1. Jia's edge filtering algorithm

- **Vertex betweenness centrality (know what the symbols in the formula mean)**

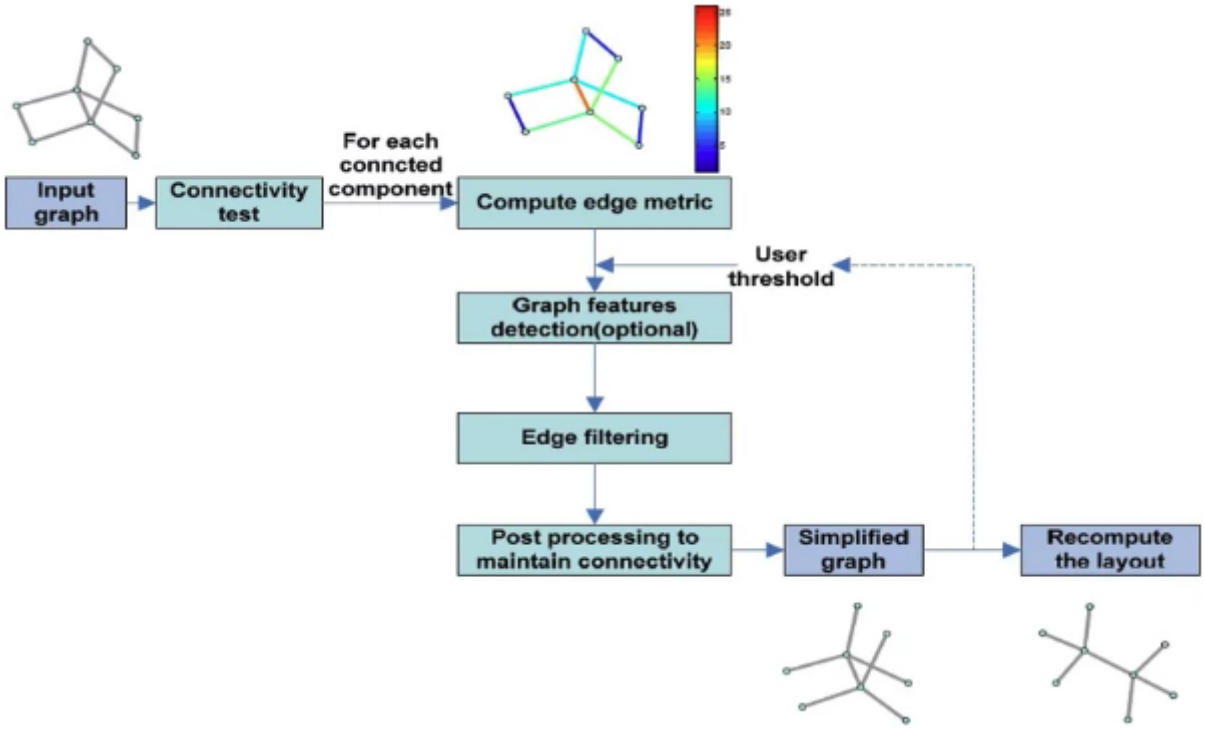
$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

  $\sigma_{st}(v)$ is the number of those paths pass through node v; σ_{st} is total number of shortest paths from s to t.

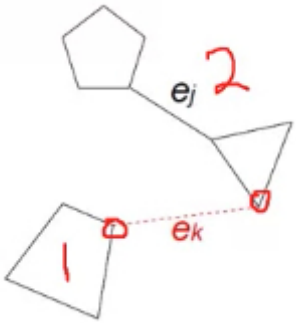
- **Edge betweeness centrality (know what the symbols in the formula mean).**

$$g(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$$

- **For which type of graphs will this work well**
  scale-free network: fraction of nodes with degree k follows power law $k^{-\alpha}$. Few hubs with many incident edges.
  Not feasible for large networks, relying on computing all-pairs shortest paths. Approximate by random sampling. Work not well for non-power law graph.
- **Know the steps of the algorithm**
  remove low BC edges, remain backbone.



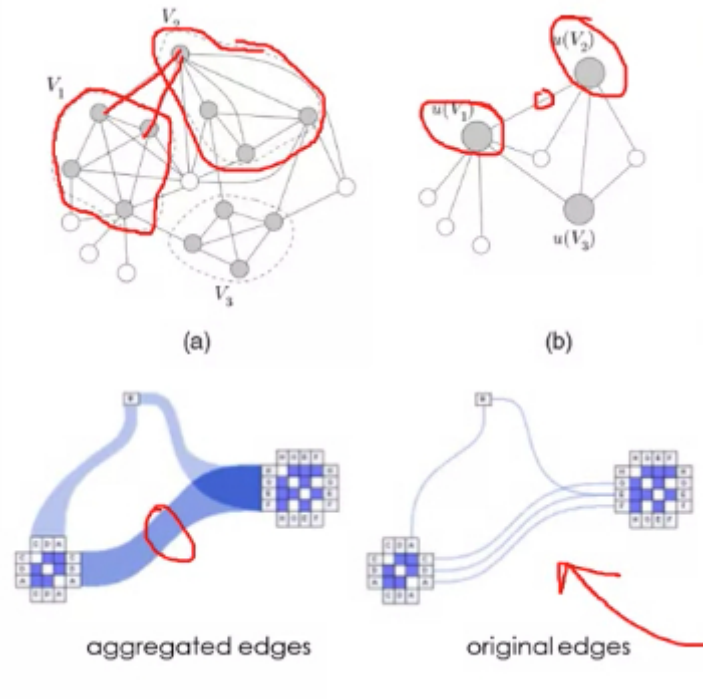
Graph feature detection: give user ability to change the metric to preserve certain features.
Post-process: restore discarded edges if removing it would cause disconnectivity.



Label vertices by component. If removed edge belongs to 2 components, restore it and unify their labels.
Edge filtering: sort BC, and remove those below the threshold.

2. Jia's edge bundling algorithm

- **Know the algorithm**
  Based on clustering labels.

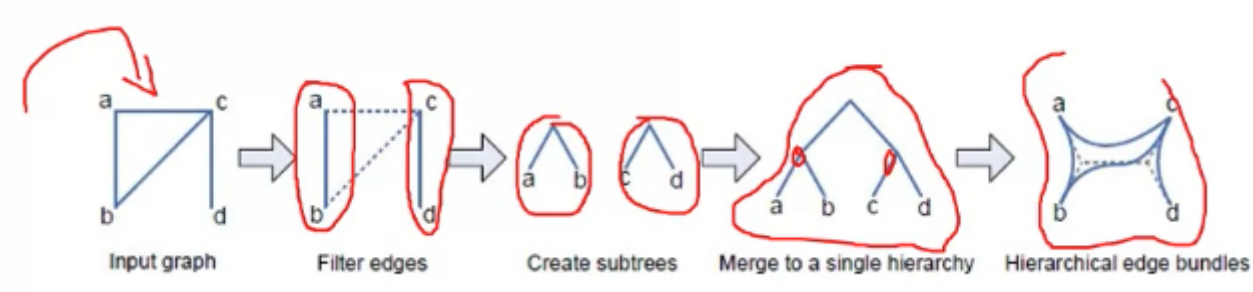


  Edge bundles are cluster of similar edges.
- **How is the vertex hierarchy constructed and used**
  Remove high-BC edges to discover clusters
  Vertices are places radially aroung circle. Root nodes of clusters in interior, not actually exist.

