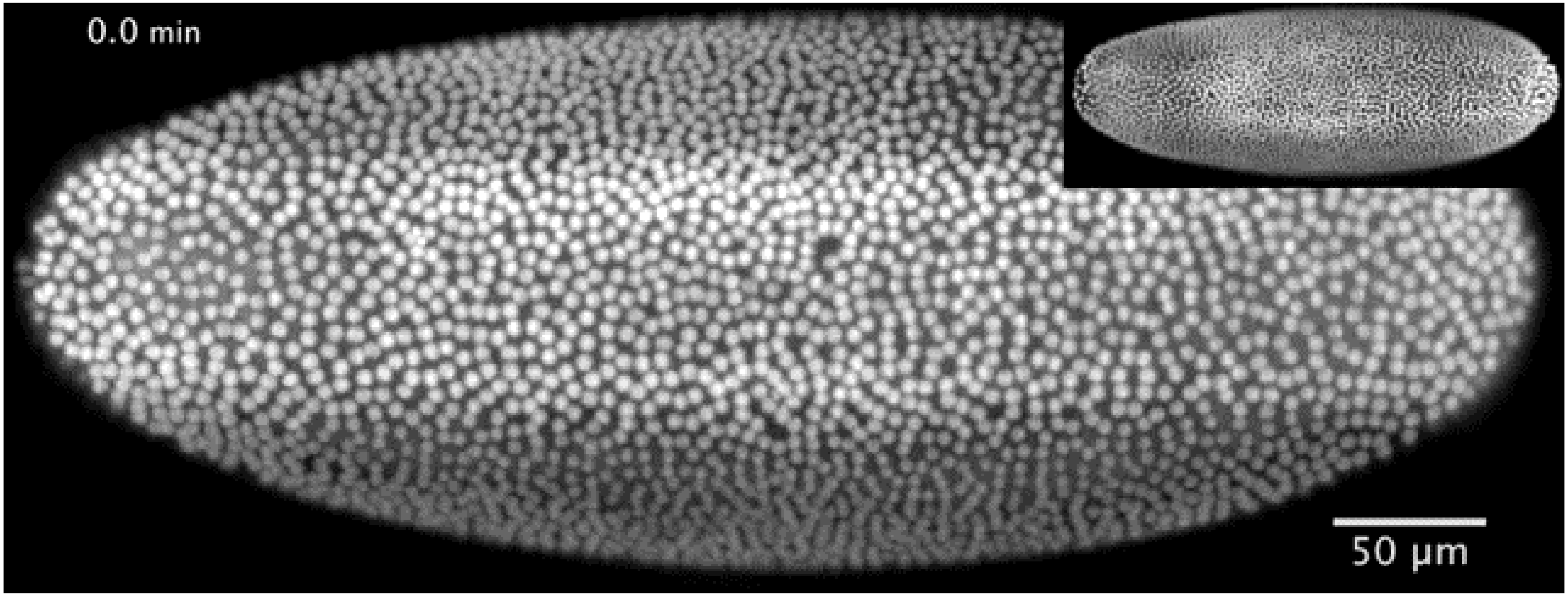


https://www.openptv.net/

Particle Data – Sources

Cell-Tracking in Microscopy

- Analysis of 3D Microscopy video data with cell tracking approaches from computer vision

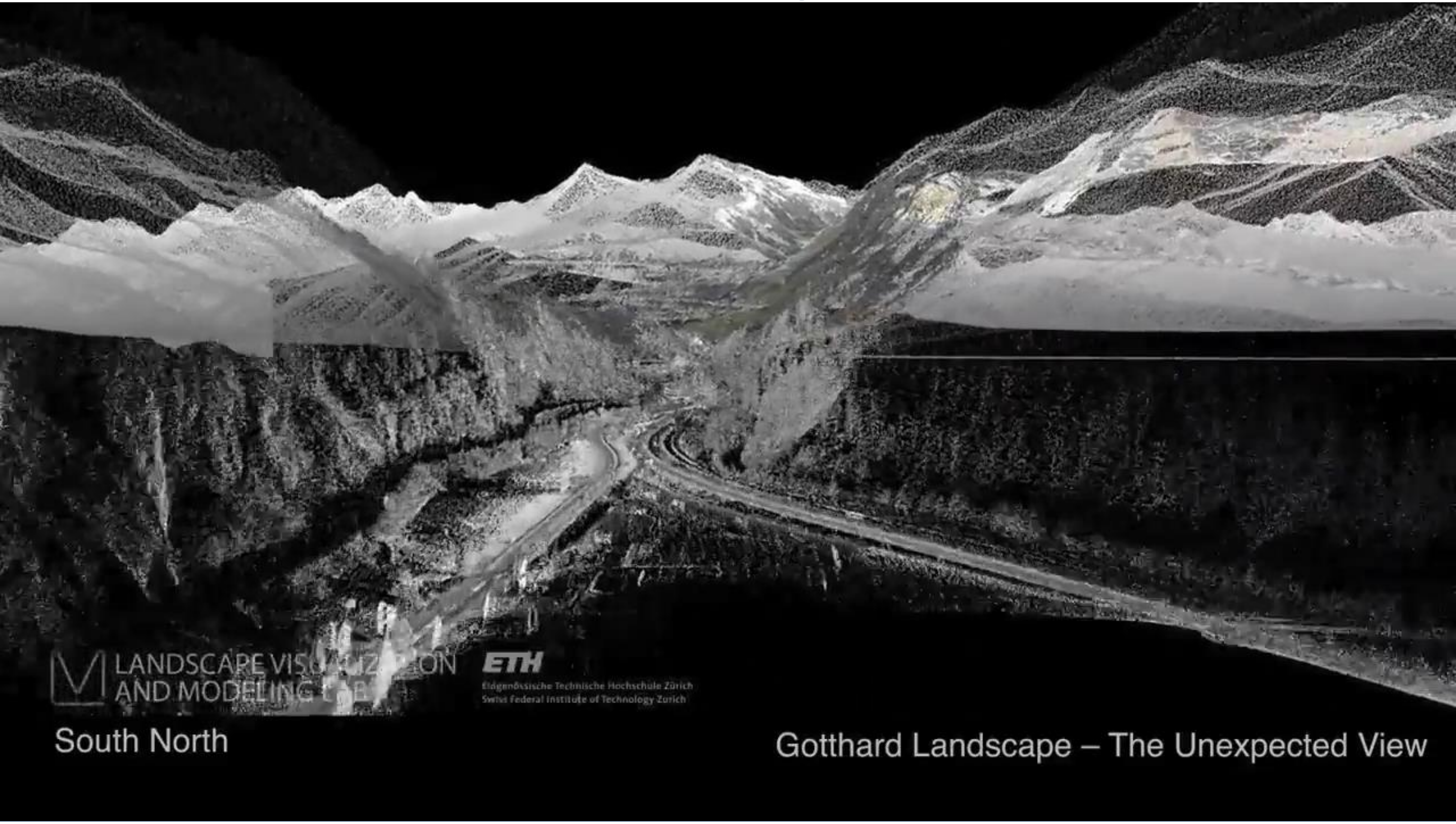


A fruit fly embryo from when it was about two-and-a-half hours old until it walked away from the microscope as a larva, filmed by a new microscope (MuVi-SPIM) developed at Luxendo. Credit to: Mette Handberg from Pavel Tomancak's lab, MPI-CBG, Dresden, Germany

Particle Data – Sources

https://girot.arch.ethz.ch/research/point-cloud-research-in-landscape-architecture

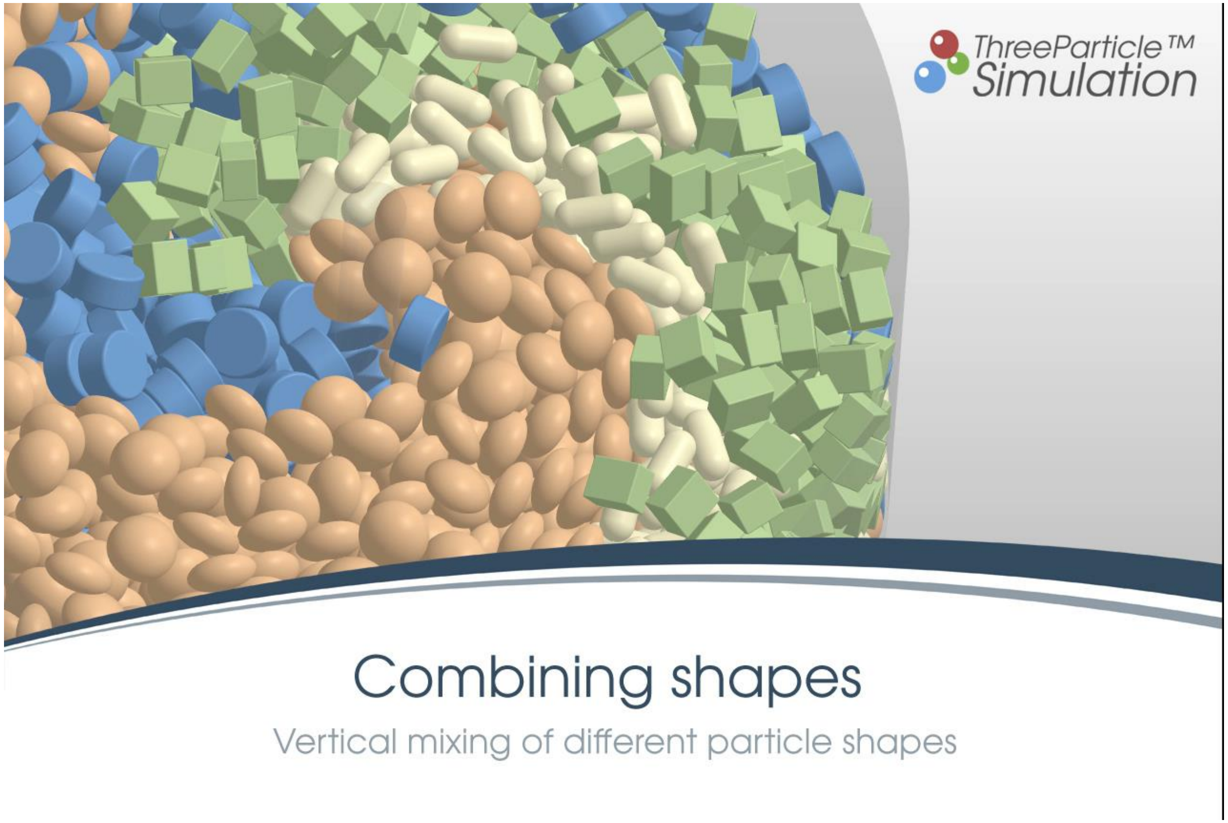
Point Clouds from 3D Scanning



Particle Data – Sources

Discrete Element Method

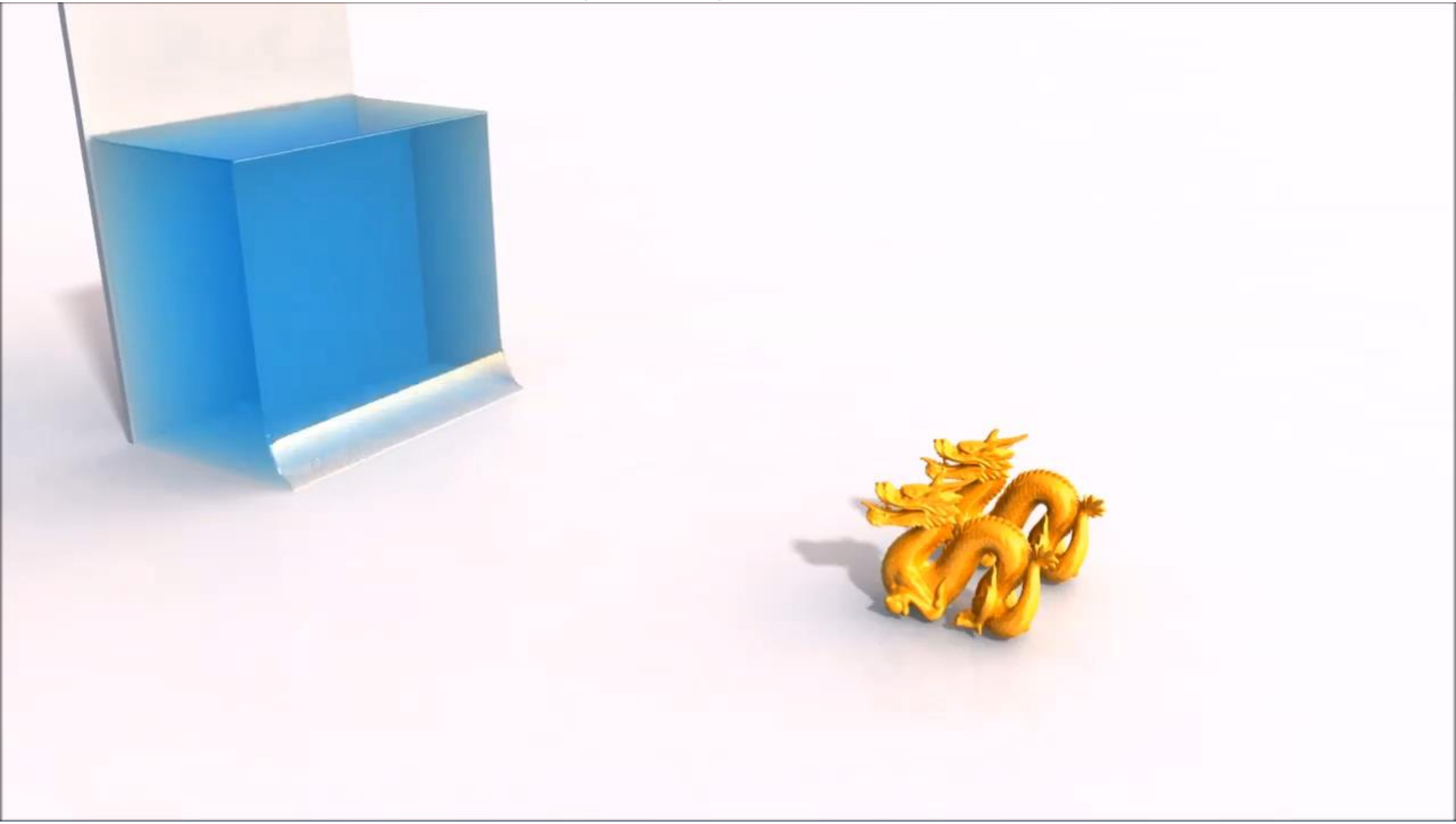
- granular media can be simulated with differently shaped particles



https://www.becker3d.com/showcase

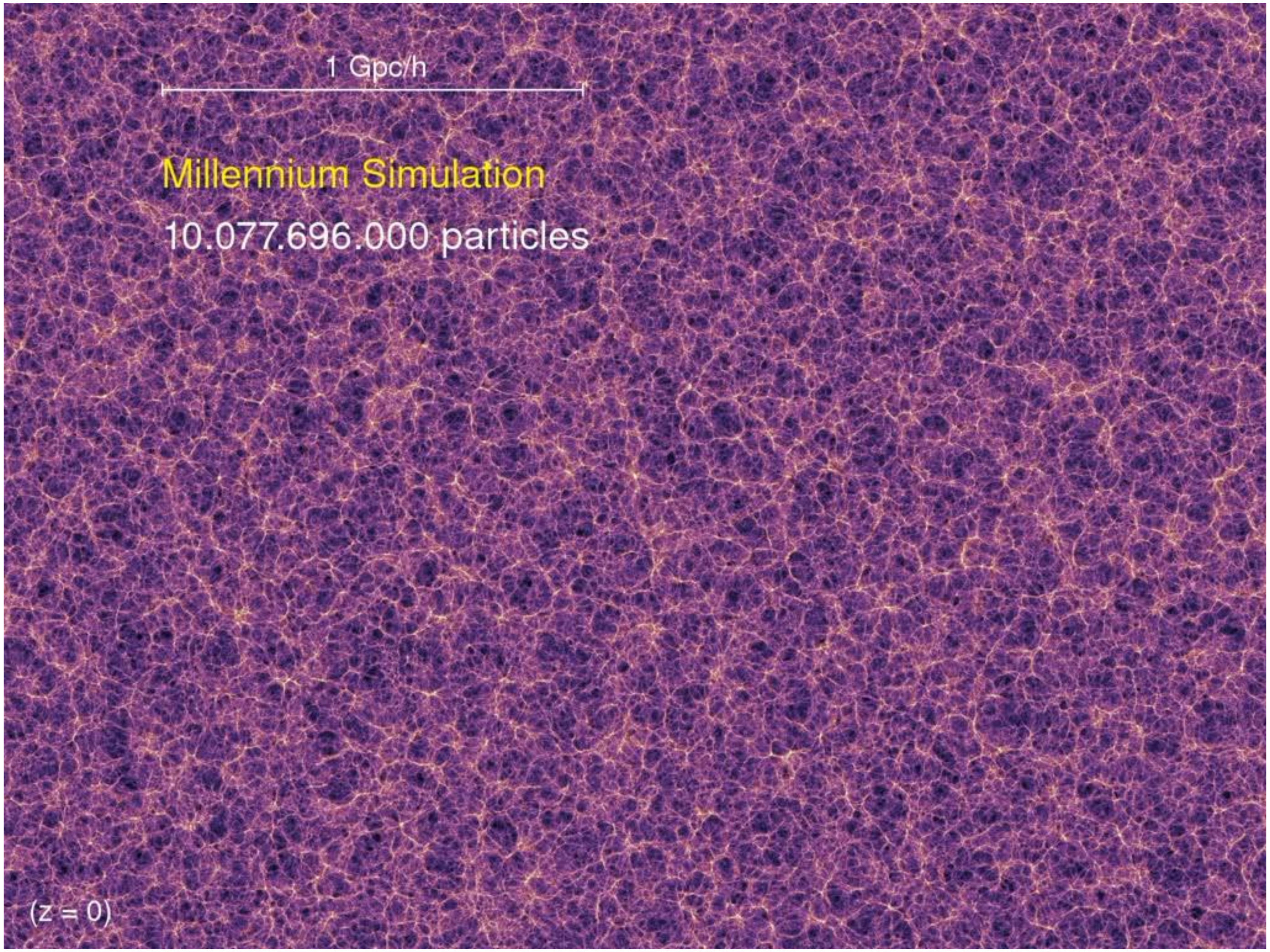
Particle Data – Sources

- Smoothed Particle Hydrodynamics



Particle Data – Sources

- N-Body Simulation (The Millennium Simulation)



Particle Data – Sources

https://www.youtube.com/watch?v=5JcFgj2gHx8

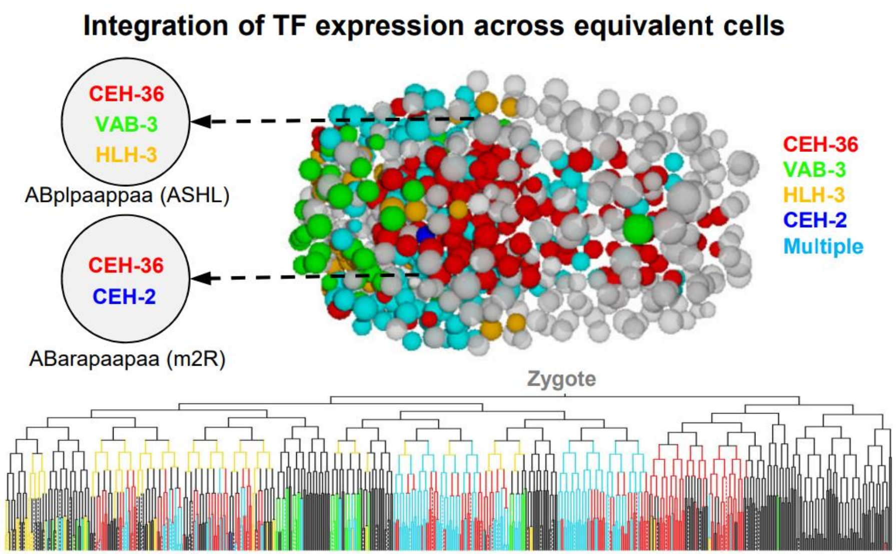
Molecular Dynamics Simulation

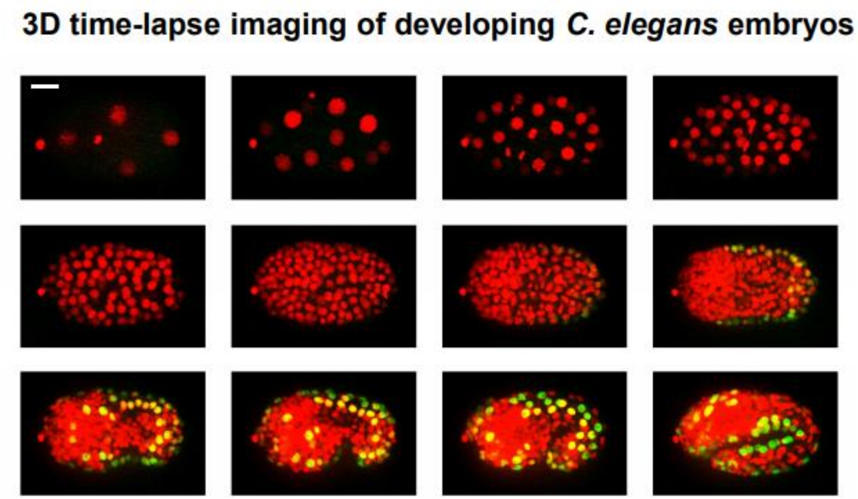
Particle Data – Basics

- Particle data P is a set of N unorganized points indexed with i and carrying further m attributes $a_{k=1\dots m}$:
$$P=\left\{P_i=\left(\underline{\boldsymbol{p}}_i,\vec{\boldsymbol{a}}_i\right)\middle|\underline{\boldsymbol{p}}_i\in\boldsymbol{R}^3,\vec{\boldsymbol{a}}_i\in\boldsymbol{R}^m\right\}$$
- Time dependent data is organized in n frames that are indexed with j and have at least the frame time $t_{j=1\dots n}$ as attribute.
- Particles have a life span $J_i=[j_0,j_1]$ of frame indices in which the particle exists.
- Integer typed particle IDs (typically equal to index i) are used to define inter-frame correspondences of the instances $P_{ij}=(\underline{\boldsymbol{p}}_{ij},\vec{\boldsymbol{a}}_{ij})$ of each particle.
- Optional cluster information is specified by per particle instance cluster IDs. Clusters also have life span and can furthermore split and merge.