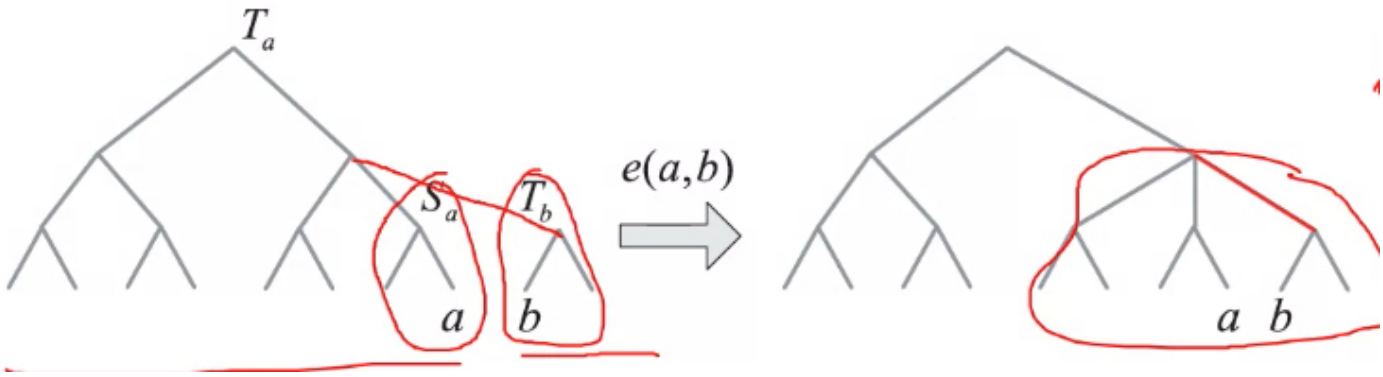
Leaves on the perimeter.

Edges are B-spline curves and control points to control hierarchies.

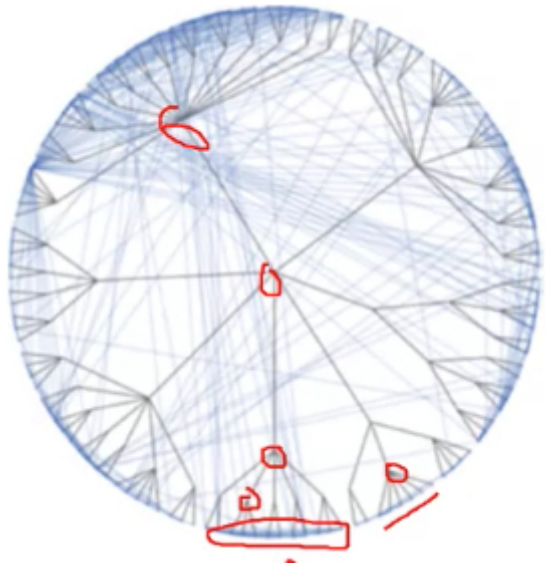
Balances Hierarchy construction: Filter edges from highest BC edges first: min(deg(a),deg(b))>1, BC(a,b)>1.



Scan removed edges in increasing order and merge subtrees connected by those edges. If Ta and Tb have same height, merge them under same new parent tree node. If Ta is taller than Tb, Sa is the unique subtree of Ta having a and have same height as Tb. Then merge them and Ta as parent of Tb.

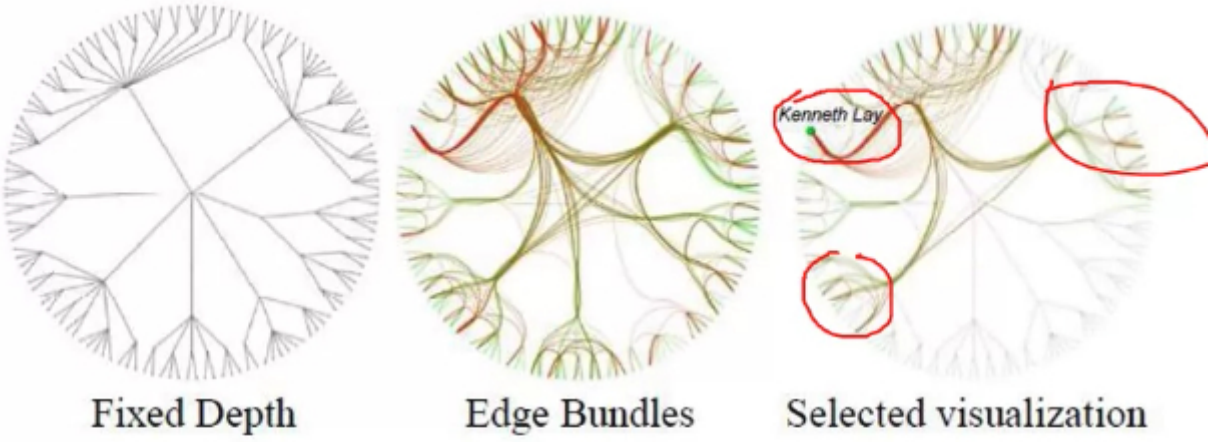


Bundle edges:



Gray lines shows hierarchies; blue lines shows actual edges.

Edge bundling to draw them as curves. Edges between 2 communities would be drawn with simiar curves.



Enron scandal 2001
389 e-mails, 132 employees
Red = sender, Green = recipient
Can select node to see which communities that person contacted

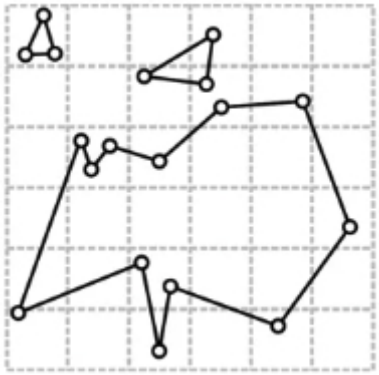
This method is intended for graphs with relatively few vertices that have many neighbors.

- **How is it used for community discovery**
  Compute edge betweeness centrality.
  Low BC connects nodes within a community; High BC edges connect communities. Remove high BC edges to discover communities.

3. Mesh Simplification

Vertex Clustering

- Cluster generation
  uniform 3D grid, map vertices to cluster cells.



hierarchical approach: bottom-up or top-down.

- compute a representative
  average/median vertex position
  median works better: a vertex of **original mesh chosest to the average position** of the vertices in the cluster. Treated as sub-sampling.
  error quadrics: optimal. squared distance to plane (implicit form of a plane: $ax + by + cz + d = 0$):

$$p = (x, y, z, 1)^T, \quad q = (a, b, c, d)^T$$

$$dist(q, p)^2 = (q^T p)^2 = p^T (qq^T) p =: p^T Q_q p$$

$$Q_q = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & b^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

(should be c^2 instead of b^2)
Compute vertice q to multiple planes.

- Sum distances to vertex' planes

$$\sum_i dist(q_i, p)^2 = \sum_i p^T Q_{q_i} p = p^T \left(\sum_i Q_{q_i} \right) p =: p^T Q_p p$$

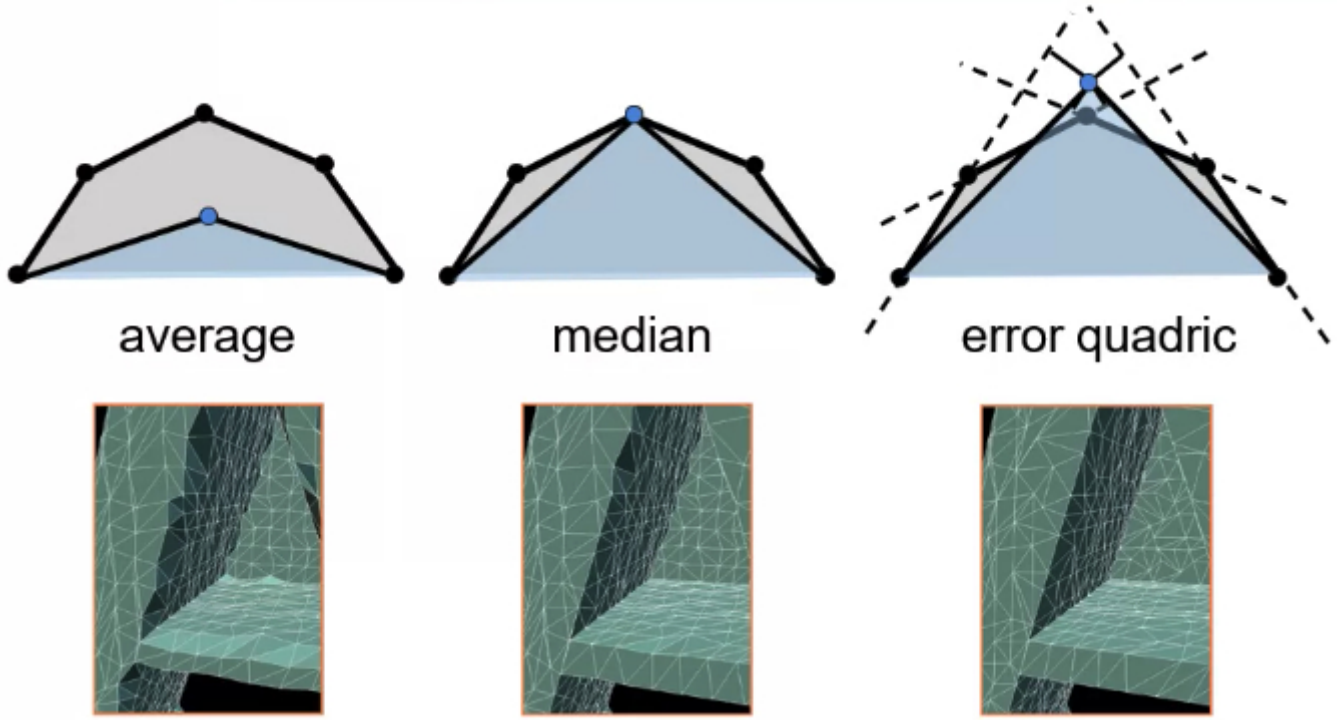
- Point that minimizes the error

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} p^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

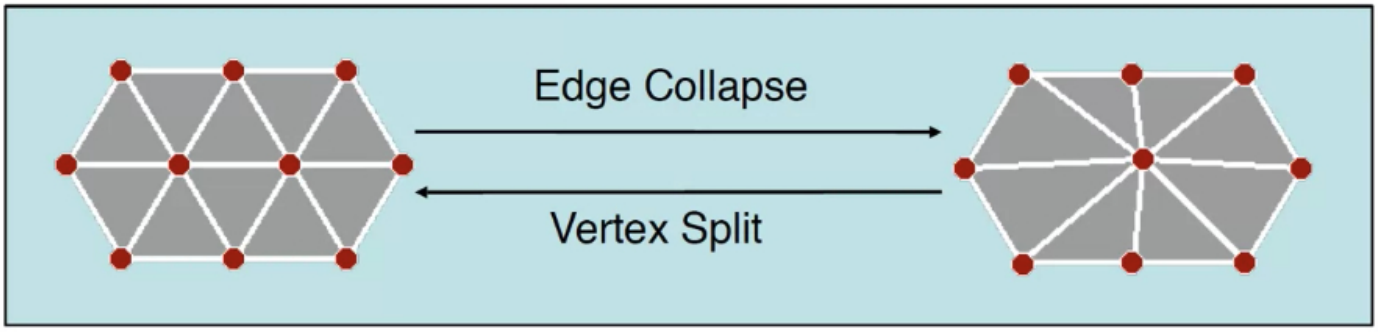
You can compute the sum of squared distances from p to N planes using a single 4x4 matrix

You simply sum up the N matrices Q_{q_i} component-wise and use it as shown here.

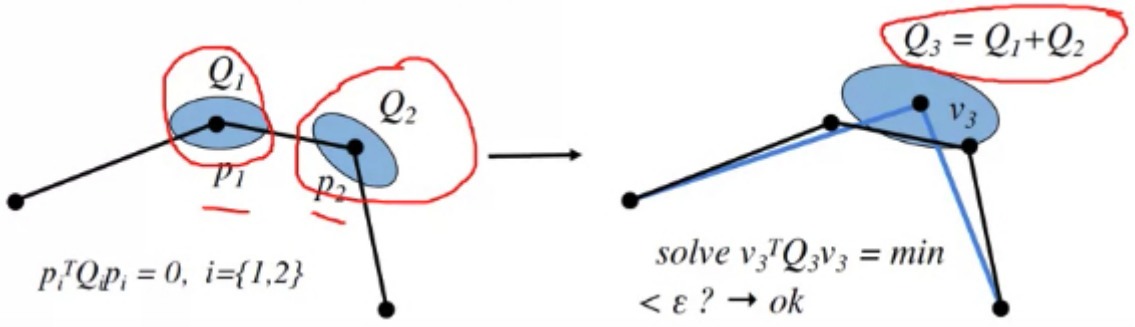
Q_p is sum of all 4x4 matrixes. The storage doesn't change.



- mesh generation
  regenerate the mesh: connect cluster q, p if there was an edge (p_i, q_j)
- Topology change
  If different sheets pass through one cell. Not manifold.
  To make manifold, use edge collapes

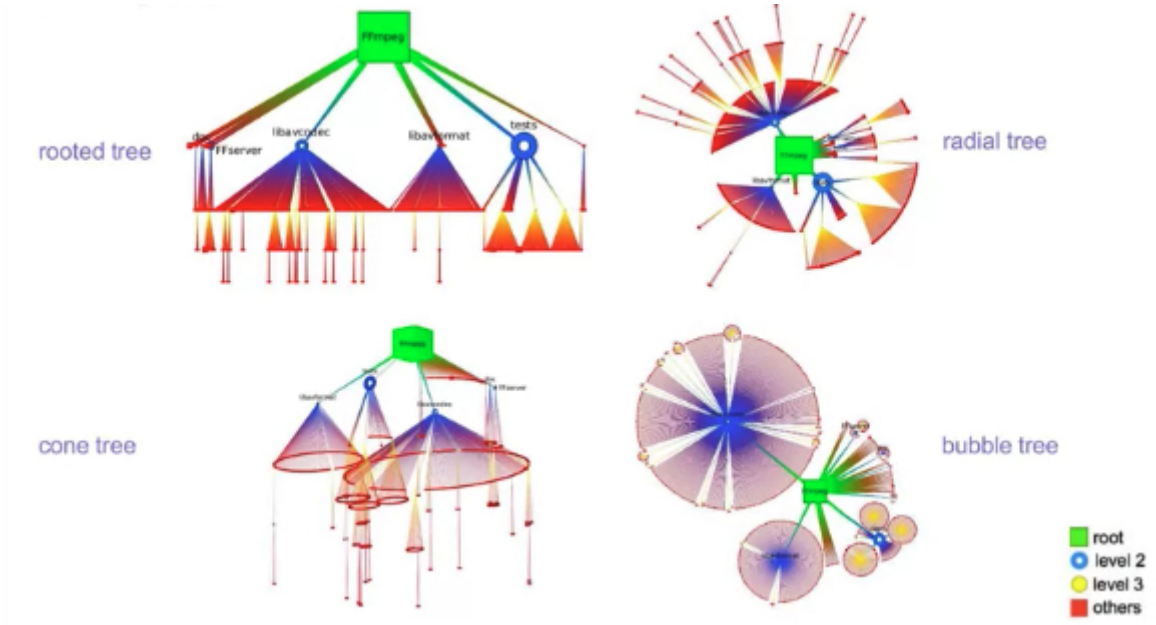


Choose edge to remove, merge vertices into singe vertex.
Place new vertex use error quadrics:



- Vertex clustering
  - fast, but difficult to control simplified mesh
  - topology changes, non-manifold meshes
  - global error bound, but often not close to optimum
- Iterative simplification with quadric error metrics
  - good trade-off between mesh quality and speed
  - explicit control over mesh topology
  - restricting normal deviation improves mesh quality

4. Tree Layout (know the basic algorithms)

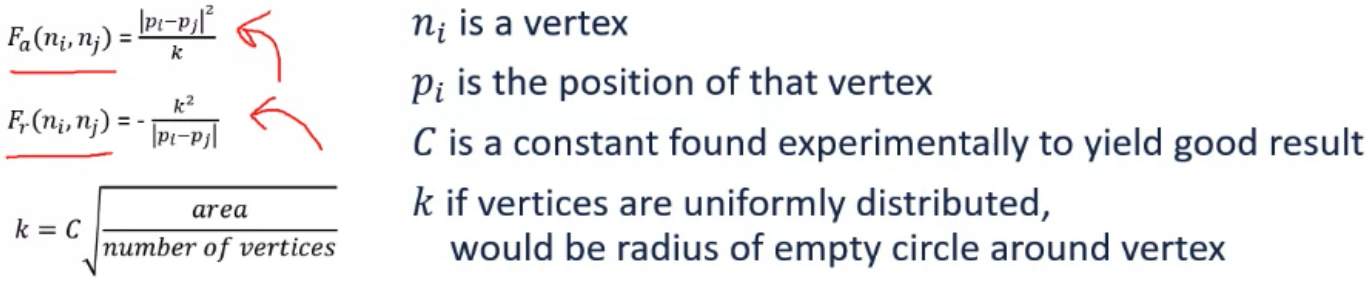


- **Rooted:** x axis is siblings; y axis is levels. Limited scalability: unbalanced aspect ratios can occur. Information is packed in the bottom line of the triangle. Directed acyclic graph could be represented in rooted form, but may have too much crossings and bad aspect ratios for large graph.
- **Radial**: use space better. Layout in concentric circles in particular level. scalability: 1-10k nodes. Nodes close to root get less space.
- **Bubble**: subtree gets own circle. Variable edge lengths. Hard to distinguish node depth. Better spread nodes in large tree.

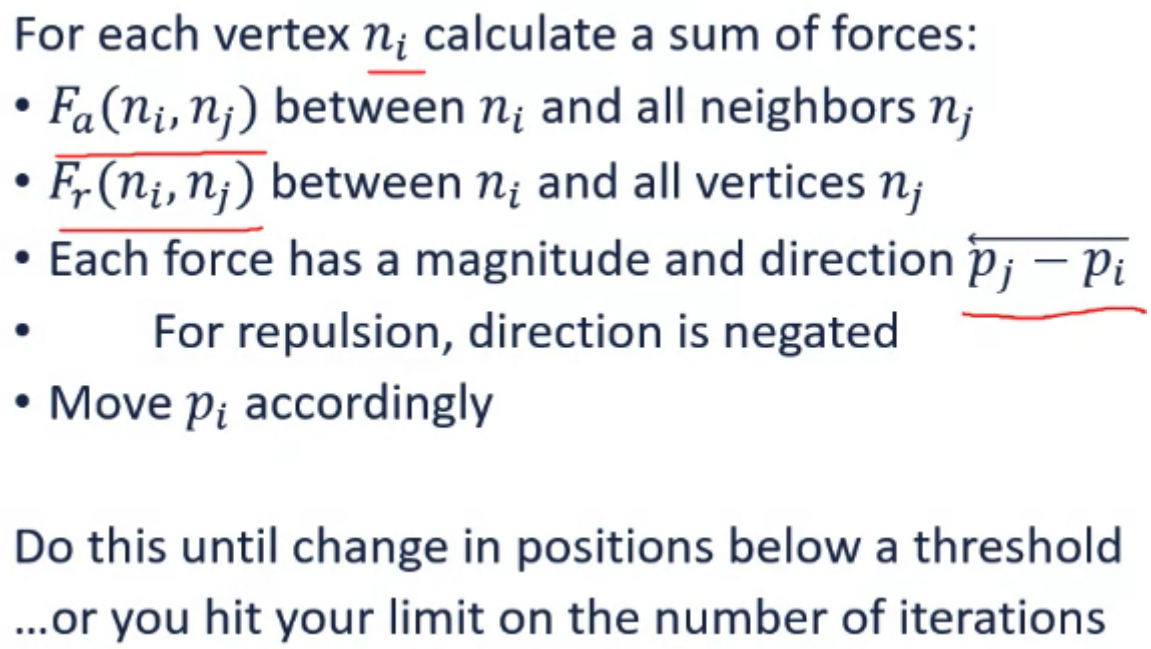
- **Cone**: subtree gets full cones. 3D show depths. Combine bubble tree with rooted tree. 3D is tricky, occlusion.

5. Force-directed layout

- **Know the FR algorithm**
  Repulse force (from all vertices) and attractive force (from vertices sharing edges).



  . k is place where fr and fa are balanced.



  . Apply scale factor to the amount of movement.
- **Know the running time and possible optimizations**
  Compuation time: slow $O(n^3)$
  Use spatial partitioning and compute repulsion for near-by nodes.
  Use multi-level computation: use average node to represent a group of vertices.
  Visual limit for large amounts of vertices: hairball problem
  Layout can be trapped in locally optimal.

6. Tractography

- **Know the algorithm**
  Fluid tend to diffuse along cells rather than across cells. Diffusion property can be represented

by a symmetric second order tensor.

$$\mathbf{T} = \begin{pmatrix} T^{xx} & T^{xy} & T^{xz} \\ T^{yx} & T^{yy} & T^{yz} \\ T^{zx} & T^{zy} & T^{zz} \end{pmatrix}$$

The 6 independent values (the tensor is symmetric)
The tensor elements vary continuously with spatial location

Trace streamline: choose seed with high **anisotropy** and stop when anisotropy is too low.
Measure of anisotropy:

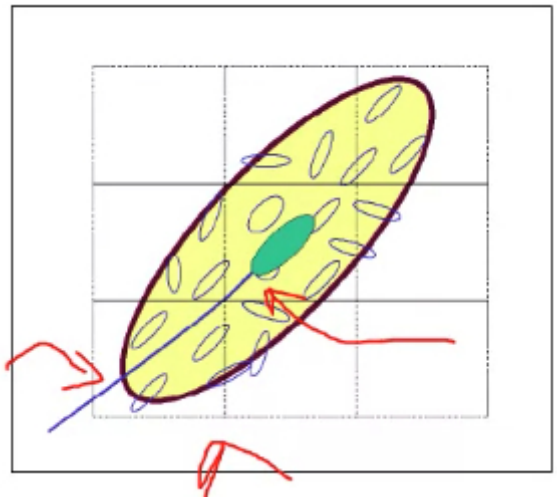
Measure

$$c_\ell = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3}$$

Values close to 1 indicate strong linear diffusion $\lambda 1 >> \lambda 2 \approx \lambda 3$

where λ is the eigenvalues of diffusion matrix.