Particle Data – Lineage Tree

- cells evolving over time can split into descendants
- ancestor-descendant relationship forms a lineage tree
- this can be stored by one parent index per cell instance
- splits / bifurcations happen when several cell instances refer to the same ancestor cell



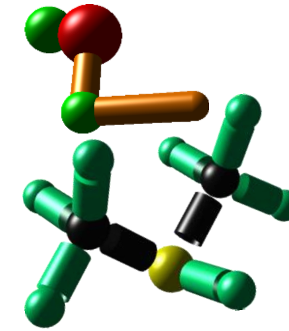


Lineage tree of C. elegans embryo from Ma, Xuehua, et al. "Single-cell protein atlas of transcription factors reveals the combinatorial code for spatiotemporal patterning the C. elegans embryo." *BioRxiv* (2020).

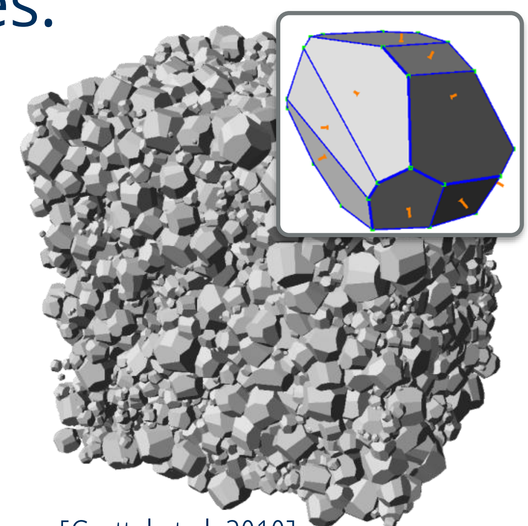
Particle Data – Shape

- Shape can be
  - the same for all particles,
  - vary between particles or even
  - vary per particle over time
- There is a variety of different shapes:
  - 0D (no shape): point,
  - 2D (surfel): (elliptic) disk,
  - standard shapes: sphere, ellipsoids, cylinder, box, cone, rod, torus, n-sided prisms
  - general shapes: like crystals, stones, or linked particles,
  - time varying shapes: bubbles,
  - fuzzy shapes: spherical smoothing kernels

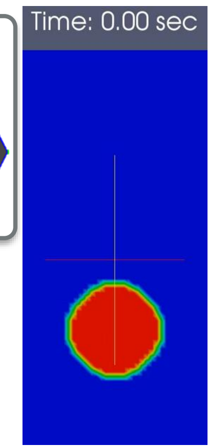
Surfels



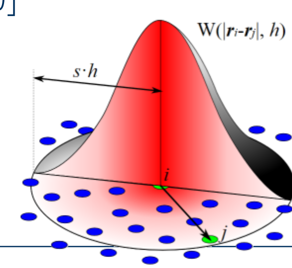
Rigidly linked particles



[Grottel et al. 2010]



air bubble in water

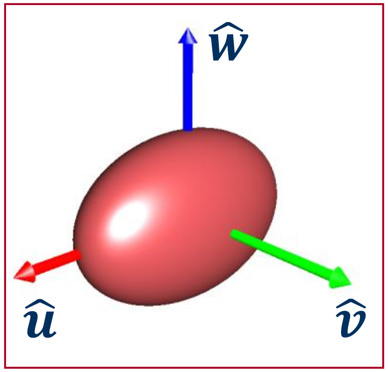


SPH Kernel ©Wikipedia

Particle Data – Orientation

- except for sphere and SPH kernels the orientation of a particle is an attribute
- It can be represented as a 3x3-matrix with the unit vectors of a local coordinate system in its rows

$$\boldsymbol{O} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix}$$



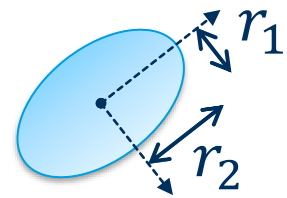
- Alternatively a quaternion is often used to represent O with a 4D vector

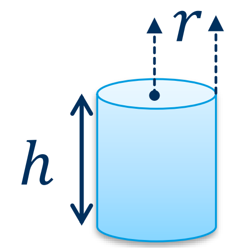
$$\boldsymbol{q} = \begin{pmatrix} s \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} s \\ \vec{\boldsymbol{v}} \end{pmatrix} \Rightarrow \boldsymbol{O} = \begin{pmatrix} s^2 + x^2 - y^2 - z^2 & 2(xy - sz) & 2(xz + sy) \\ 2(xy + sz) & s^2 - x^2 + y^2 - z^2 & 2(yz - sx) \\ 2(xz - sy) & 2(yz + sx) & s^2 - x^2 - y^2 + z^2 \end{pmatrix}$$

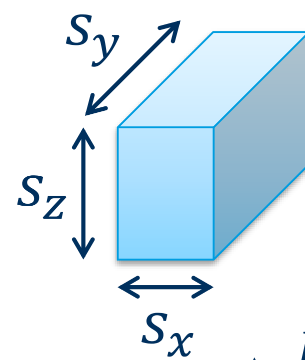
- For a rotation of angle α around axis $\hat{\boldsymbol{n}}$, the entries of the corresponding quaternion are $s = \cos\frac{\alpha}{2}$; $\vec{\boldsymbol{v}} = \sin\frac{\alpha}{2}\hat{\boldsymbol{n}}$

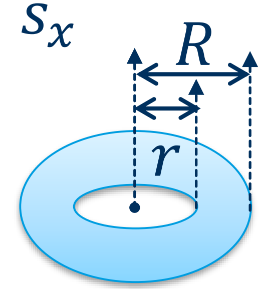
Particle Data – Size

- size is specified in particle aligned coordinate system (e.g. height in local z-direction)
- surfel: one (circular) or two radii (elliptical)
- sphere, SPH kernel: radius r
- cylinder/cone/rod/n-sided prism: radius r + height h
- ellipsoid/box: 3d size vector $\vec{s}$
- torus: minor radius r and major radius R





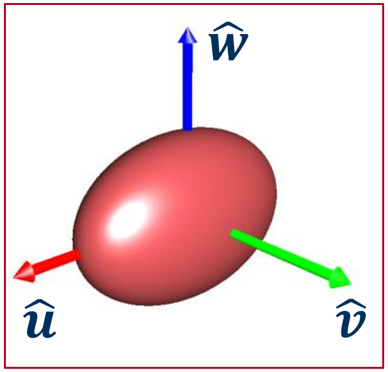




Particle Data – Orientation and Size

Computer Graphics and Visualization

- Ellipsoids are the standard visual representation of symmetric positive semi-definite tensors: $\boldsymbol{T} = \begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}, \det \boldsymbol{T} \geq 0$



- Symmetric tensors have a real valued eigenvalue decomposition:

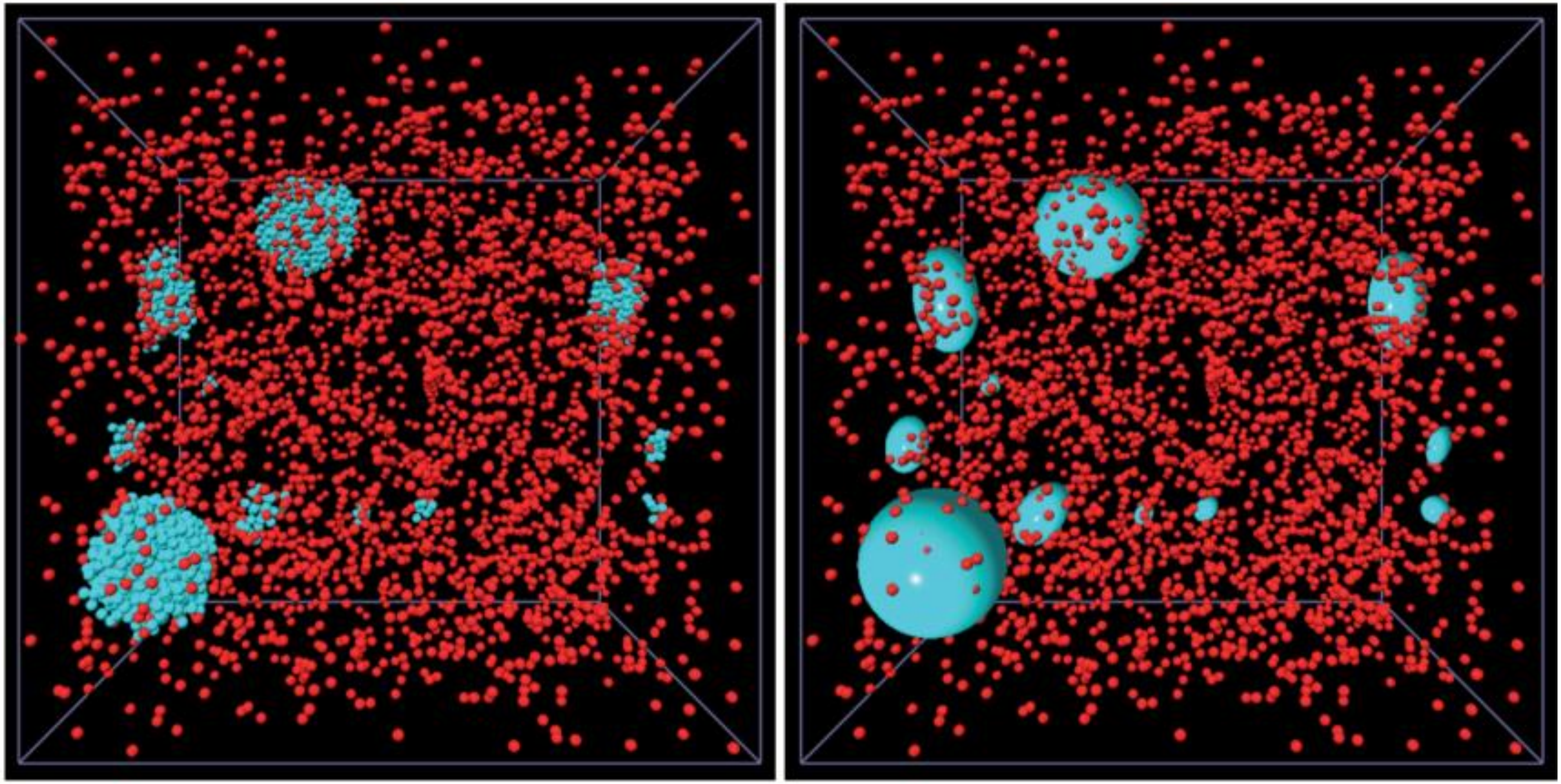
$$\boldsymbol{T} = \mathbf{O}\boldsymbol{\Lambda}\boldsymbol{O}^T = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} \lambda_u & 0 & 0 \\ 0 & \lambda_v & 0 \\ 0 & 0 & \lambda_w \end{pmatrix} \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}$$

- For positive semi-definite tensors all $\lambda_i \geq 0$.
- T implicitly represents orientation $\boldsymbol{O}$ *and* size with

$$\vec{\boldsymbol{s}}^T = (\lambda_u \quad \lambda_v \quad \lambda_w)$$

Particle Cluster to Ellipsoid

- In some applications clusters are ellipsoidal



Particle Cluster to Ellipsoid

Computer Graphics and Visualization

- Given the particles $\underline{\boldsymbol{p}}_i$ with radii R_i of a cluster, the tensor representation $\boldsymbol{T}$ can be computed from the covariance matrix as follows:
- compute particle weights from particle sizes: $\omega_i = \frac{R_i^3}{\sum_{i'} R_{i'}^3}$
- compute cluster center: $\underline{\boldsymbol{c}} = \sum_i \omega_i \, \underline{\boldsymbol{p}}_i$
- Compute cluster covariance matrix and eigenvalue decomposition:

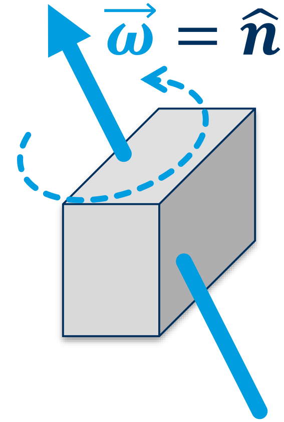
$$\boldsymbol{C} = \sum_i \omega_i \left(\underline{\boldsymbol{p}}_i - \underline{\boldsymbol{c}}\right)\left(\underline{\boldsymbol{p}}_i - \underline{\boldsymbol{c}}\right)^T = \boldsymbol{O}\boldsymbol{\Gamma}\boldsymbol{O}^T$$

- Form tensor with the square root of the diagonal entries $\boldsymbol{\Gamma}$ of the covariance decomposition: $\boldsymbol{T} = \boldsymbol{O}\sqrt{\boldsymbol{\Gamma}}\boldsymbol{O}^T$

Particle Data – Time Derivative

Computer Graphics and Visualization

Velocity



$$\vec{\boldsymbol{\omega}} = \hat{\boldsymbol{n}} \cdot \frac{\Delta\phi}{\Delta t}$$

- linear velocity $\vec{\boldsymbol{v}}_i$ of the particles is time derivative of position
- time derivative of the orientation is typically stored as angular velocity vector $\vec{\boldsymbol{\omega}}_i$ with axis of rotation as direction and rotation angular speed in radiant per second as length
- Further velocities from time derivatives of other attributes like size

Acceleration

- Time derivative of velocity
- often interesting feature as it measures the speed of change with respect to a uniform motion

Particle Data – Further Attributes

- physics provides many more attributes:

Scalar

- mass/density, charge, capacity, load, temperature,

Vector

- linear and angular momentum, spin,
  field vectors (electric, magnetic, forces, ...)
- color, surface normal

Tensor

- positional uncertainty, directional diffusion, stress, strain, deformation, ...

Particles

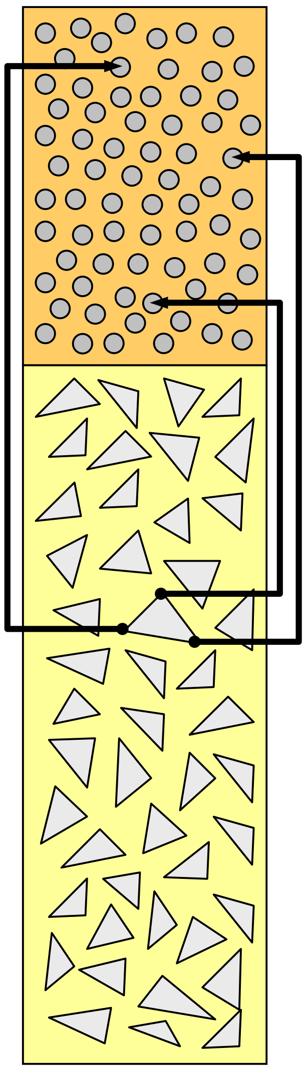
SHADER BASED RAYCASTING

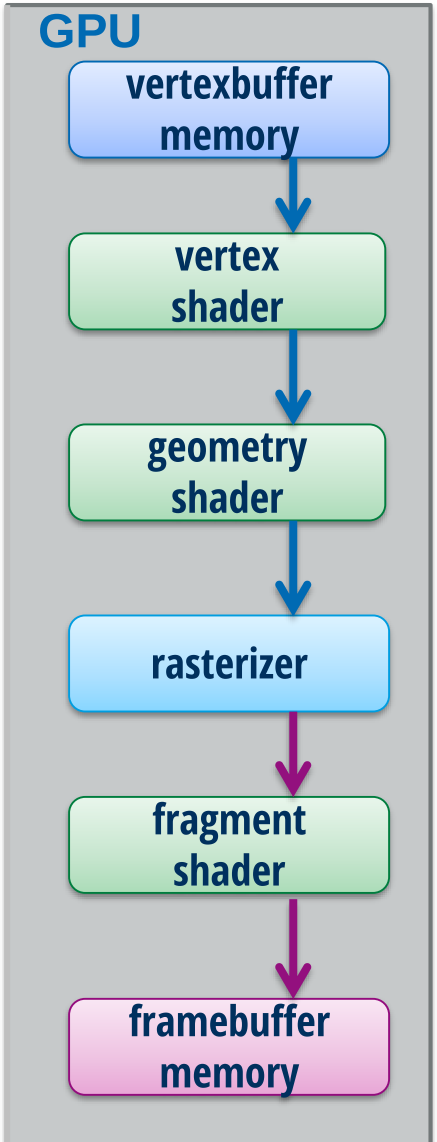
SPHERES

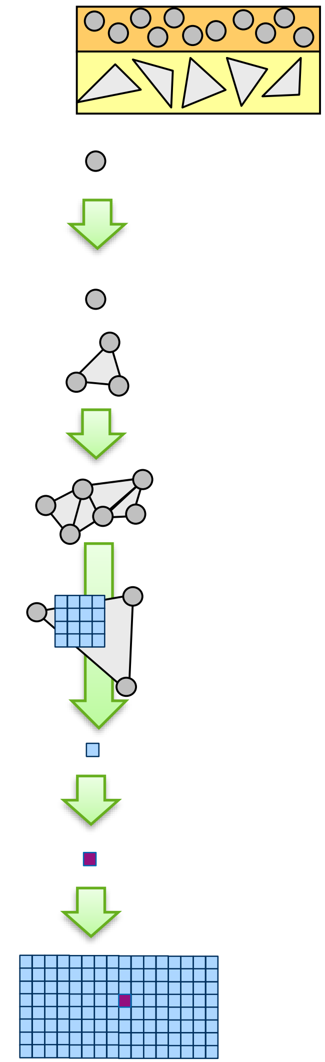
Shader Pipeline – Standard Usage

Computer Graphics and Visualization

Indexed mesh rendering





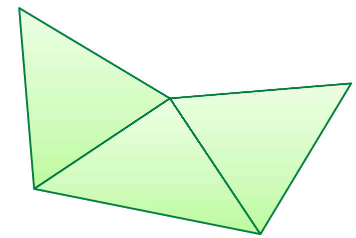


Shader Pipeline – Limitations

- points, lines and triangles are only supported graphics primitives

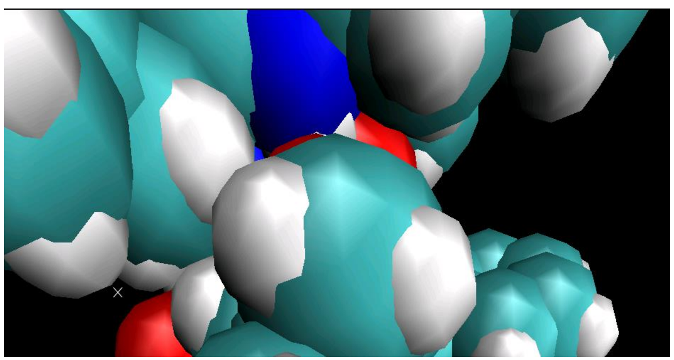
points

lines

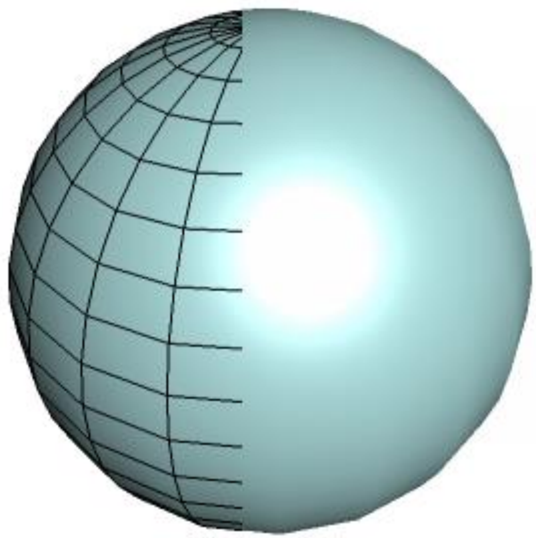


triangles

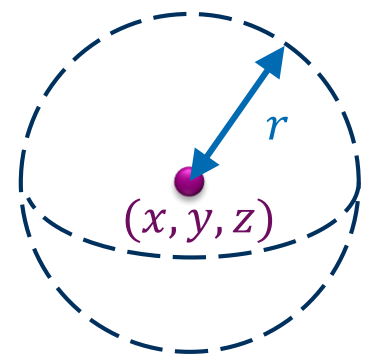
- all other primitives must be tesselated