First, use Moving least squares (MLS) to filter noisy data. Gaussian would blur directional information. MLS find **low-degree polynomial** to fit data in a **small** region, then replace data value at the point value of the polynomial (like what linear regression does).



. Yellow is the filtering region, small ellipsoid are interpolated tensor and the green one is the filter tensor constructed by MLS.

- **Understand the meaning of the tensor eigenvectors/values**
  3-D local axis direction of neuron fibers will be the **dominant** eigenvector.
- **How are tensors interpolated?**
  Reconstruct continuous tensor field using linear interpolation. Value of a tensor inside a voxel is a linear combination of 8 corner values. Interpolation is **tri-linear and component-wise**.

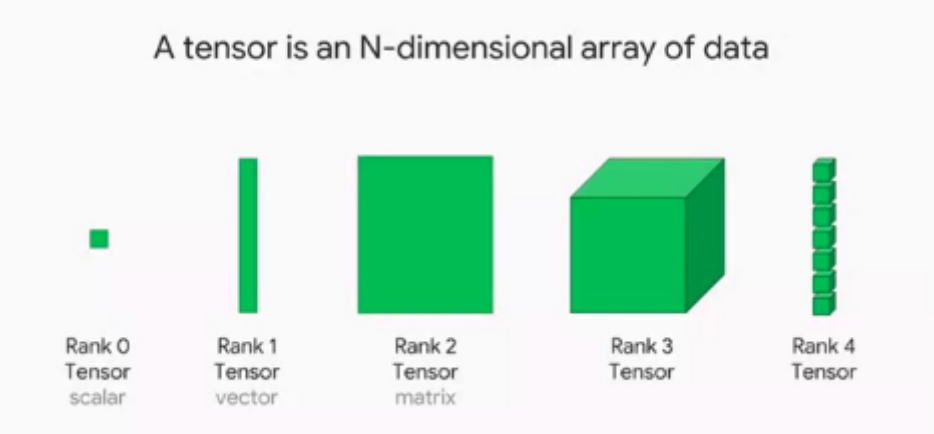
$$\begin{aligned}\mathbf{T}(x,y,z) = \mathbf{T}_{ijk}\,(1-x)(1-y)(1-z) + \\ + \quad \mathbf{T}_{i+1,jk}\,x(1-y)(1-z) + \mathbf{T}_{i,j+1,k}\,(1-x)y(1-z) \\ + \quad \mathbf{T}_{ij,k+1}\,(1-x)(1-y)z + \mathbf{T}_{i+1,j,k+1}\,x(1-y)z \\ + \quad \mathbf{T}_{i,j+1,k+1}\,(1-x)yz + \mathbf{T}_{i+1,j+1,k}\,xy(1-z) \\ + \quad \mathbf{T}_{i+1,j+1,k+1}\,xyz\end{aligned}$$

$x = x - x_{min} / x_{max} - x_{min}$

Cannot use component-wise interpolation directly on eigenvectors: a linear interpolation between 2 unit vectors is not a unit vector anymore.

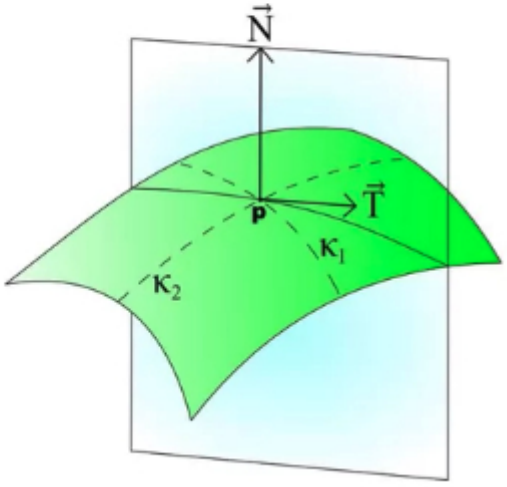
7. Tensors

- **What is a tensor?**
  A machine takes in some vectors and spits out other vectors in linear fashion. Describe the mapping between objects, including scalars, vectors, tensors.



NOT all matrices are tensors. Tensors have certain properties.

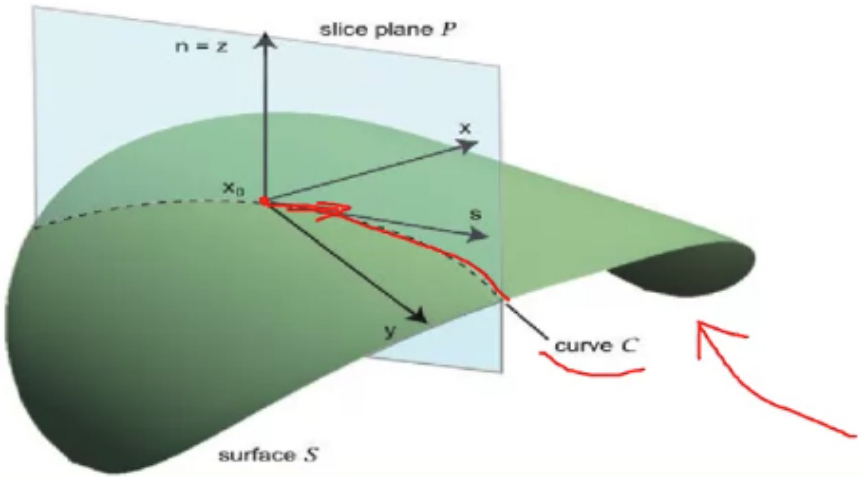
scalar: **magnitude** (of some signal at a point in space)
vector: **magnitude** and **direction** (of some signal at some point in space)
tensor: **variation** of **magnitude** (of some signal at some point in space)



- **Be able to construct and evaluate a Hessian matrix**

How to describe 2D curvature?

- 1D analogy: how quickly the normal $\mathbf{n}_S$ changes around x_0
- problem: we have a surface – in which direction to look for change?