too coarse tesselation gives bad results for intersecting spheres



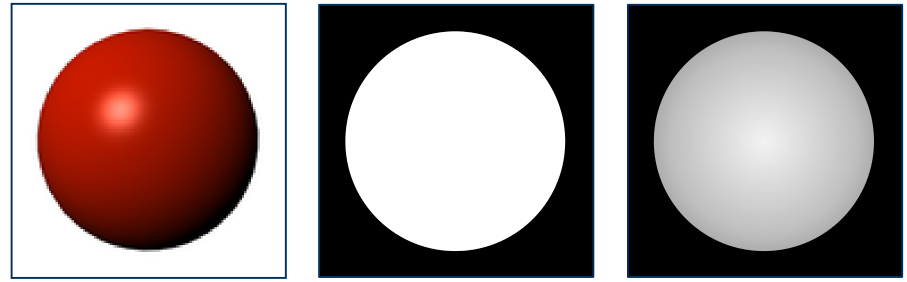
20x20 = 400 vertices plus 800 triangles



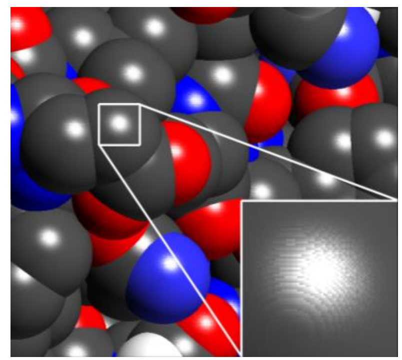
vs 4 floats

Billboard Spheres

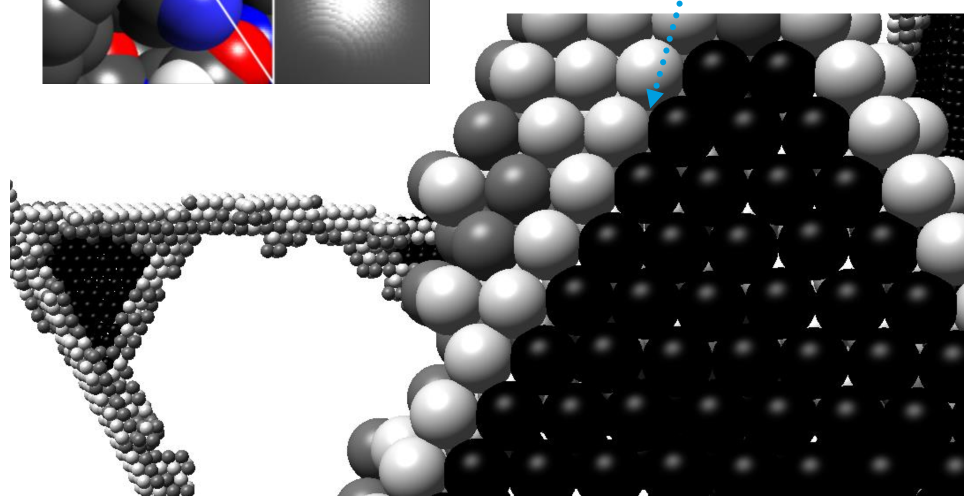
- Billboard: use rgbα image of lit sphere with α-channel used as mask.
- optionally add depth texture to support correct intersection between spheres
- Problems:
  - 8-bit depths not sufficient
  - not correct for perspective projection
  - only directional lighting is possible
  - texture fetch expensive



rgb α d

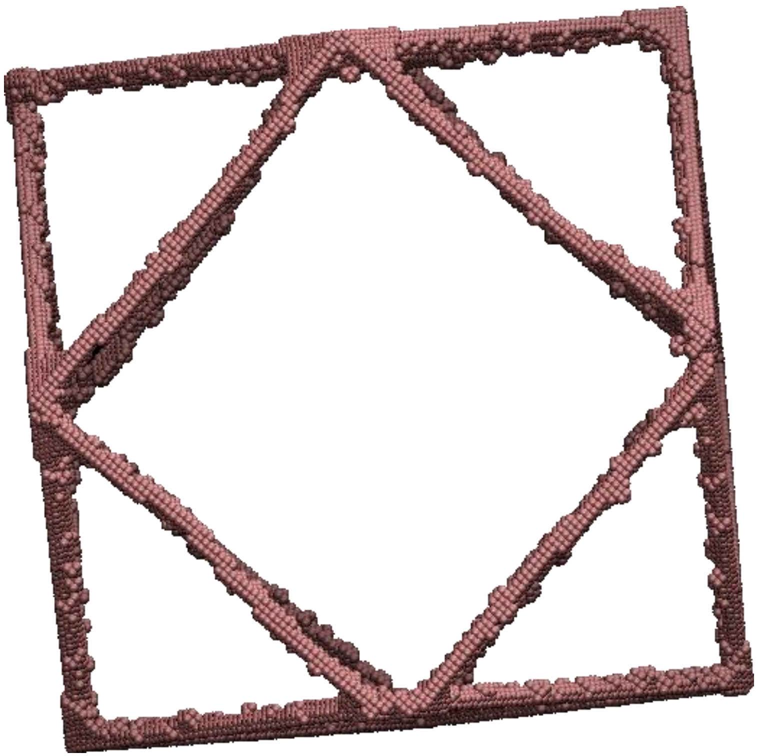


spheres wrongly intersect in perspective projection

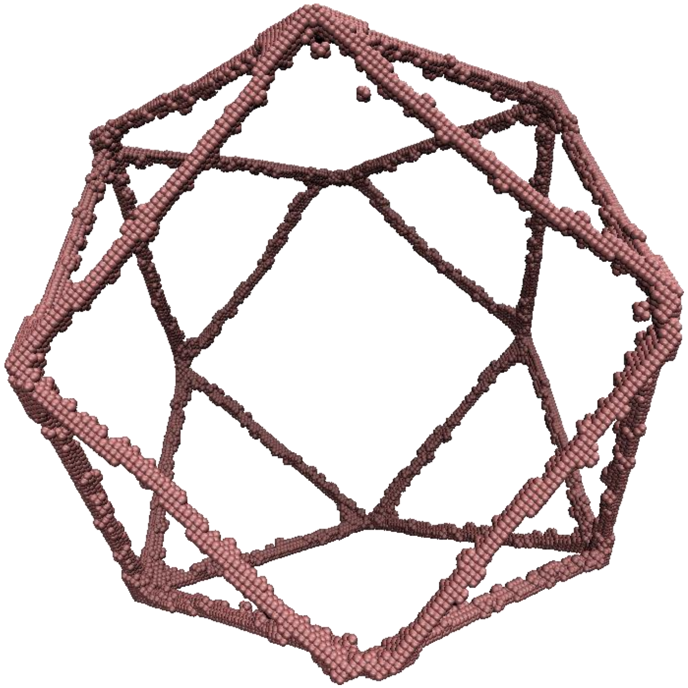


Billboard Spheres

perspective projection provides better spatial perception



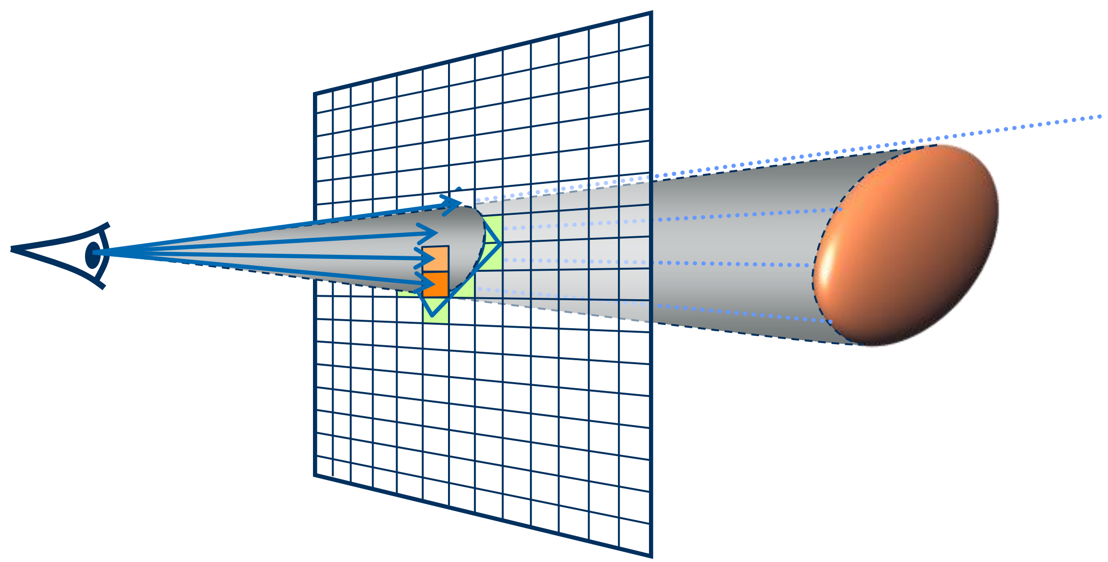
Orthographic Projection



Perspective Projection

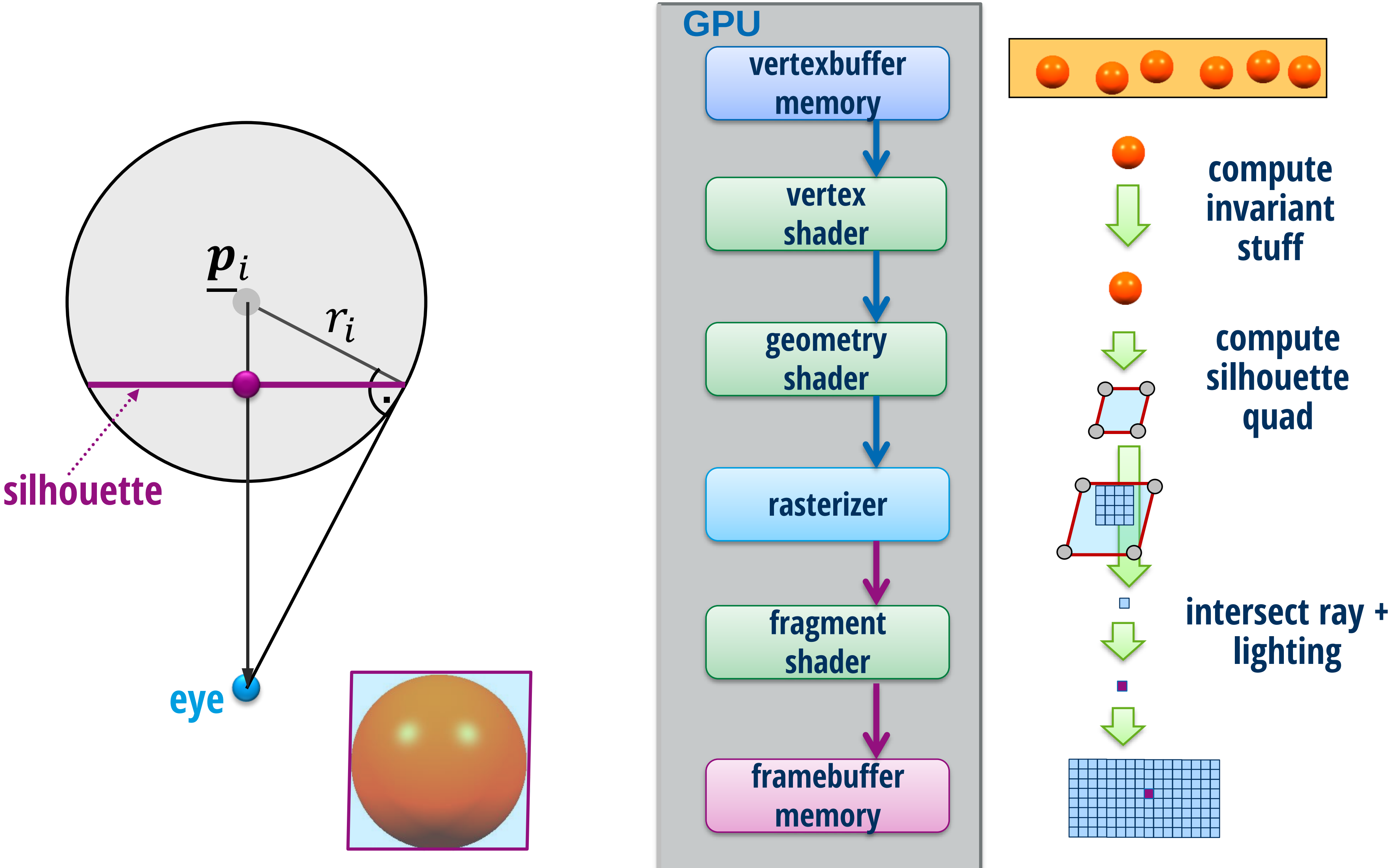
Particle Raycasting – Motivation

- Compute correct projected silhouette
- cover silhouette with graphics primitive[s]



- per fragment intersect ray with particle
- compute illumination

Shader Pipeline – Sphere Raycasting



Sphere Raycasting – Silhouette

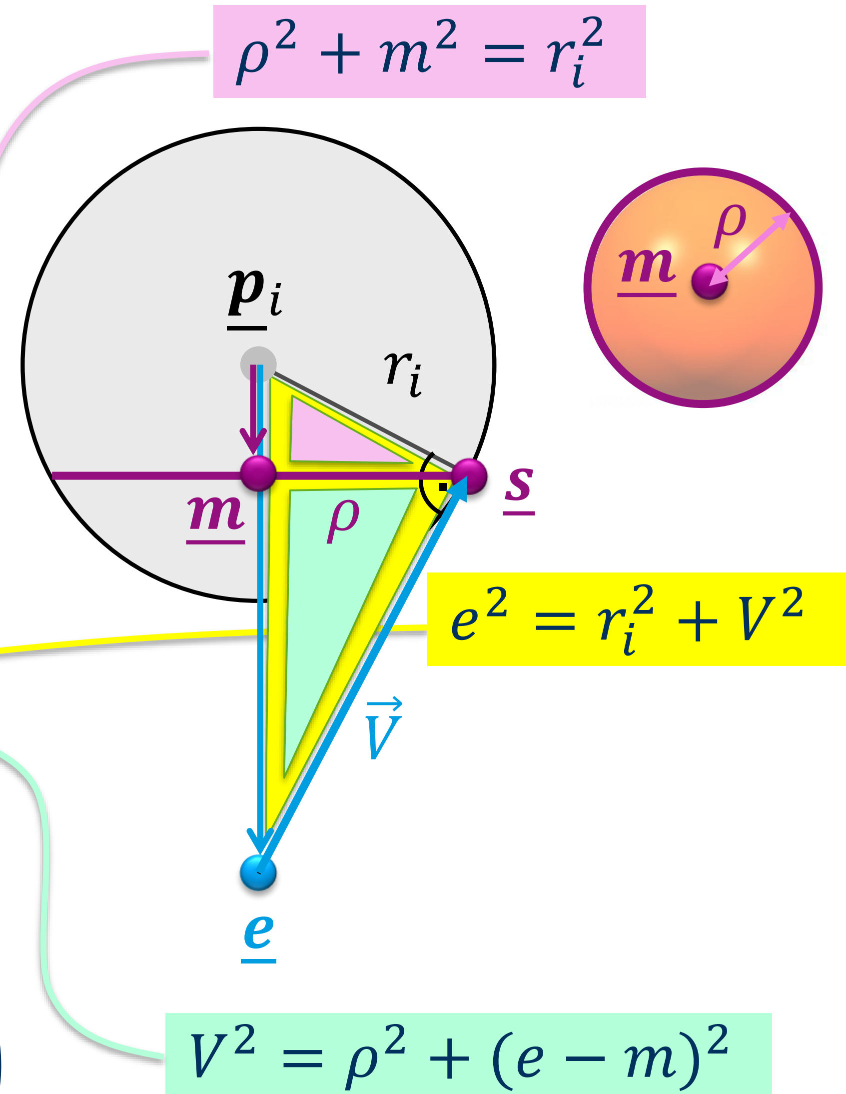
- Consider particle coordinates with sphere center $\underline{\boldsymbol{p}}_i$ in the origin
- silhouette points $\underline{\boldsymbol{s}}$ form a circle of radius $\rho < r_i$ around center $\underline{\boldsymbol{m}}$.
- First compute length m of silhouette center $\underline{\boldsymbol{m}}$:

$$2em + V^2 = \rho^2 + e^2 + m^2$$

$$2em = \rho^2 + r_i^2 + m^2 = 2r_i^2$$

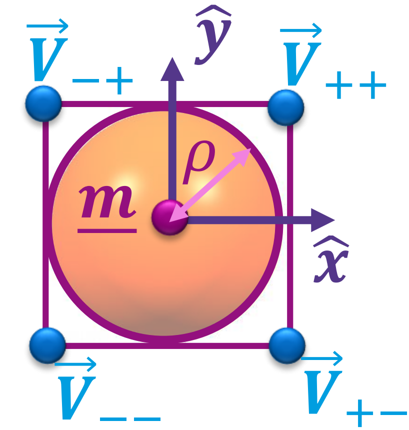
- Thus we have $m = \frac{r_i^2}{e}$ and

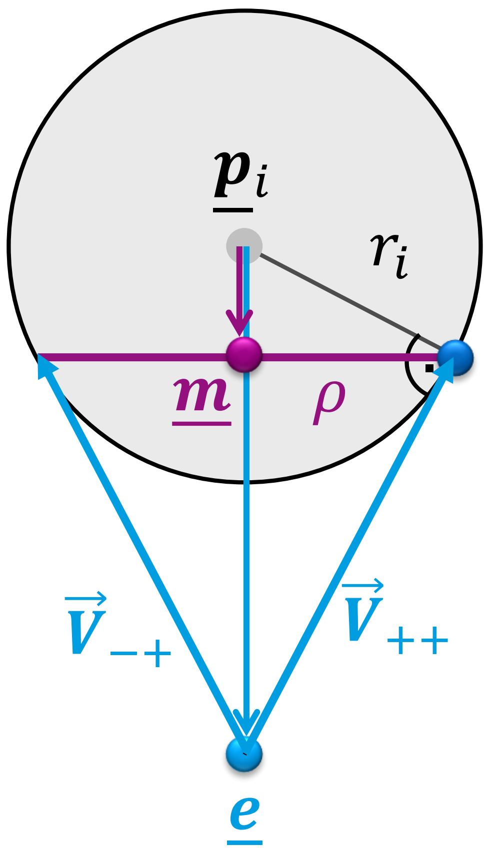
$$\underline{\boldsymbol{m}} = \frac{r_i^2}{e^2}\left(\underline{\boldsymbol{e}} - \underline{\boldsymbol{p}}_i\right), \rho^2 = r_i^2\left(1 - \frac{r_i^2}{e^2}\right)$$



Sphere Raycasting – Silhouette

- To cover the sphere with a quad of corners $\vec{V}_{\pm\pm}$, two orthogonal directions $\hat{x}$ and $\hat{y}$ are computed
- The 4 quad corners are
  $$\vec{V}_{\pm\pm} = \underline{m} \pm \rho\hat{x} \pm \rho\hat{y}$$
- Attaching texture coordinates $\underline{q} \in [-1,1]^2$ to the quad corners, the ray-sphere intersection test simplifies to:
  $$\left\|\underline{q}\right\|^2 \leq 1$$





Sphere Raycasting – Silhouette

- rasterizer interpolates $\vec{V}_{\pm\pm}$ over quad to per fragment ray vector $\vec{V}$
- equations for ray-sphere intersection:

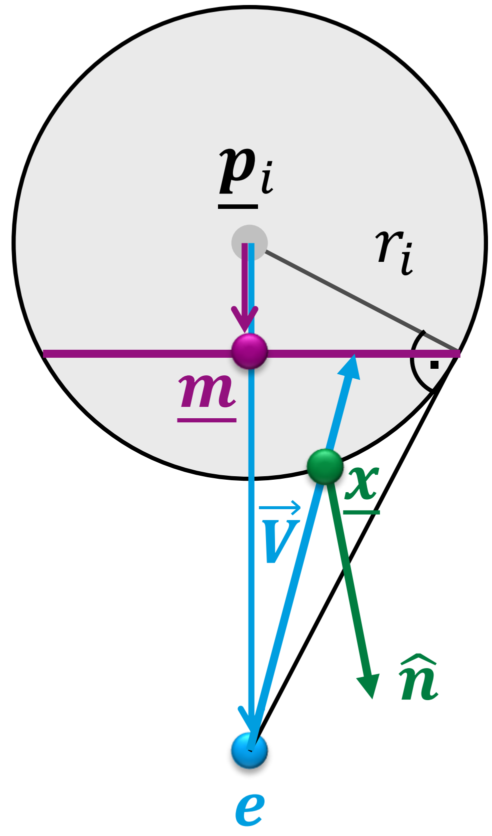
$$\underline{x} = \underline{e} + \lambda\vec{V}, \left\|\underline{x} - \underline{p}_i\right\|^2 = r_i^2$$

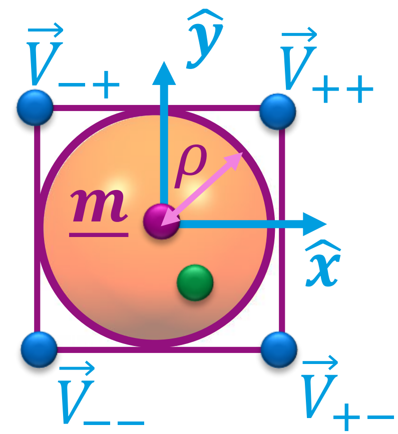
➔two solutions (homework)

- special form where λ_+ is first intersection along ray:

$$\underline{x}_{\pm} = \underline{e} + \lambda_{\pm}\vec{V}, \lambda_{\pm} = \frac{1}{1 \pm \beta}$$

$$\beta = r_i\sqrt{1 - \left\|\underline{q}\right\|^2} \Big/ \left\|\underline{e} - \underline{p}_i\right\|$$





$\underline{q} \in [-1,1]^2$

- The surface normal follows immediately:

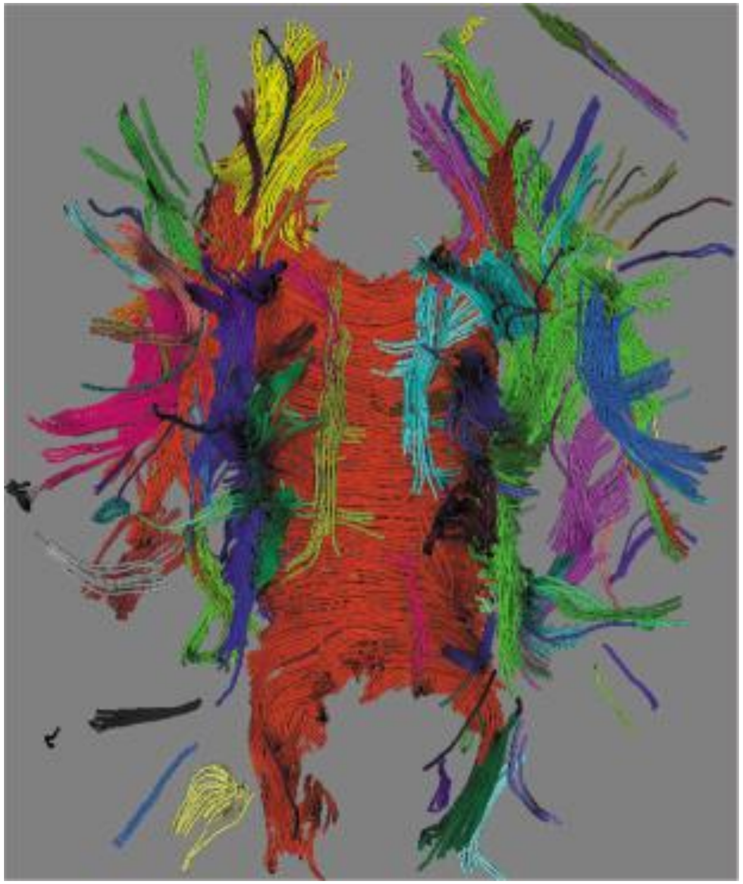
$$\hat{n} = \frac{1}{\left\|\underline{x} - \underline{p}_i\right\|}\left(\underline{x} - \underline{p}_i\right)$$

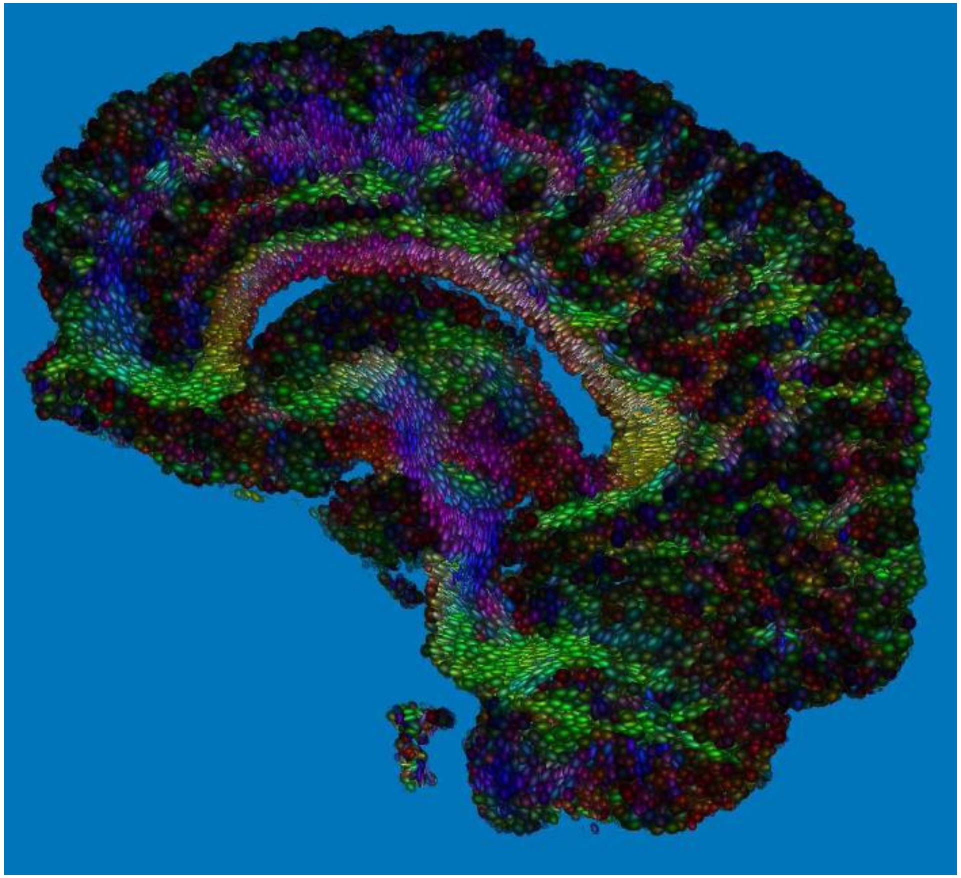
Particles

SHADER BASED RAYCASTING

ELLIPSOIDS

Examples of Diffusion Tensors





©Moberts, Vilanova, van Wijk

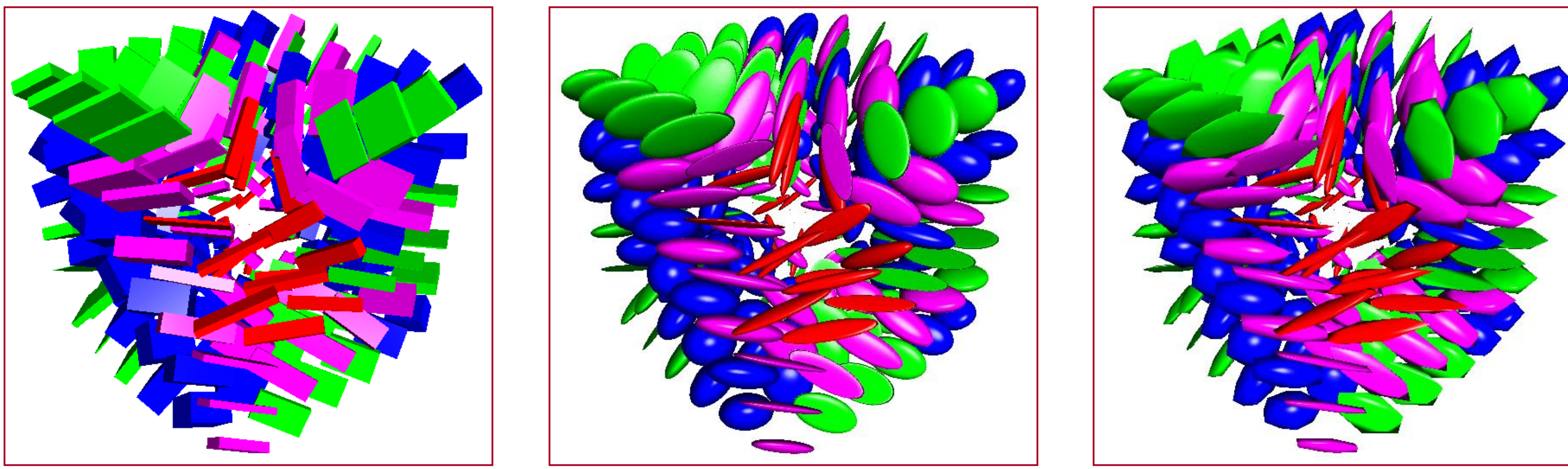
©Kondratieva, Krüger, Westermann

Ellipsoid Raycasting

- A symmetric tensor $\boldsymbol{T} = \mathbf{O}\boldsymbol{\Lambda}\boldsymbol{O}^T$ can be used to transform points $\underline{\tilde{\boldsymbol{x}}}$ on an arbitrary primitive according to

$$\underline{\boldsymbol{x}} = \boldsymbol{T}\underline{\tilde{\boldsymbol{x}}} = \mathbf{O}\boldsymbol{\Lambda}\boldsymbol{O}^T\underline{\tilde{\boldsymbol{x}}}$$

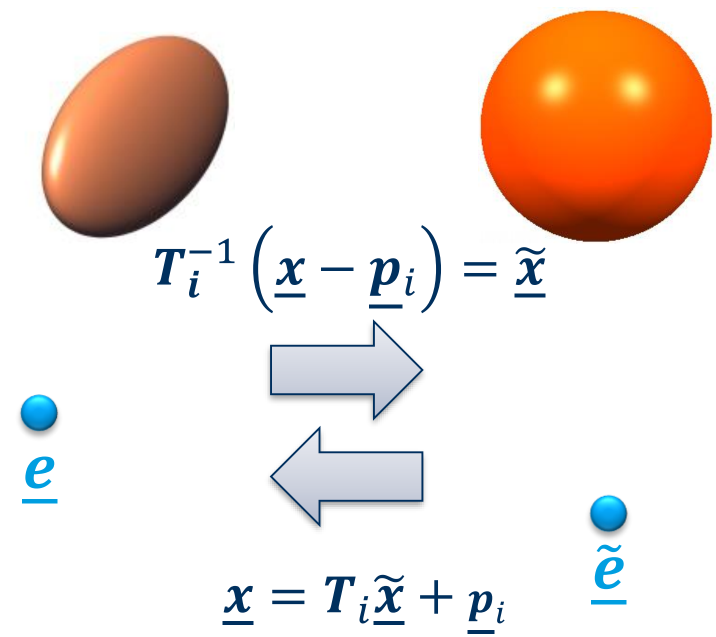
rotate back **scale** **rotate**



Examples of application of tensor to box, sphere and icosahedron primitives

Ellipsoid Raycasting

- For ellipsoid raycasting we apply inverse tensor transformation $\boldsymbol{T}_i^{-1}$ to all points and vectors that are then denoted with a tilde on top
- As inversely transformed ellipsoid is a sphere of radius 1, we drop r_i & get
  $$\underline{\tilde{\boldsymbol{m}}} = \frac{1}{\tilde{e}^2}\underline{\tilde{\boldsymbol{e}}}, \tilde{\rho}^2 = 1 - \frac{1}{\tilde{e}^2}$$
- With orthonormal basis we get quad corners
  $$\widetilde{\vec{\boldsymbol{V}}}_{\pm\pm} = \underline{\tilde{\boldsymbol{m}}} \pm \tilde{\rho}\widetilde{\hat{\boldsymbol{x}}} \pm \tilde{\rho}\widetilde{\hat{\boldsymbol{y}}}$$



- Quad is transformed back to world or eye coordinates before rendering

Ellipsoid Raycasting – Intersection

- equations for ray-ellipsoid intersection:

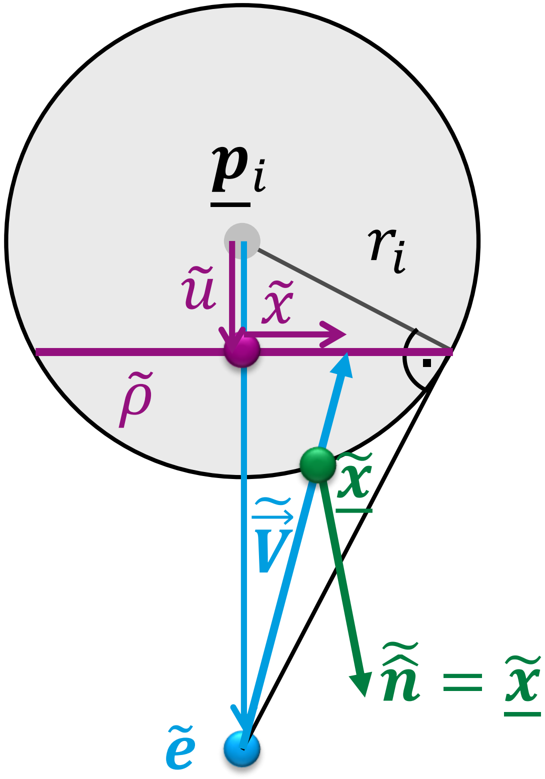
$$\underline{\tilde{\boldsymbol{x}}} = \underline{\tilde{\boldsymbol{e}}} + \lambda \vec{\tilde{\boldsymbol{V}}}, \vec{\tilde{\boldsymbol{V}}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{u} \end{pmatrix}, \tilde{u} = \frac{1}{\tilde{e}}$$

- texture coordinates:

$$\underline{\boldsymbol{q}} = \begin{pmatrix} \pm 1 \\ \pm 1 \end{pmatrix} = \frac{1}{\tilde{\rho}} \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}$$

- Skipping some steps, the solution of the quadratic equation for 1st inters. is:

$$\lambda_+ = \frac{1}{1 + \tilde{u}\sqrt{1 - \underline{\boldsymbol{q}}^2}}$$



- normal is transformed with inverse transposed:

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

Ellipsoid Raycasting – Incremental

- transforming back to world coordinates yields:

$$\underline{\boldsymbol{x}} = \underline{\boldsymbol{e}} + \lambda_+\left(\tilde{u}, \underline{\boldsymbol{q}}\right)\vec{\boldsymbol{V}}$$

$$\vec{\boldsymbol{n}} = \boldsymbol{T}^{-2}\left(\underline{\boldsymbol{e}} - \underline{\boldsymbol{p}}_i\right) + \lambda_+\left(\tilde{u}, \underline{\boldsymbol{q}}\right)\boldsymbol{T}^{-2}\vec{\boldsymbol{V}}$$

const per frame — CPU program

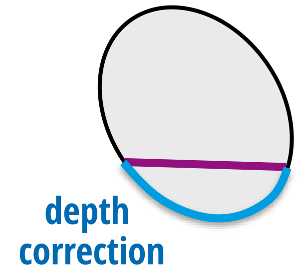
const per ellipsoid — vertex shader

per silhouette vertex — geometry shader

- check if $\underline{\boldsymbol{q}}^2 \leq 1$ and `discard` fragment otherwise; compute $\lambda_+ = \frac{1}{1+\tilde{u}\sqrt{1-\underline{\boldsymbol{q}}^2}}$, $\underline{\boldsymbol{x}}$ and $\vec{\boldsymbol{n}}$ in eye space for lighting — fragment shader

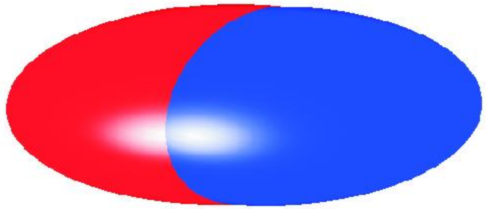
Ellipsoid Raycasting – Depth

- besides color, each fragment contains a depth value used for z-buffer based extraction of the visible surface
- By default, the depth is generated by the rasterizer and corresponds to the depth of the silhouette quad
- To overwrite the default value one can assign in the fragment shader a new depth to `gl_FragDepth`
- This depth value needs to be specified in window coordinates
  - Last stage before rasterizer uses projection matrix **P** to output z and w clip-coordinates
  - Fragment shader inputs clip z and w and performs w-clip to compute z in normalized device coordinates
  - Finally, depth is computed by remapping interval $[-1,1]$ to $[0,1]$