We must compute

$$C(x,s)=\frac{\partial^2 f(x)}{\partial s^2}$$

for any direction s

General solution:

Describe S as an implicit function (i.e. the zero-level isosurface of a function)

$$S=\left\{x\in\mathbf{R}^3 \middle| f(x)=0\right\} \text{ for a given } \quad f:\mathbf{R}^3\to\mathbf{R}$$

Then, we still have

$$\frac{\partial^2 f}{\partial s^2}(x_0)=\mathbf{s}^T H\mathbf{s} \quad \text{where H is the 3x3 Hessian matrix} \quad H=\begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x\partial y} & \frac{\partial^2 f}{\partial x\partial z} \\ \frac{\partial^2 f}{\partial y\partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y\partial z} \\ \frac{\partial^2 f}{\partial z\partial x} & \frac{\partial^2 f}{\partial z\partial y} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix}$$

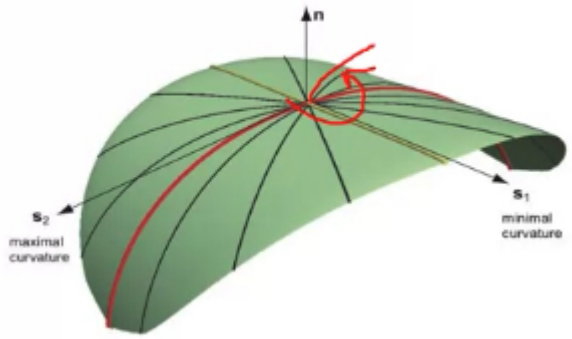
Conclusion

- A curvature tensor is fully described by a 3x3 matrix of 2nd order derivatives

f: mapping R^3 to R.

- **Understand the meaning of the anisotropy measures**

  Diffusion tensor: $D(x, s) = \partial^2 f(x)/\partial s^2$, where s is the diffusion direction s.



$$C(x,s)=\frac{\partial^2 f(x)}{\partial s^2}$$

- fix some point x_0 on the surface
- compute $C(x_0,s)$ for all possible tangent directions s at x_0
- denote α = angle of s with local coordinate axis x_0

So we have

$$\frac{\partial^2 f}{\partial s^2} = s^T H s = h_{11}\cos^2\alpha + (h_{12}+h_{21})\sin\alpha\cos\alpha + h_{22}\cos^2\alpha$$

Now, let's look for the values of α for which this function is extremal!

. To achieve extreme, $\partial C/\partial\alpha = 0$.

This is equivalent to a system of equations

$$\begin{cases} h_{11}\cos\alpha + h_{12}\sin\alpha &= \lambda\cos\alpha \\ h_{21}\cos\alpha + h_{22}\sin\alpha &= \lambda\sin\alpha, \end{cases}$$ which in matrix form is $H\mathbf{s} = \lambda\mathbf{s}$ or $(H-\lambda I)\mathbf{s}=0$

Since we're looking for the non-trivial solution $\mathbf{s}\neq\mathbf{0}$ this means

$$\det(H-\lambda I) = (h_{11}-\lambda)(h_{22}-\lambda) - h_{12}h_{21} = 0$$

Solving the above 2nd order equation in λ yields

- two real values λ_1, λ_2 eigenvalues (principal values) of tensor

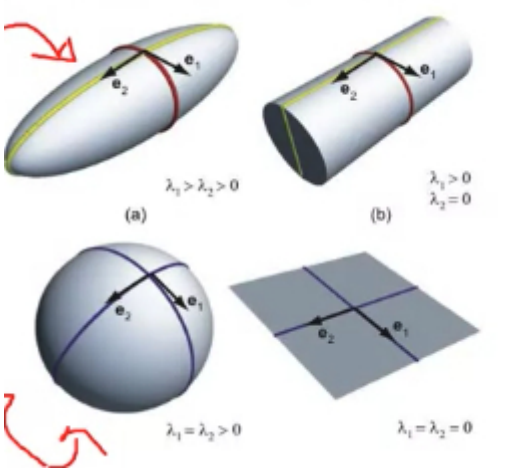
Plugging λ_1, λ_2 into $H\mathbf{s} = \lambda\mathbf{s}$ yields

- two direction vectors $\mathbf{s}_1, \mathbf{s}_2$ eigenvectors (principal directions) of tensor

. The 2x2 tensor gives us the principle directions in which tensor has minimal and maximal values.

For 3x3 tensor, 3 eigenvalues and 3 eigenvectors.

$\lambda_1, \mathbf{s}_1$	**major** eigenvector i.e. direction of strongest diffusion
$\lambda_2, \mathbf{s}_2$	**medium** eigenvector (no particular meaning)
$\lambda_3, \mathbf{s}_3$	**minor** eigenvector i.e. direction of weakest diffusion



a,b) all values ordered: unique eigendirections

c,d) equal eigenvalues: eigendirections not determined (any two orthogonal vectors tangent to surface are valid eigendirections)

Mean diffusivity: $\mu = (\lambda_1 + \lambda_2 + \lambda_3)/3$

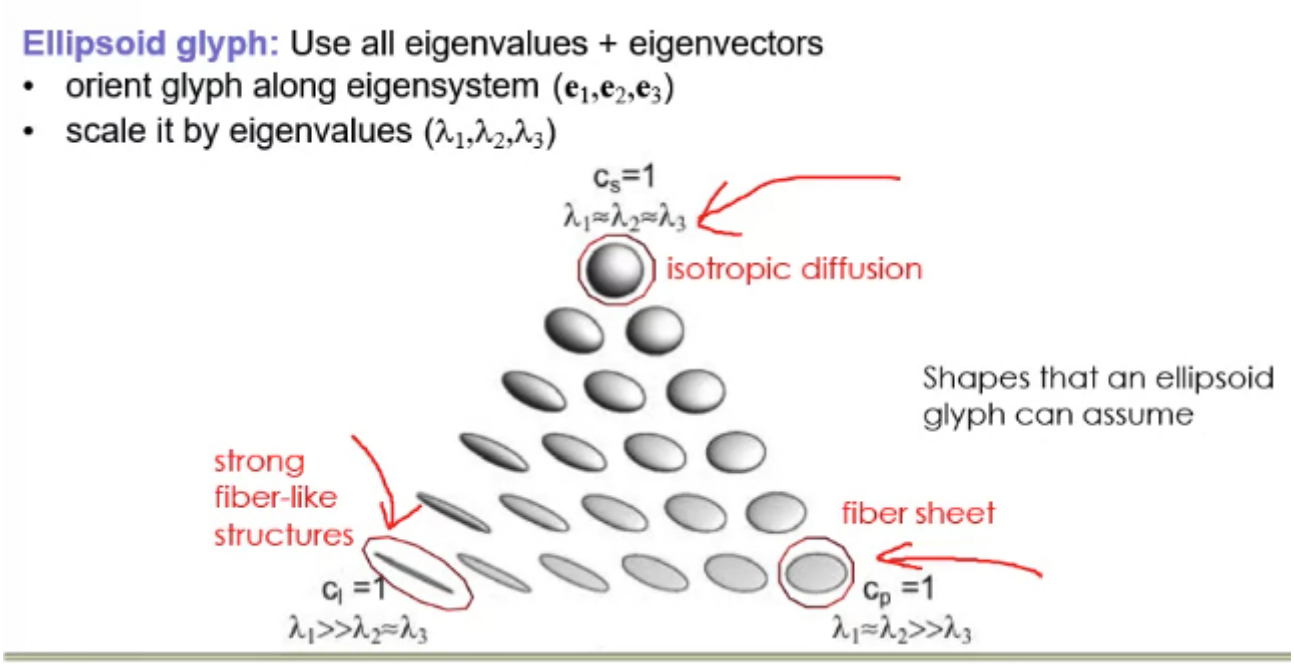
Look for strong difference in the eigenvalue magnitudes:

Fractional anisotropy $FA = \sqrt{\frac{3}{2}} \frac{\sqrt{\sum_{i=1}^{3} (\lambda_i - \mu)^2}}{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}$ where $\mu = \frac{1}{3}(\lambda_1 + \lambda_2 + \lambda_3)$

Relative anisotropy $RA = \sqrt{\frac{3}{2}} \frac{\sqrt{\sum_{i=1}^{3} (\lambda_i - \mu)^2}}{\lambda_1 + \lambda_2 + \lambda_3}$

. Large part means fiber.