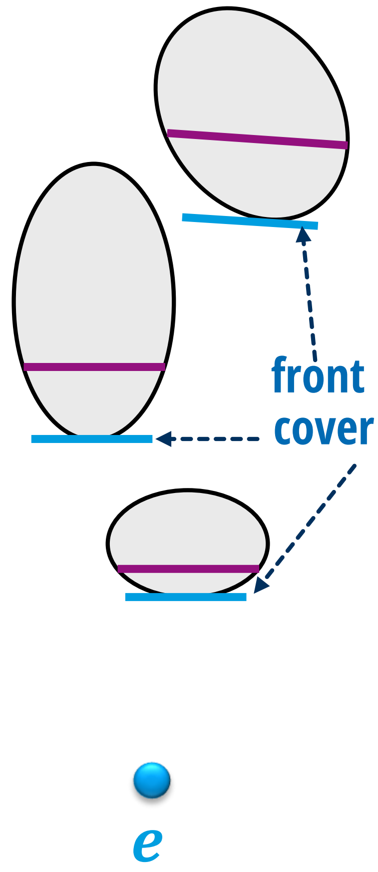

```
// vertex/geometry shader
out vec2 e_zw_clip;
out vec2 V_zw_clip;
:
   e_zw_clip = -(P*vec4(0,0,0,1)).zw;
   V_zw_clip =  (P*V_eye).zw;
:

// fragment shader
in vec2 e_zw_clip;
in vec2 V_zw_clip;
:
   vec2 zw_cl = e_zw_clip+lambda*
                V_zw_clip;
   float z_NDC = zw_cl.z/zw_cl.w;
   gl_FragDepth = 0.5*(z_NDC+1.0);
:
```



Early Depth Test

- If you overwrite the depth value in the fragment shader, by default no depth test is done before.
- Therefore fragment shader is executed also for all hidden fragments wasting compute power
- With the extension `GL_ARB_conservative_depth` one can enable an additional depth test on the fragment depth generated during rasterization
- For this to work correctly, the silhouette quad needs to be placed completely in front of primitive such that all rastierizer depth values are smaller or equal to fragment depths
- In this way most fragments hidden during rendering can be discarded before execution of fragment shader




```
#extension GL_ARB_conservative_depth : require
layout ( depth_greater ) out float gl_FragDepth;
```

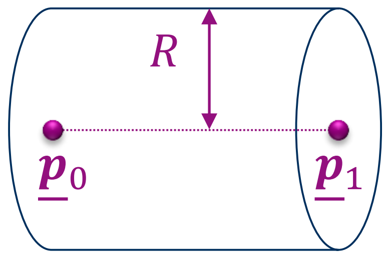
Particles

SHADER BASED RAYCASTING

CYLINDERS

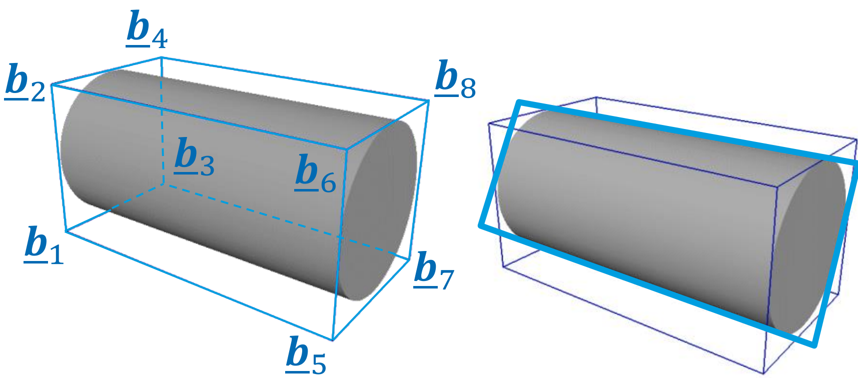
Cylinder Raycasting

Definition: start and end points $\underline{\boldsymbol{p}}_0$ and $\underline{\boldsymbol{p}}_1$ plus radius R.



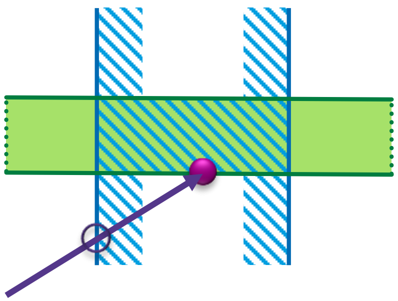
Silhouette Cover

- tessellate object oriented bounding box (OOBB)
- view-dependently align single quad to silhouette



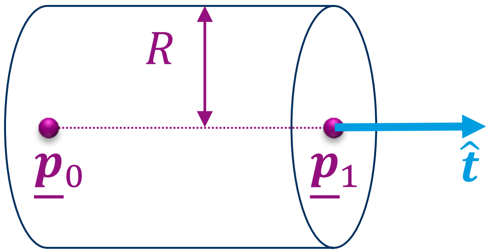
Ray Intersection

- represent cylinder as intersection of two planar half-spaces and cylinder barrel
- Intersection is first ray point that is inside of all three parts



Cylinder Raycasting – OOBB

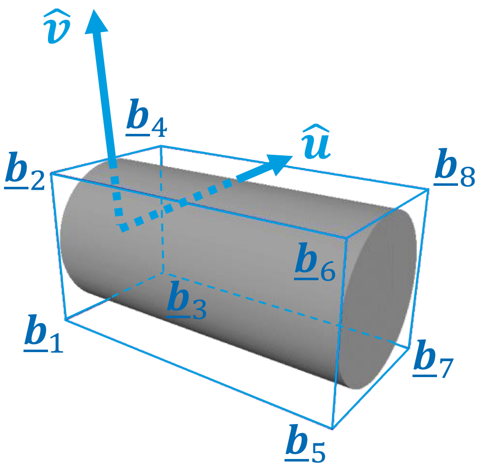
OOBB Tessellation



- compute tangent vector $\hat{\boldsymbol{t}}$ from $\underline{\boldsymbol{p}}_{0|1}$
- extent to orthonormal object coordinate system $\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{t}}$:

$$\hat{\boldsymbol{u}} = \text{normalize}\left(\hat{\boldsymbol{t}} \times \begin{cases} \hat{\boldsymbol{z}} & t_x^2 + t_y^2 > \epsilon \\ \hat{\boldsymbol{y}} & \text{sonst} \end{cases}\right)$$

$$\hat{\boldsymbol{v}} = \hat{\boldsymbol{t}} \times \hat{\boldsymbol{u}}$$



- box corners: $\underline{\boldsymbol{b}}_{1..8} = \underline{\boldsymbol{p}}_{0|1} \pm R\hat{\boldsymbol{u}} \pm R\hat{\boldsymbol{v}}$
- For more efficient transformation encode information in a single 4x4-matrix:

$$\widetilde{\boldsymbol{B}}^{\text{world}} = \begin{pmatrix} \underline{\boldsymbol{p}}_0 & \underline{\boldsymbol{p}}_1 & R\hat{\boldsymbol{u}} & R\hat{\boldsymbol{v}} \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

OOBB Tessellation

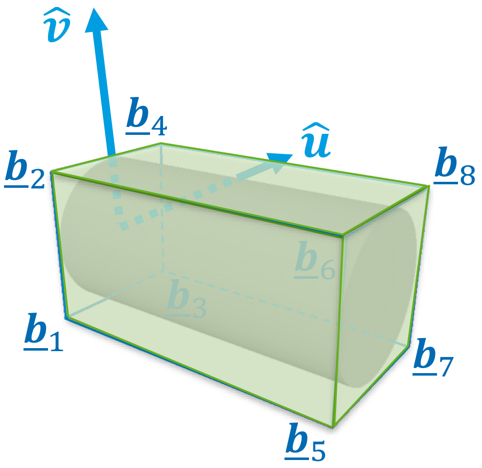
- already in vertex shader transform to clip coordinates

$$\widetilde{\boldsymbol{B}}^{\text{clip}} = \boldsymbol{MVP} \cdot \widetilde{\boldsymbol{B}}^{\text{world}}$$

- Pass $\widetilde{\boldsymbol{B}}^{\text{clip}}$ to geometry shader, recover clip space corners and emit length 2 triangle strip per OOBB face

$$\widetilde{\boldsymbol{b}}_{1..8}^{\text{clip}} = \widetilde{\boldsymbol{B}}_{0|1}^{\text{clip}} \pm \widetilde{\boldsymbol{B}}_{2}^{\text{clip}} \pm \widetilde{\boldsymbol{B}}_{3}^{\text{clip}}$$

- Careful: this pre-transformation only works with points/vectors that have either 1 or 0 in the w-component.
- Optionally perform culling of backfacing facettes

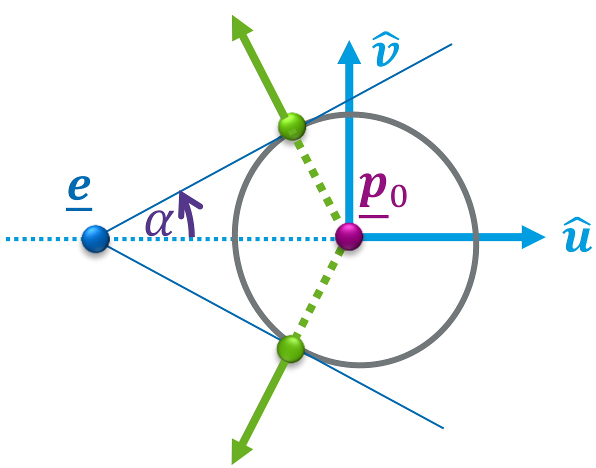


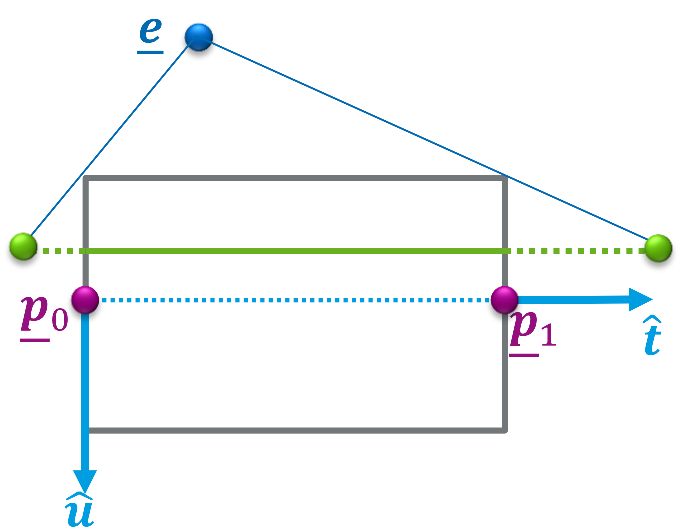
Cylinder Raycasting

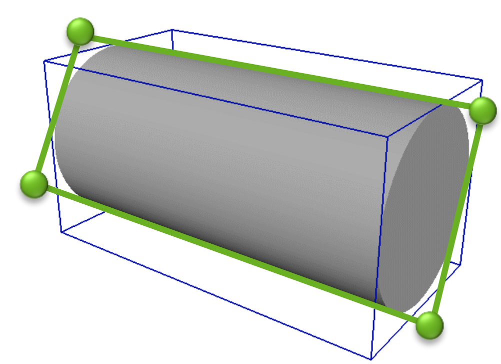
Computer Graphics and Visualization

Silhouette Covering Quad

- View-dependently choose orthonormal coord.system $\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{t}}$ to align $\hat{\boldsymbol{t}}$ with cylinder and $\hat{\boldsymbol{u}}$ to span together with $\hat{\boldsymbol{t}}$ plane through eye $\underline{\boldsymbol{e}}$ and center line $\underline{\boldsymbol{p}}_0\underline{\boldsymbol{p}}_1$.
- Silhouette is bound by 2 lines parallel to $\hat{\boldsymbol{t}}$ with uv-coords from eye distance e to center line: $u = -\frac{R^2}{e}$ [see m on slide 25], $v_{1|2} = \pm\sqrt{R^2 - u^2}$
- Extent rectangle in ut-plane according to middle figure



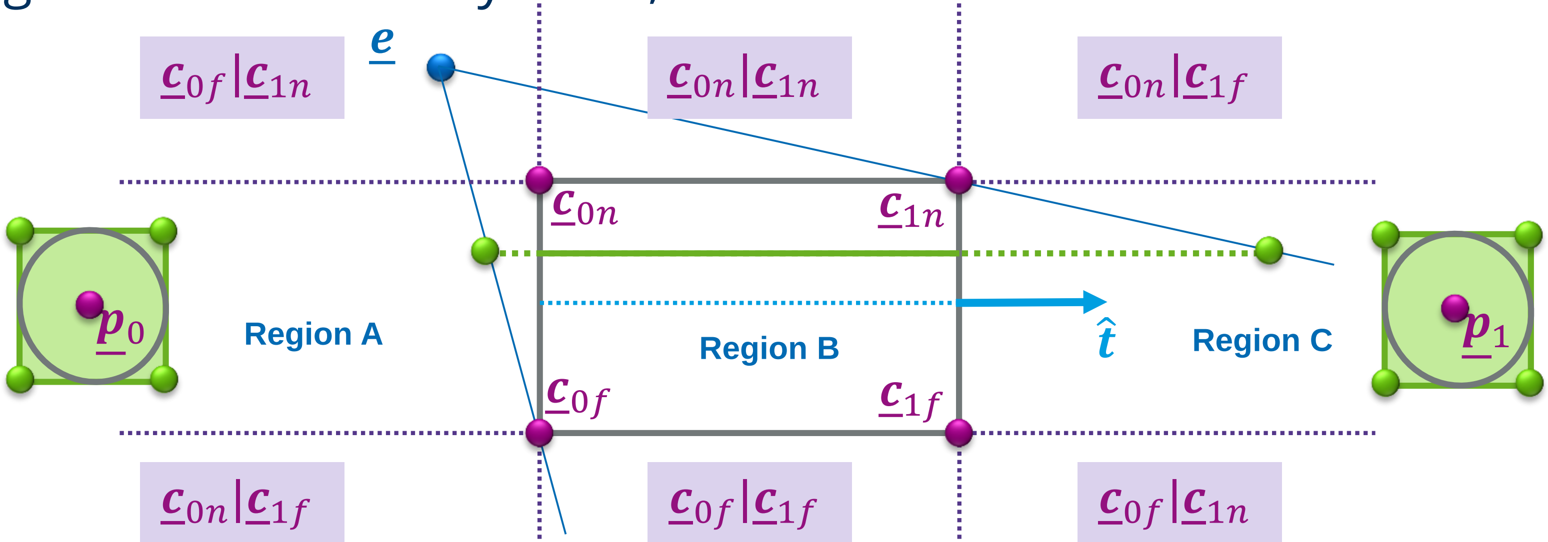




Cylinder Raycasting

Silhouette Covering Quad

- Construction from previous slide only valid if eye $\underline{e}$ is outside of cylinder barrel and inside of both half spaces.
- The figure below shows all possible cases where $\underline{e}$ can be with respect to cylinder. Previous approach can be applied with the extension corners shown in figure except if $\underline{e}$ is in regions A/B/C:
- In regions A & C silhouette is a circle that can be covered by quad
- Region B is inside of cylinder, where whole screen is covered



Particles

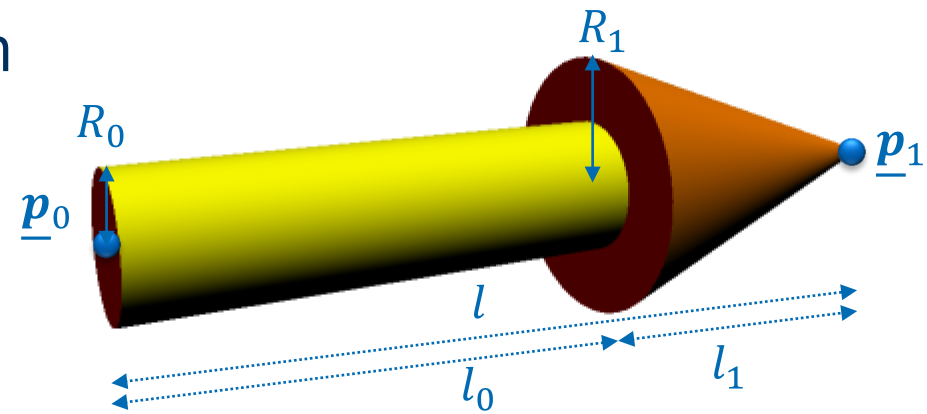
SHADER BASED RAYCASTING

ARROWS

Arrow Glyphs – Shape from length

Computer Graphics and Visualization

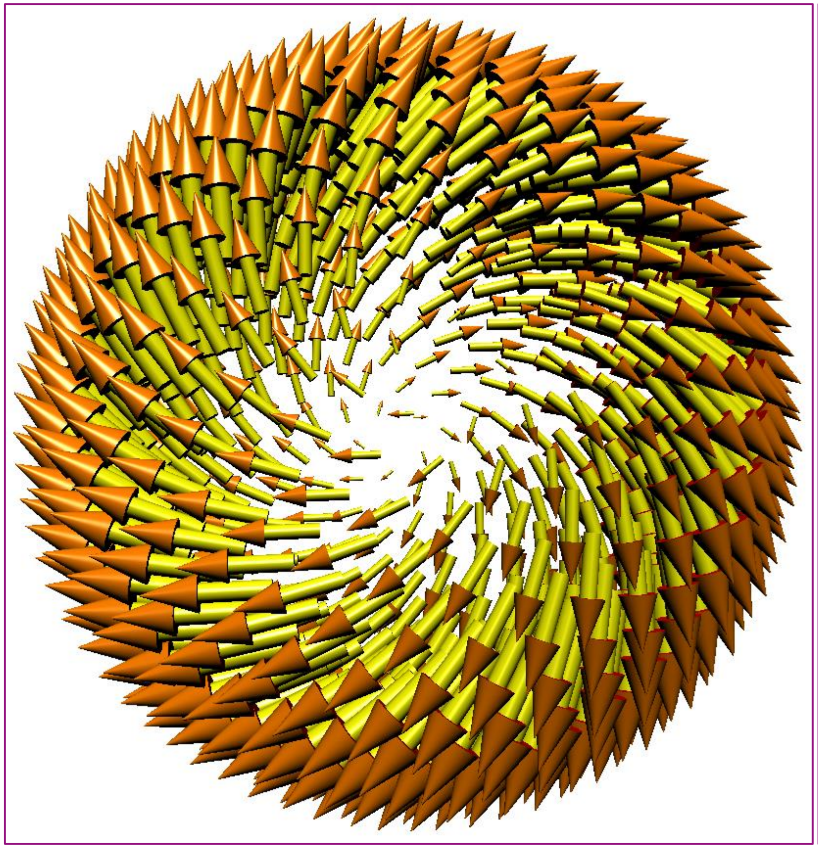
- Arrow glyphs are used to visualize vector quantities
- Arrow shape is defined from
  - $\underline{\boldsymbol{p}}_0$ and $\underline{\boldsymbol{p}}_1$
  - 2 radii R_0 and R_1
  - head length l_1 ($l_0 = l - l_1$)