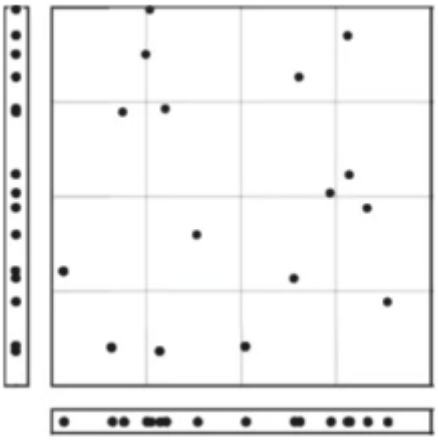
- **Color coding using eigenvalue/vectors**
  Measure of alignment.
  use simple colormap
  $$\begin{aligned} R &= |\mathbf{e}_1 \cdot \mathbf{x}|, \\ G &= |\mathbf{e}_1 \cdot \mathbf{y}|, \\ B &= |\mathbf{e}_1 \cdot \mathbf{z}|. \end{aligned}$$
  , only magnitude matters.
- **How are tensor glyphs associated with eigenvalues/vectors**
  Use hedgehog or glyphs to indicate major eigenvector. Only use glyphs where anisotropy is large enough.



Superquadrics: use most often.

8. Sampling (know the algorithms)

- **Jittered**

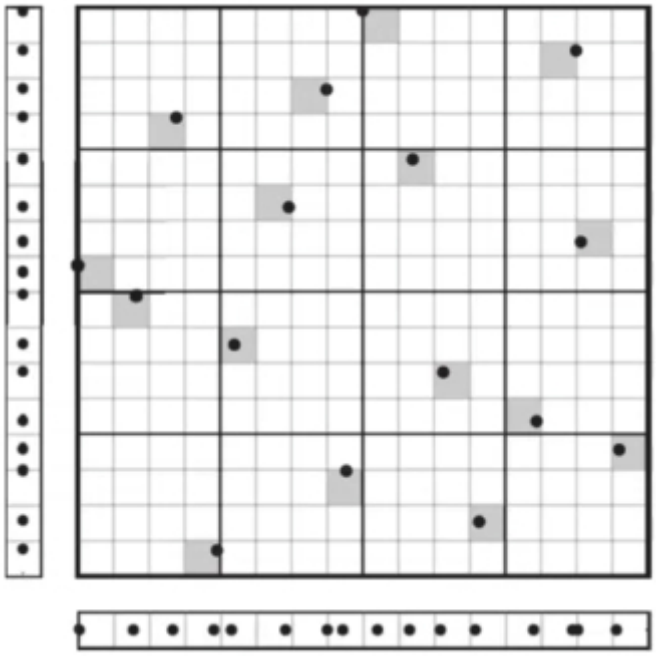


create nxn cells and randomly generate one in each cell

x-y projections could be still poorly distributed.

Samples: nxn

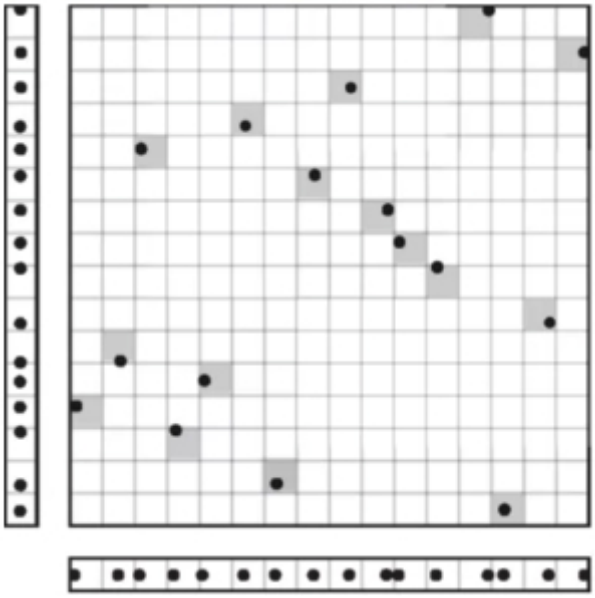
- **Multi-jittered**



2 grids. Coarse grid keeps jittered condition. Find grid keeps rook condition.

Sample: n

- **N-Rooks**



Use nxn grid and one sample exactly in each row and column

2D distribution is worse than jittered.

Samples: n

- **Hammersley**

  Use quasi-random sequnene.

$$\Phi_2(i) = \sum_{j=0}^{n} a_j(i) 2^{-j-1} = a_0(i)\frac{1}{2} + a_1(i)\frac{1}{4} + a_2(i)\frac{1}{8} ....$$

Radical inverse function of integer i to base 2

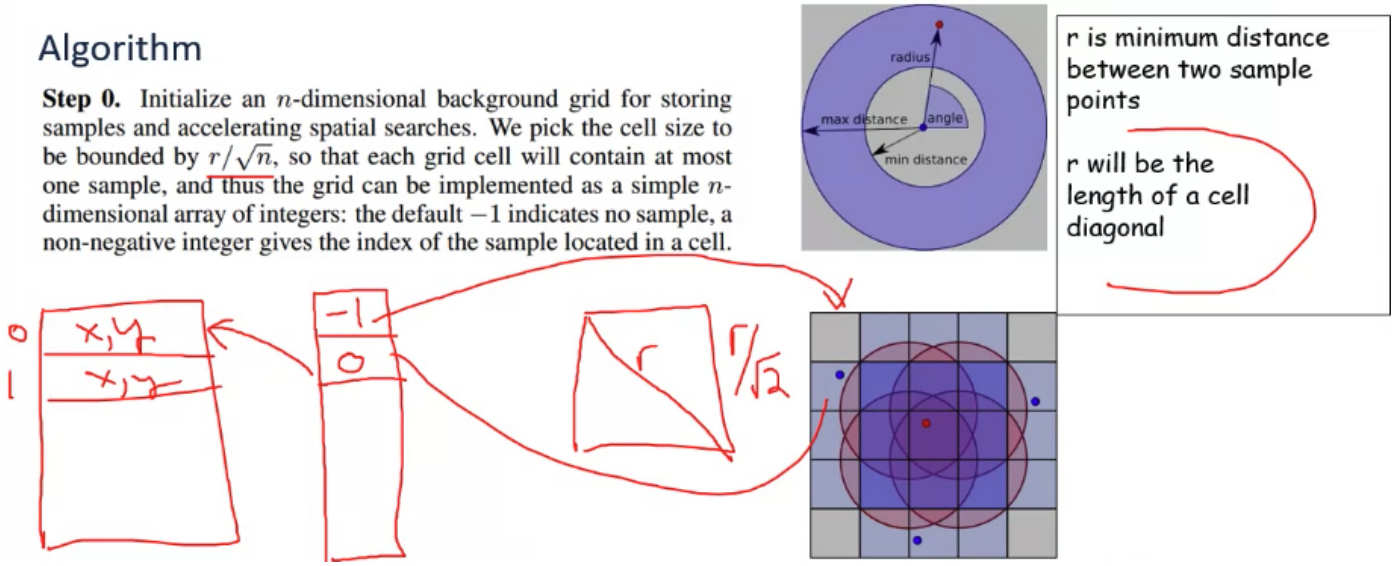
- reflect binary digits of i across decimal point
- evaluate this new number now in [0,1)

i		Reflection around the Decimal Point		$\Phi_2(i)$ (base 2)
1	1_2	$.1_2$	1/2	0.5
2	10_2	$.01_2$	1/4	0.25
3	11_2	$.11_2$	1/2 + 1/4	0.75
4	100_2	$.001_2$	1/8	0.125
5	101_2	$.101_2$	1/2 + 1/8	0.635
6	110_2	$.011_2$	1/4 + 1/8	0.325
7	111_2	$.111_2$	1/2 + 1/4 + 1/8	0.875
8	1000_2	$.0001_2$	1/16	0.0625

Set n samples, $p_i = (x_i, y_i) = (i/n, \Phi_2(i))$

But 1-D projection are regular; for given n, only one sequence; need to know ahead of time.