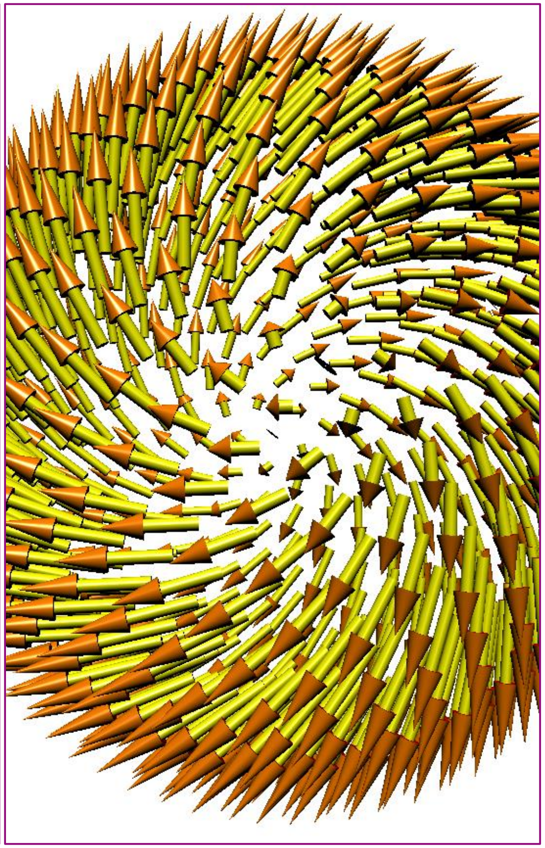
- Given to be visualized vector, there are different strategies to adapt shape to vector length l:
  1. radii and head length relative to l:
     e.g.: $R_0 = 7{,}5\% \cdot l$, $R_1 = 15\% \cdot l$, $l_1 = 45\% \cdot l$
  2. radii fixed, head length relative:
     e.g.: $R_0 = const$, $R_1 = 2 \cdot R_0$, $l_1 = 45\% \cdot l$
  3. radii and head length fixed:
     e.g.: $R_0 = const$, $R_1 = 2 \cdot R_0$, $l_1 = 1.5 \cdot R_1$

Arrow Glyphs – Shape from length

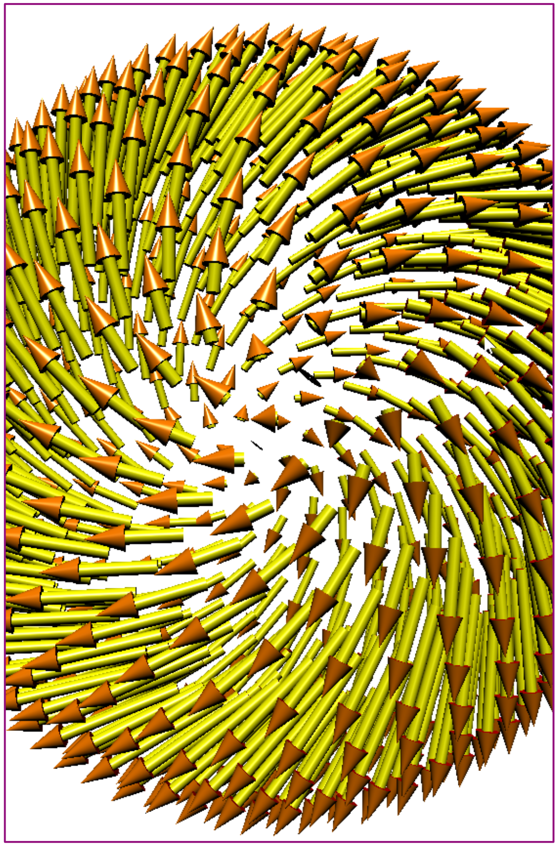
Comparison of Strategies



$R_0 = 7{,}5\% \cdot l$
$R_1 = 15\% \cdot l$
$l_1 = 45\% \cdot l$



$R_0 = const$
$R_1 = 2 \cdot R_0,$
$l_1 = 45\% \cdot l$



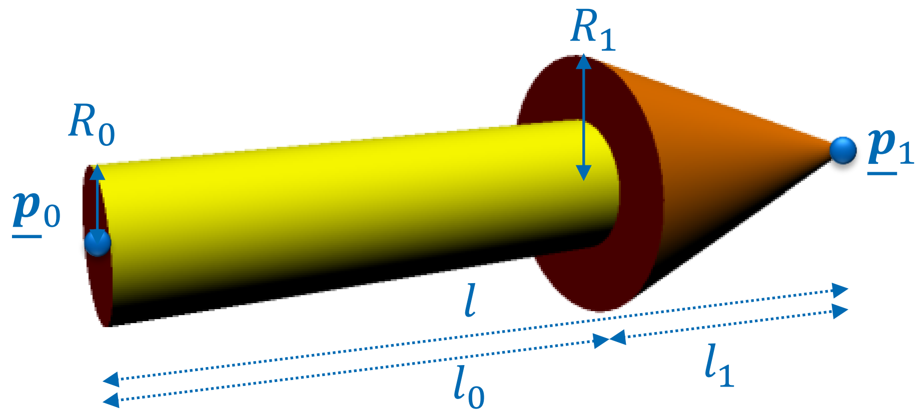
$R_0 = const$
$R_1 = 2 \cdot R_0,$
$l_1 = 1.5 \cdot R_1$

Arrow Glyphs – Shape from length

$R_0 = 7{,}5\% \cdot l$
$R_1 = 15\% \cdot l$
$l_1 = 45\% \cdot l$

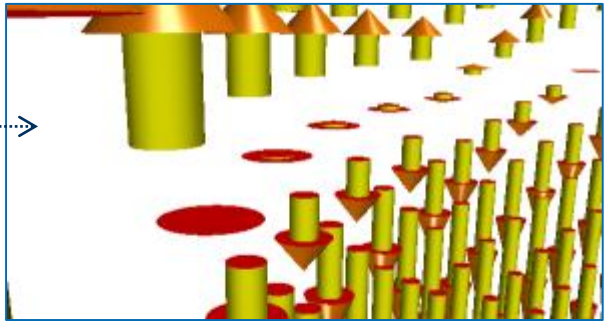
$R_0 = const$
$R_1 = 2 \cdot R_0,$
$l_1 = 45\% \cdot l$

$R_0 = const$
$R_1 = 2 \cdot R_0,$
$l_1 = 1.5 \cdot R_1$

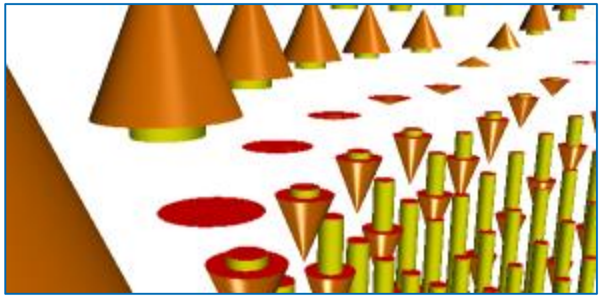


Special cases:

1. radii and head length relative:
   - exception for $l = 0$
2. radii fix, head length relative:
   - exception for $l = 0$,
   - arrow can become very obtuse:



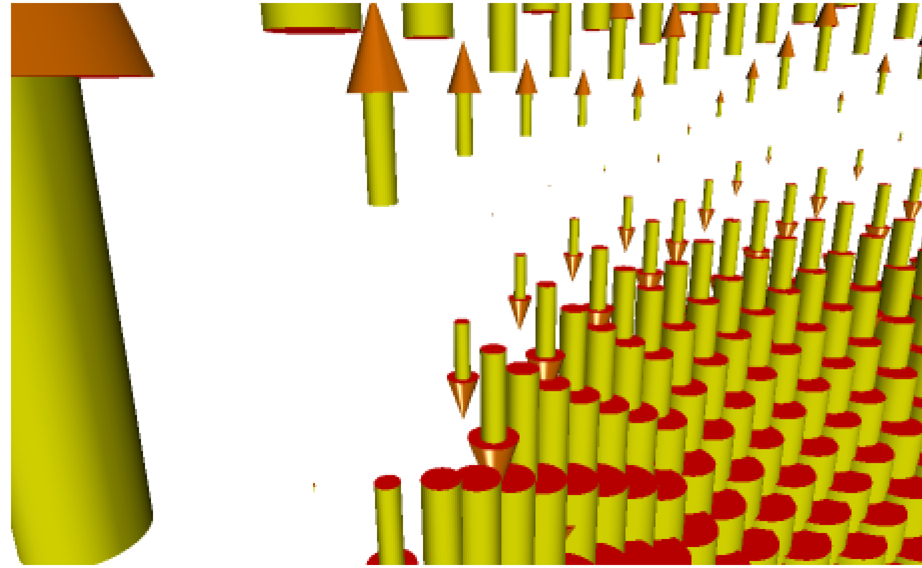
3. radii and head length fix:
   - exception for $l = 0$,
   - exception for $l_1 > l$



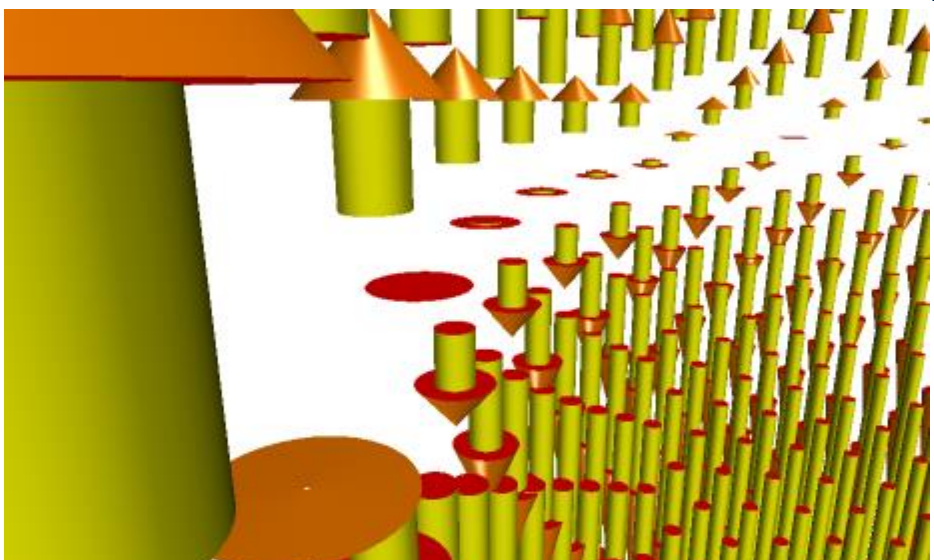
Arrow Glyphs – Shape from length

Comparison of special cases

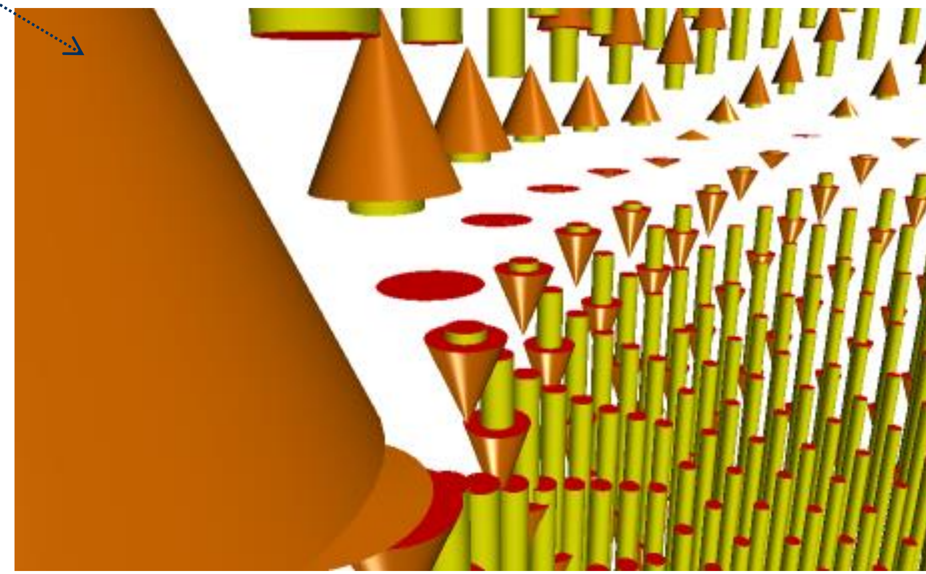
$R_0 = 7{,}5\% \cdot l$
$R_1 = 15\% \cdot l$
$l_1 = 45\% \cdot l$



$R_0 = const$
$R_1 = 2 \cdot R_0,$
$l_1 = 45\% \cdot l$

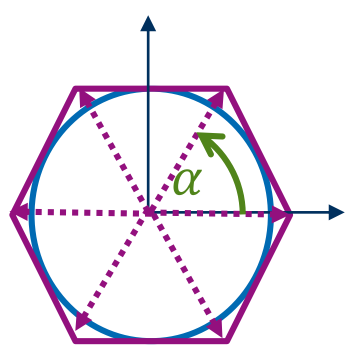


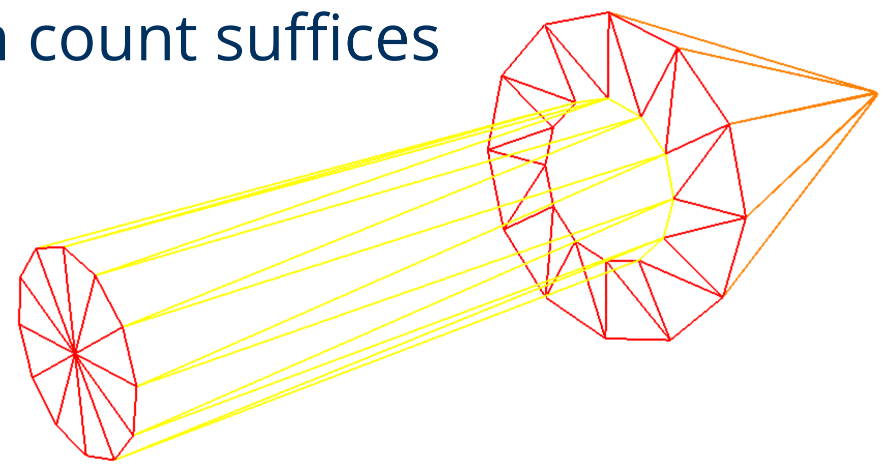
$R_0 = const$
$R_1 = 2 \cdot R_0,$
$l_1 = 1.5 \cdot R_1$



Arrow Glyphs – Tessellation

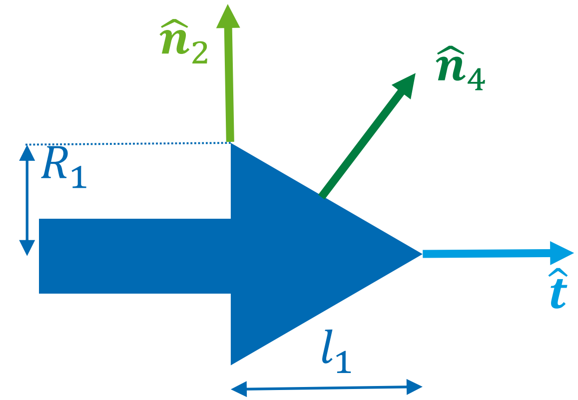
- Tessellation is a reasonable option for cylinder and arrows as subdivision is only necessary radially
- To avoid limitiations on output vertex count of geometry-shader, tessellation shader or instancing can be used instead.
- Scaling radial vectors with $\frac{1}{\cos\frac{\alpha}{2}}$ covers the silhouette which is useful for arrow raycasting approaches, where a small subdivision count suffices



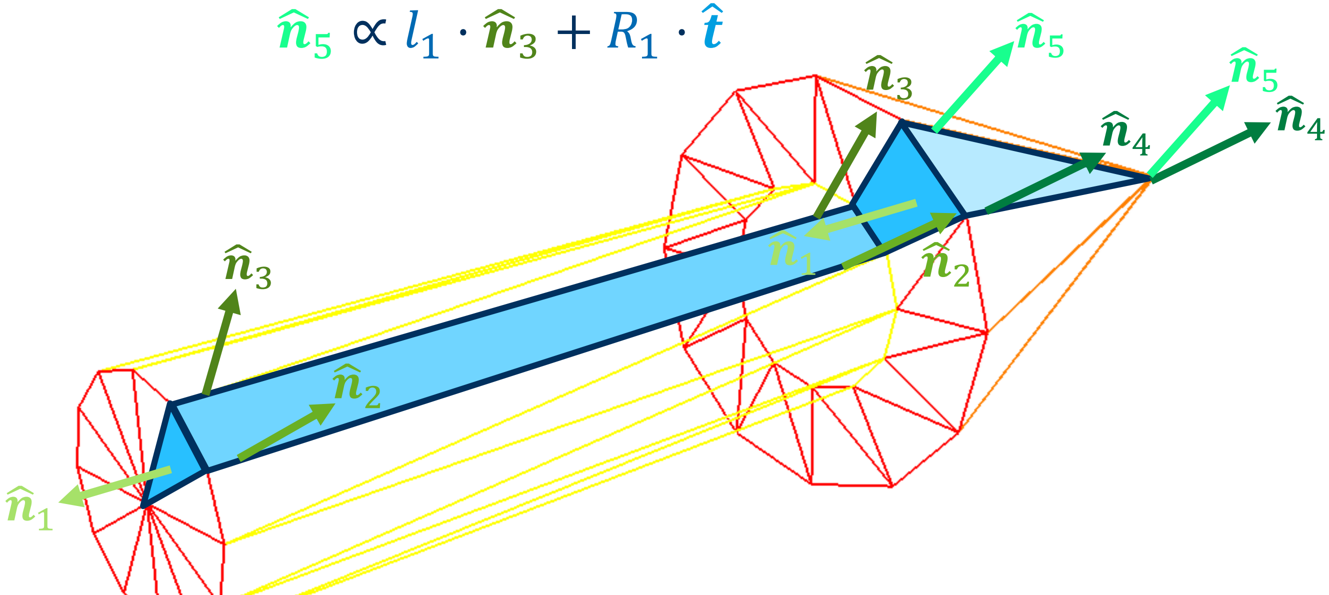


Arrow Glyphs – Tessellation

- When using instancing, per instance the blue polygon strip is generated: (from left to right)
  - 1 triangle with constant normal $\hat{\boldsymbol{n}}_1$
  - 1 quad with 2 normals $\hat{\boldsymbol{n}}_2$ and $\hat{\boldsymbol{n}}_3$
  - 1 quad with constant normal $\hat{\boldsymbol{n}}_1$
  - 1 **quad** with 2 normals $\hat{\boldsymbol{n}}_4$ and $\hat{\boldsymbol{n}}_5$ **?**

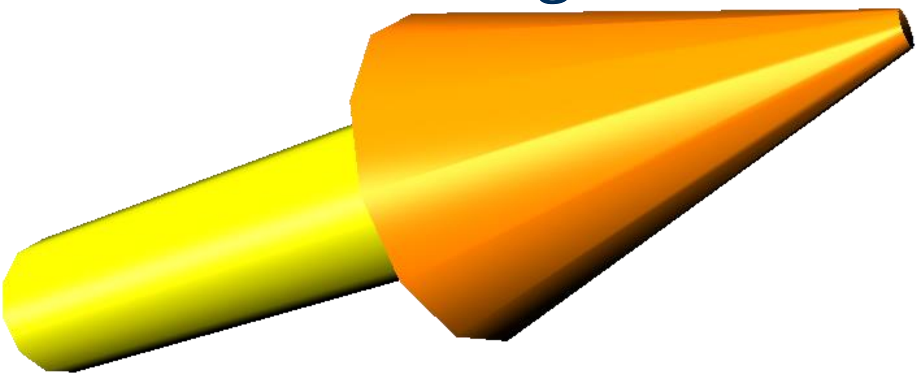


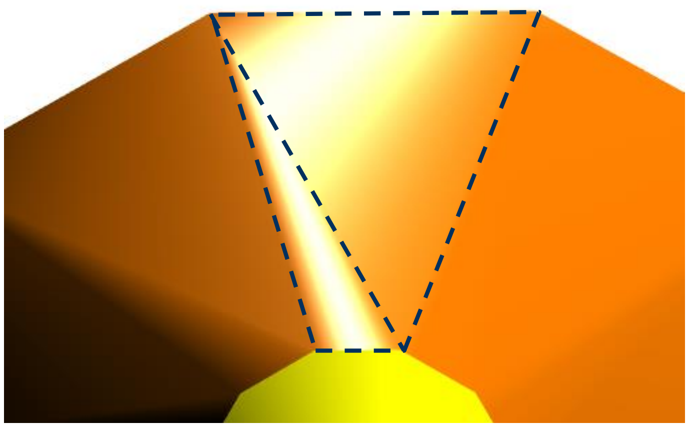
$$\hat{\boldsymbol{n}}_4 \propto l_1 \cdot \hat{\boldsymbol{n}}_2 + R_1 \cdot \hat{\boldsymbol{t}}$$
$$\hat{\boldsymbol{n}}_5 \propto l_1 \cdot \hat{\boldsymbol{n}}_3 + R_1 \cdot \hat{\boldsymbol{t}}$$



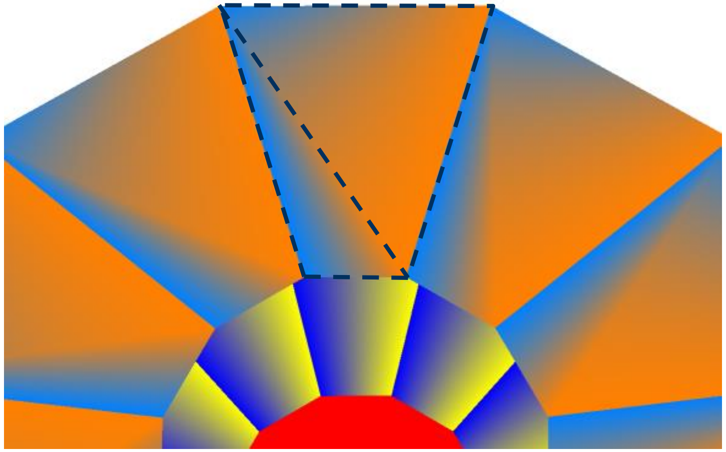
Arrow Glyphs – Tessellation

- Despite using a quad with 4 normals, shading of the arrow head is not smooth
- same problem for color interpolation
- This is due to decom-position of quad into 2 very badly shaped triangles





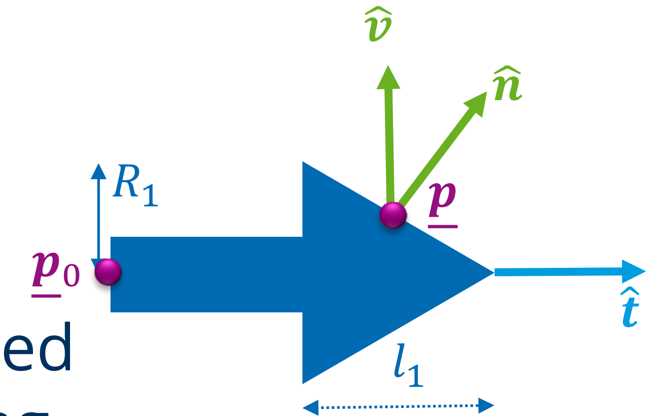
Illumination artefacts due to splitting of quad into badly shaped triangles



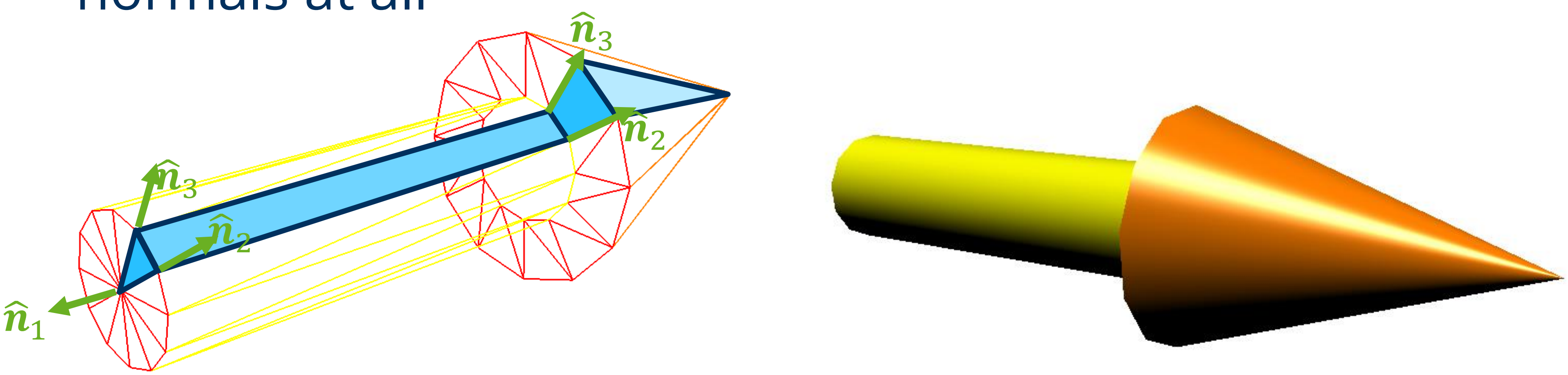
same artefacts for color interpolation

Arrow Glyphs – Tessellation

- Surface normal needs to be recomputed in fragment shader:
  - $\hat{\boldsymbol{v}} = \text{normalize}\left[\left(\underline{\boldsymbol{p}} - \underline{\boldsymbol{p}}_0\right) - \left\langle \underline{\boldsymbol{p}} - \underline{\boldsymbol{p}}_0, \hat{\boldsymbol{t}} \right\rangle \hat{\boldsymbol{t}}\right]$
  - $\hat{\boldsymbol{n}} = \text{normalize}(l_1 \cdot \hat{\boldsymbol{v}} + R_1 \cdot \hat{\boldsymbol{t}});$



- With this, arrow head can be rendered with single triangle without specifying normals at all

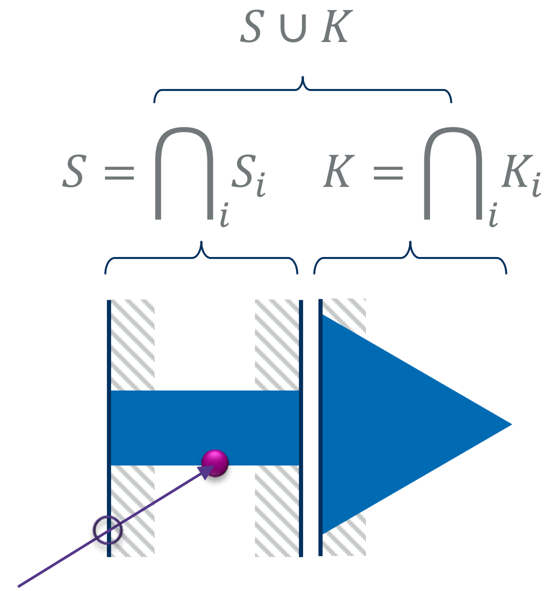


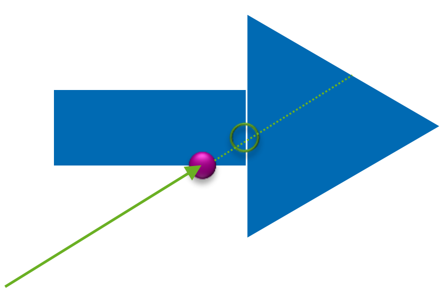
Arrow Glyphs – Ray Casting

- Silhouette cover by one/two screen aligned quads as in cylinder case or by radius corrected tessellation

Ray Intersection

- Arrow head is intersection of envelope of cone and half space
- Arrow tail is intersection of cylinder barell and 2 half spaces
- Arrow is union of head and tail
- *intersection:* first point of entry in all parts at same time (purpure ray)
- *union:* first point of entry in on part (green ray)



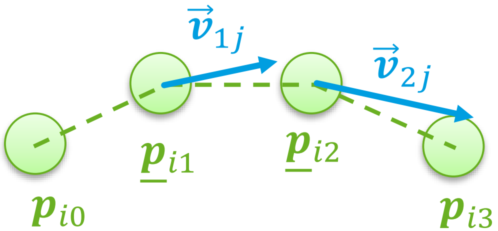


Sources and Types

TUBE BASE VISUALIZATION

Particle Data – Tubes

- Time dependent particle data P describes the states P_{ij} (also called particle instances) of $i = 1 \dots N$ particles over $j = 1 \dots n$ time frames with per particle life span $J_i = [j_0, j_1]$ and m further attributes $a_{k=1\dots m}$:
$$P = \left\{ P_{ij} = (\underline{\boldsymbol{p}}_{ij}, \vec{\boldsymbol{a}}_{ij}) \middle| \underline{\boldsymbol{p}}_{ij} \in \boldsymbol{R}^3, \vec{\boldsymbol{a}}_{ij} \in \boldsymbol{R}^m, j \in J_i \right\}$$
- By rearrangement of the data, we can extract N particle trajectories indexed over i as a sorted list of all particle instances of the ith particle:
$$T_i = \left(P_{ij} \middle| j \in J_i \right)$$
- If not given in the attributes, time derivatives can be estimated from finite differences:
$\vec{\boldsymbol{v}}_{ij} \approx \left(\underline{\boldsymbol{p}}_{i(j+1)} - \underline{\boldsymbol{p}}_{i(j-1)} \right) / \left(t_{j+1} - t_{j-1} \right)$ or
$\partial_t P_{ij} \approx \left(P_{i(j+1)} - P_{i(j-1)} \right) / \left(t_{j+1} - t_{j-1} \right)$



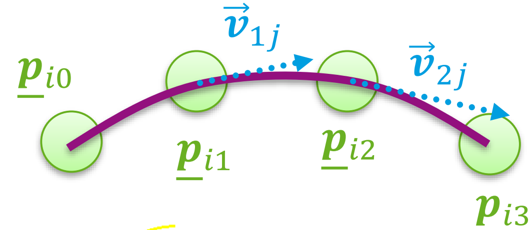
Tubes – Temporal Interpolation

- Trajectories are rendered with lines or tubes interpolated over time; where simplest is linear:

$$\forall t \in [t_j, t_{j+1}]: P_i(t) = (1-\lambda)P_{ij} + \lambda P_{i(j+1)}, \lambda = \frac{t - t_j}{\underbrace{t_{j+1} - t_j}_{\Delta_j}}$$

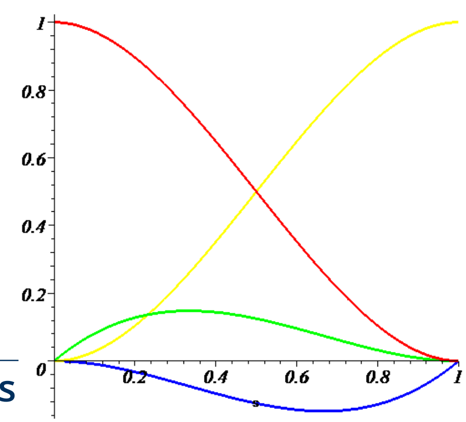
- If the temporal sampling is sparse, one can estimate the time derivatives and use Hermite interpolation:

$$P_i(t) = H_0^3(\lambda)P_{ij} \quad + \quad \Delta_j H_1^3(\lambda)\partial_t P_{ij} + \Delta_{j+1} H_2^3(\lambda)\partial_t P_{i(j+1)} + H_3^3(\lambda)P_{i(j+1)}$$



- with the Hermite polynomials

$$H_0^3(\lambda) = (1-t)^2(1-2t)$$
$$H_1^3(\lambda) = (1-t)^2 t$$
$$H_2^3(\lambda) = t^2(t-1)$$
$$H_3^3(\lambda) = t^2(3-2t)t$$

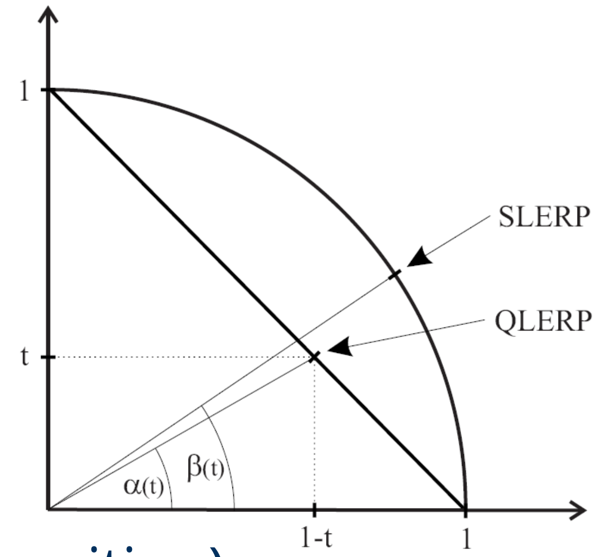


Tubes – Temporal Interpolation

- Orientations as quaternions q_j or orientation matrices $\boldsymbol{O}_j$ need to be interpolated differently
- Simplest approach is to correct linear interpolant:

$$^{\text{QLERP}}q(q_0,\lambda,q_1) = \frac{(1-\lambda)q_0 + \lambda q_1}{\|(1-\lambda)q_0 + \lambda q_1\|}$$

(for orientation matrix use polar decomposition)



- For two quaternions one can use the slerp (spherical linear interpolation) operations that produces uniform interpolation speed:

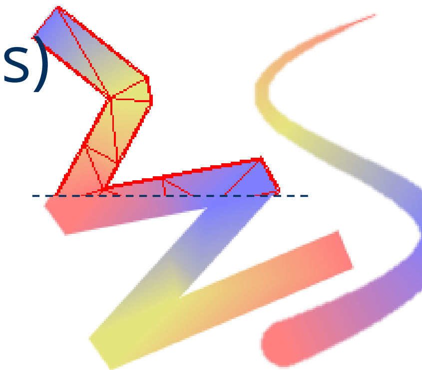
$$^{\text{SLERP}}q(q_0,\lambda,q_1) = \frac{\sin((1-\lambda)\theta)\, q_0 + \sin(\lambda\theta)\, q_1'}{\sin(\theta)}$$

with $q_1' = \begin{cases} q_1 & \langle q_0, q_1\rangle \geq 0 \\ -q_1 & otherwise \end{cases}$ and $\cos\theta = \langle q_0, q_1'\rangle$

Hermite interpolation: convert to Bezier re-presentation and build DeCasteljau algorithm on SLERP operations

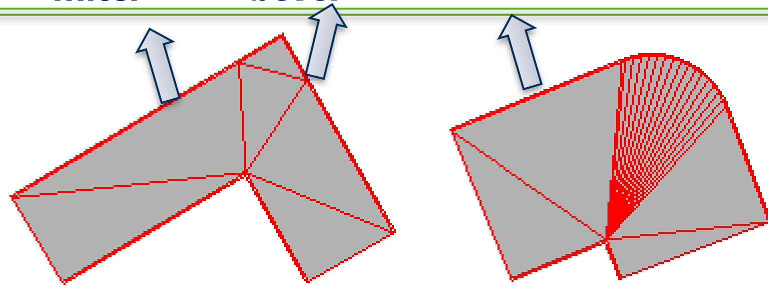
Tubes – 2D Rendering Primitives

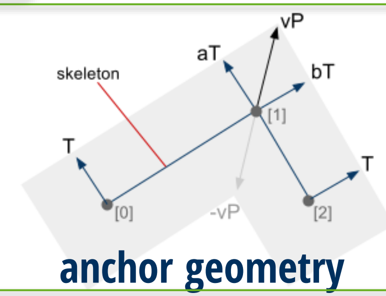
Computer Graphics and Visualization



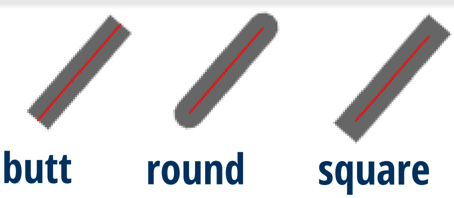
- Visual attributes: color and size (width/radius)
- **Thick lines** with the [vaserenderer]()
  - corner types

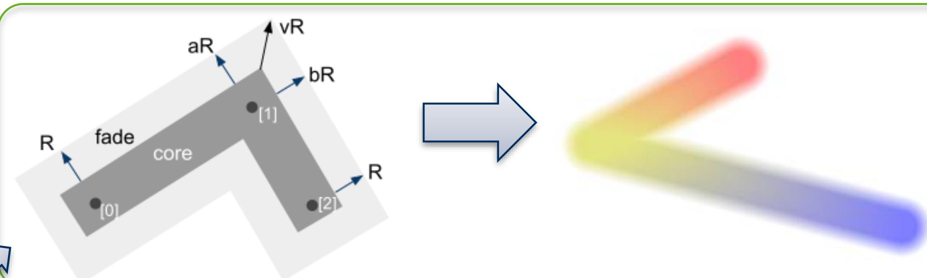
  - tessellation with help of anchor geometry



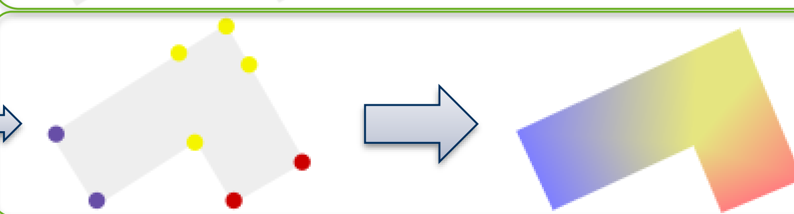


  - caps



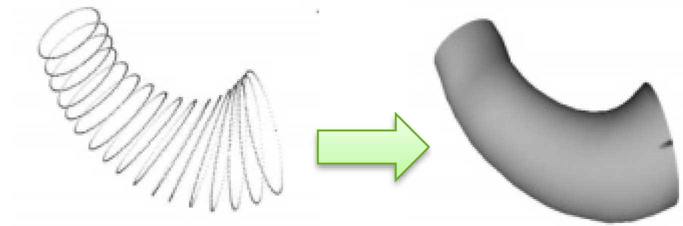


  - fade region for antialiasing
  - support for color mapping



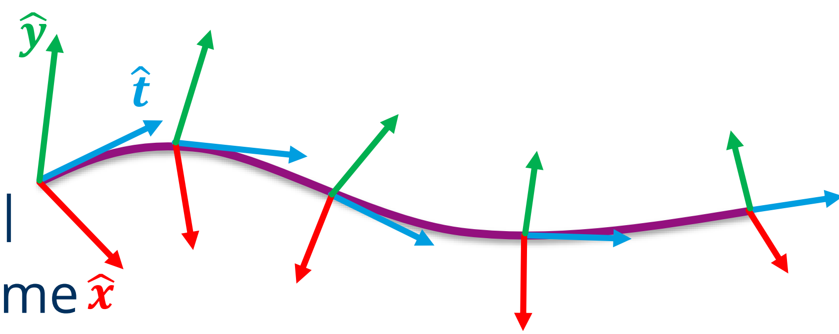
Tubes – 3D Rendering Primitives I

Generalized Cylinders

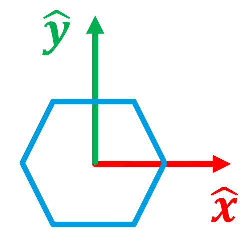


Images from [Handy Potter design]

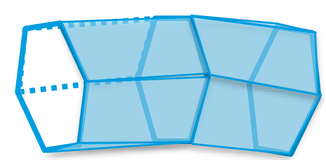
- Idea: trajectory is center line along which closed profile curve is swept
- Typical profiles: circle, ellipse, rectangle, super-quadric
- Approach:
  - Sample time t along trajectory
  - compute tangent $\hat{t}(t) = \vec{v}(t)/\|\vec{v}(t)\|$
  - extent with $\hat{x}$, $\hat{y}$ to orthonormal frame



  - minimize twist between frame along trajectory (optionally introduce twist by mapping some attribute)
  - at each sample, tessellate profile curve in xy-coordinates



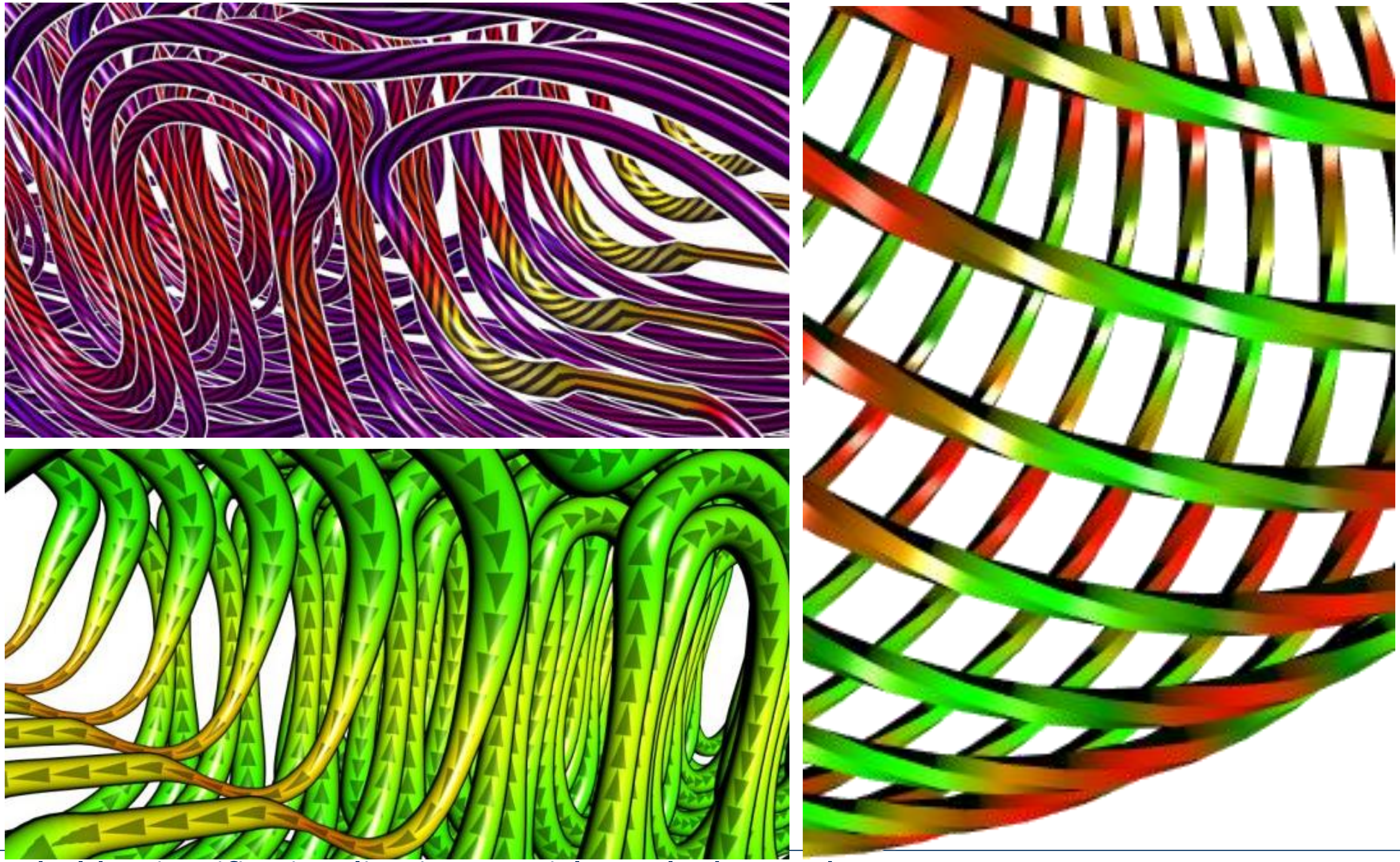
  - connect corresponding edges of successive samples with quad strip



Tubes – Generalized Cylinders

Computer Graphics and Visualization

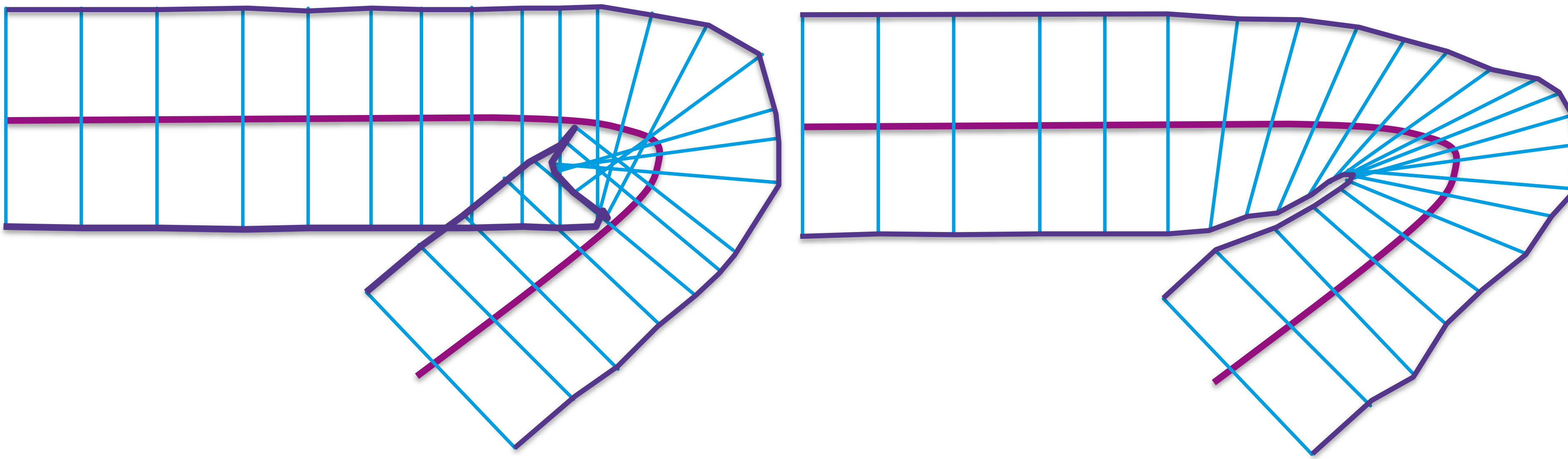
- optionally define texture coordinates with/without twist mapping to map lines or arrow glyphs to tube



Tubes – Generalized Cylinders

Limitations of generalized cylinders

- If the curvature radius of the center curve is smaller than the maximum radius of the profile curve, self-intersection can arise (see relative curvature condition)

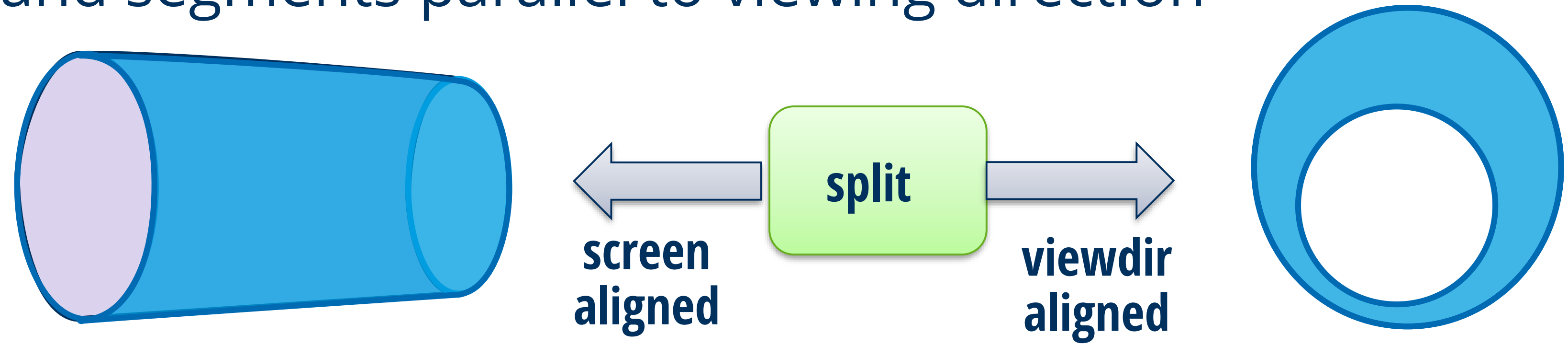


- The problem can be removed by allowing profile curves in planes not orthogonal to tangent:

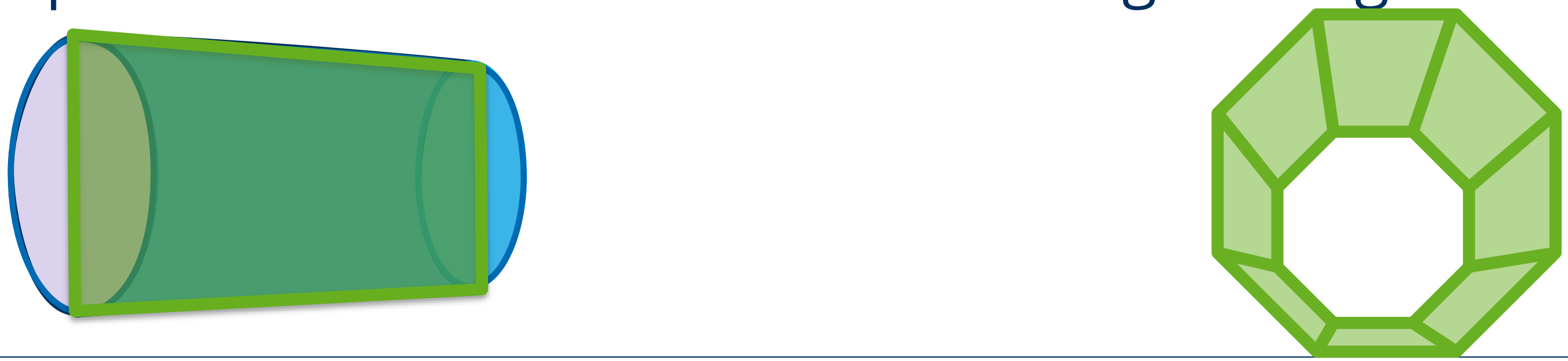
Skeleton-based Generalized Cylinder Deformation under the Relative Curvature Condition

Tubes – Generalized Cylinders

- For efficient rendering one splits the generalized cylinder tubes into segments tangential to screen plane and segments parallel to viewing direction

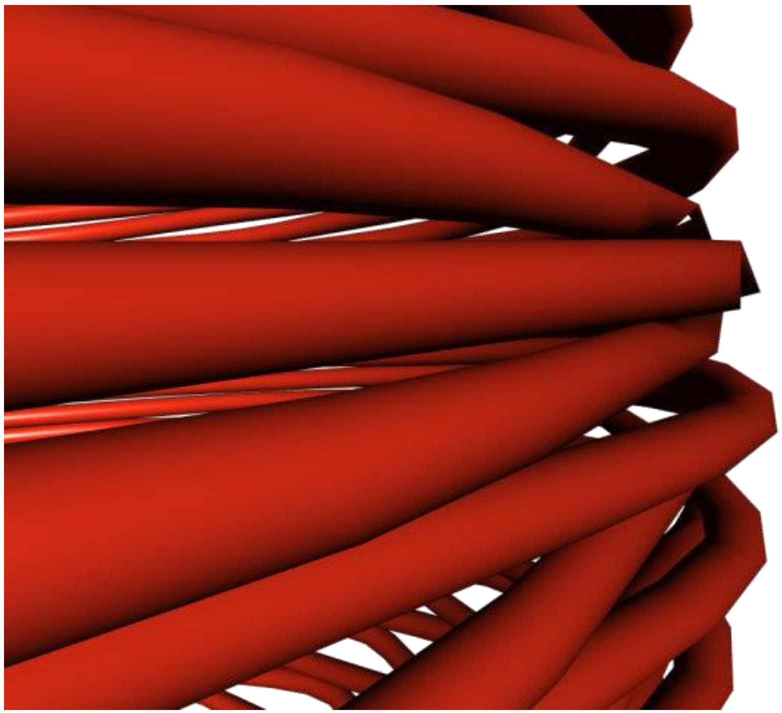


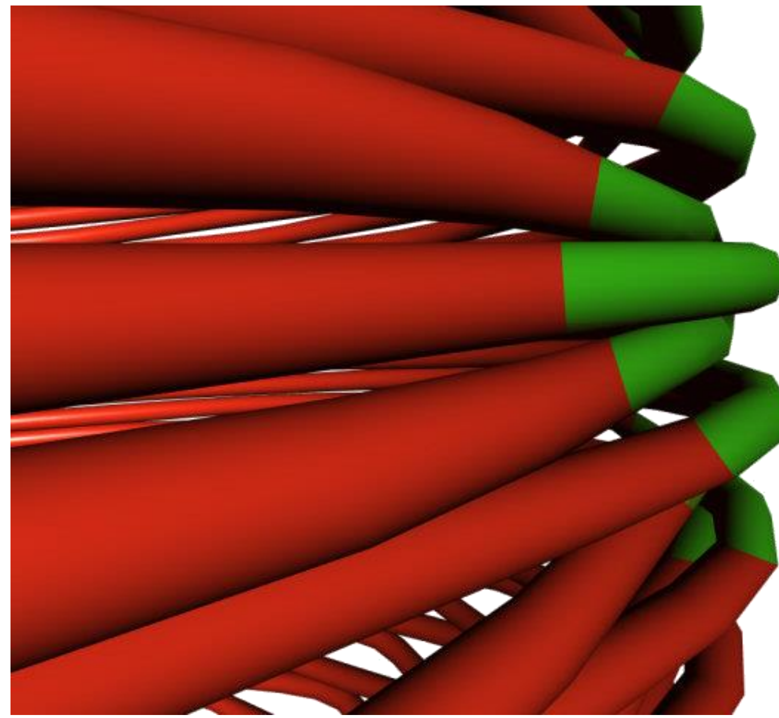
- hybrid rendering: screen aligned segments with single quad and tessellate view direction aligned segments



Tubes – Generalized Cylinders

- Comparison of the use of only single quad per generalized cylinder segment with hybrid approach

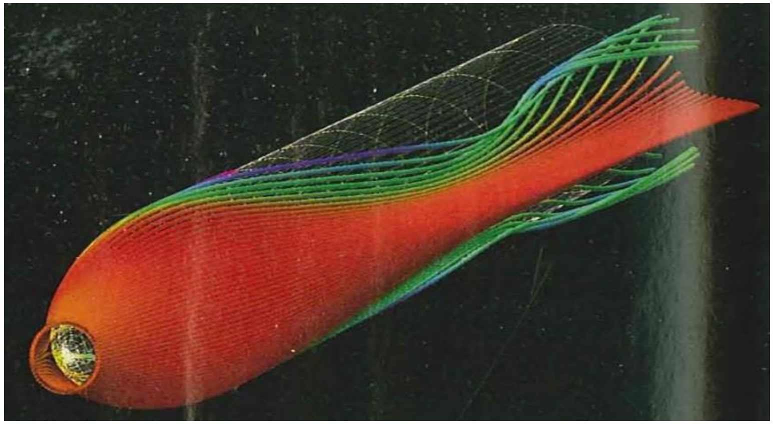


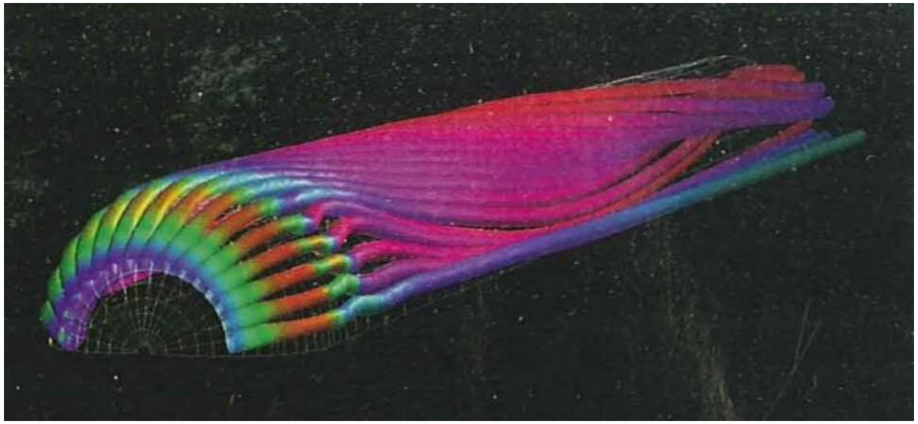


see paper: [Visualization with Stylized Line Primitives]

Tubes – Extrusion

- In diffusion tensor imaging of the brain, [tractography](#) integrates fiber bundles of nerves (nerve tracts)
- This is a trajectory with the diffusion tensor as attribute
- So called Hyperstreamlines are geometric representations of nerve tracts where ellipsoids representing the diffusion tensors are extruded along the nerve tracts.

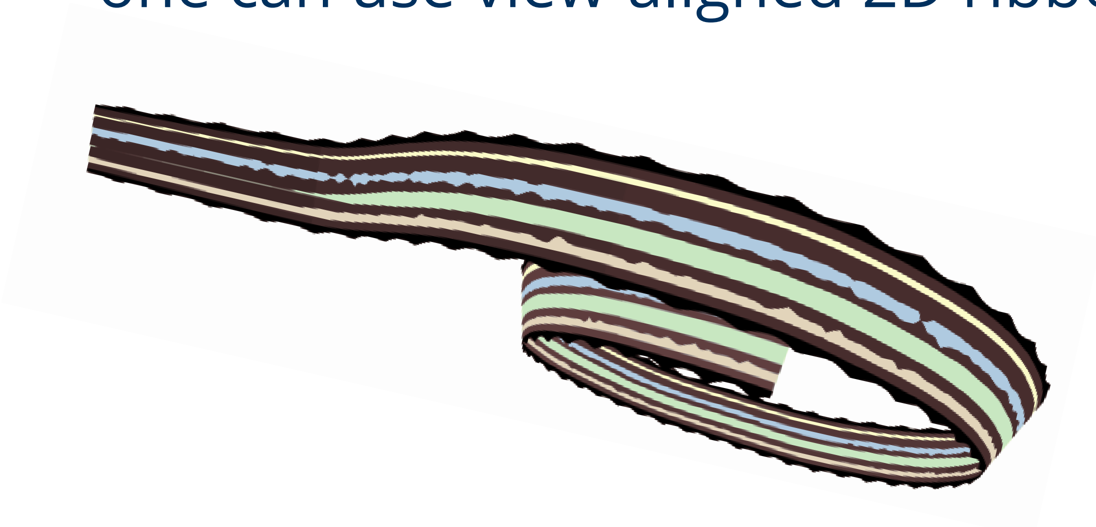




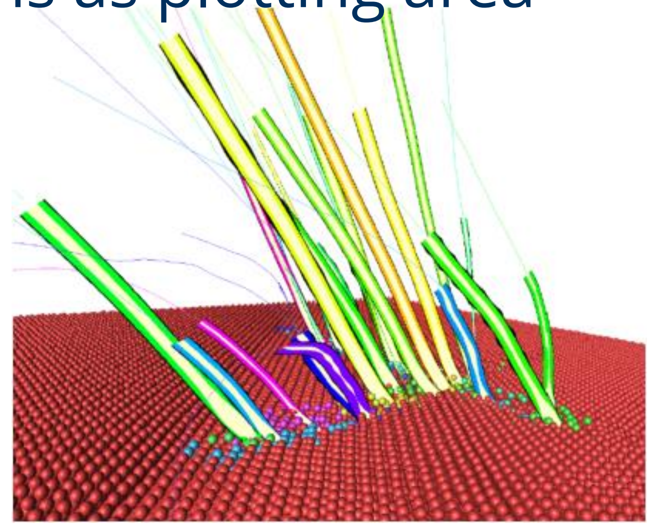
Example images from paper: Visualizing second-order tensor fields with hyperstreamlines

Tubes – Ribbons

- To visualize multiple scalar attributes along trajectories one can use view-aligned 2D ribbons as plotting area



partially screen aligned ribbon showing 4 scalar attributes in their spatial context



ribbons used to visualize attributes of particle clusters for a temporal context

- During 3D rotation of screen aligned ribbons, fold overs cannot be avoided completely (see also [demo here])
- a trade-off between screen alignment and rotational stability is necessary **see paper: Temporal Focus+Context for Clusters in Particle Data**