- **Halton sequence**
  no need to known the number of samples.
  Take radical inverse based of some prime of i along each of the dimensions: $p_i = (\Phi_2(i), \Phi_3(i), \Phi_5(i), ...)$
- **Poisson disk sampling**
  Not actually a low-discrepancy sequence. Assure **min distance** between points. Pick cell size $r/\sqrt{n}$ so each grid cell only has one sample at most.



Then select the initial sample and insert 0 into active list.

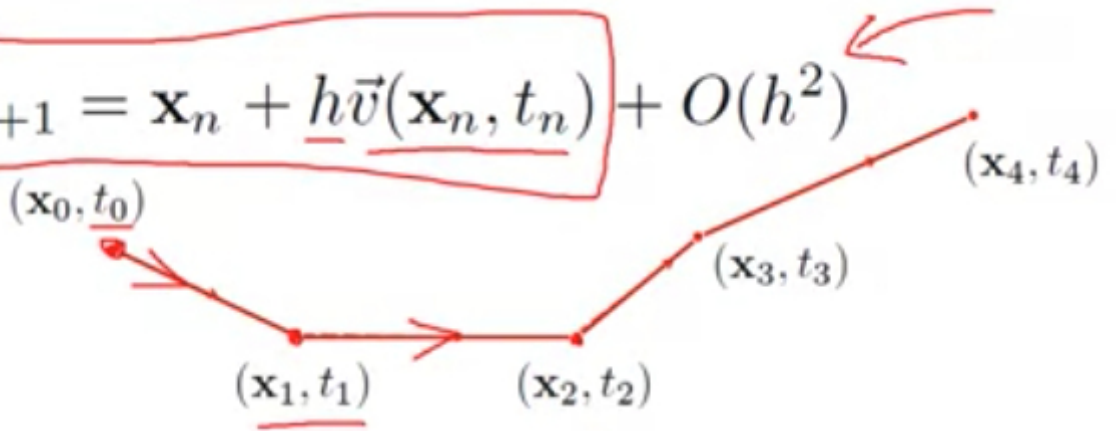
1. while active list is not empty, choose a random index i.
2. generate up to k points chosen uniformly from the region between r and 2r
3. For each point, check if it is within distance r of existing samples
4. If far enough, emit it as next sample and add to active list.
5. If no such a point after k attempts, remove from active list.

- **What does it mean to be well-distributed?**
  not structured, have some randomness.
  uniform distribution, avoud gaps.
  projection into 1D are aslo uniform.
  non-trivial minimum distances between sample points.
- **What does low discrepancy mean?**
  - We often want a low discrepancy sequence
    - e.g. Monte Carlo methods
  - Imagine points in some space S = $[0,1]^n$
  - Suppose we sample using K points…
  - We can evaluate the quality by
    - take portion V of S
    - volume V/volume S should equal (number points in V)/(number points in S)
      - …but it generally won't
    - the difference is the *discrepancy*

9. Numerical Methods

- **Euler's Method**
  - Memorize the formula and know the algorithm
    A vector field: a field of velocities.
    Solve ODEs to get solutions: $x(t) = x_0 + \int v(x(u))du.$
    No analytical solution, use Euler's method to get numerical solution.

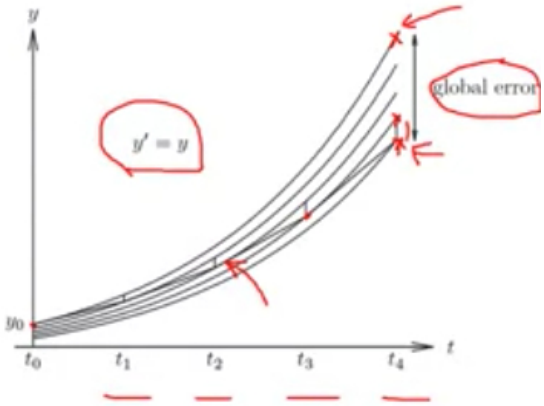
$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\vec{v}(\mathbf{x}_n, t_n) + O(h^2)$$



- What is the error?
  Roudning error: finite precision of floating point arithmetic
  Truncations error: approximate an infinite process with a finite number of steps.
  They are not independent and truncation error usually dominates.

At kth step

- Global error is the cumulative overall error
  $e_k = y_k - y(t_k)$
- Local error is the error in one step
  $\ell_k = y_k - u_{k-1}(t_k)$



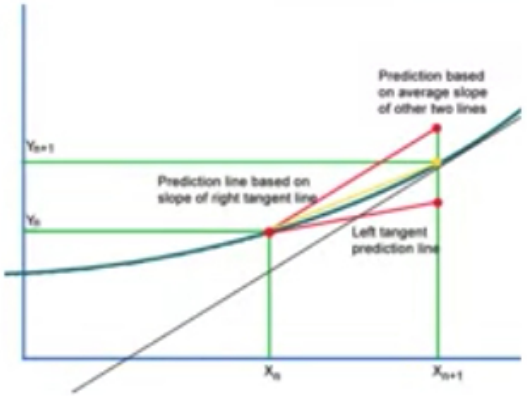
Global error: $O(h)$; Local error: $O(h^2)$. Use higher-order method.

- **RK-4**
  - Understand what the symbols mean…
    Higher accuracy. Euler's method is the first order RK method.

Heun's 2nd Order Method

Second order RK method

1. $\vec{k}_1 = \vec{v}(x_n, t_n)$
2. $\vec{k}_2 = \vec{v}(x_n + h_n k_1, t_n + h_n)$
3. $x_{n+1} = x_n + \frac{h_n}{2}(\vec{k}_1 + \vec{k}_2)$



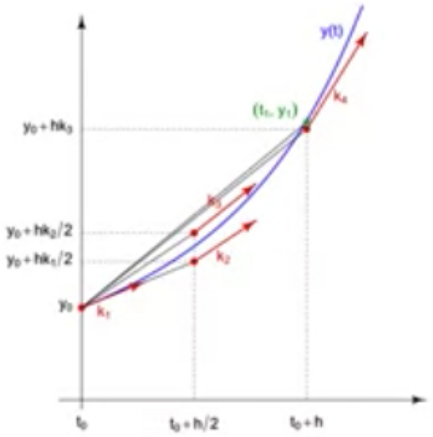
Local truncation error is $O(h^3)$

, h_n is step size; could vary h_n by step.

- Fourth order Runge-Kutta

$$\vec{k}_1 = h\vec{v}(\mathbf{x}_n, t_n)$$
$$\vec{k}_2 = h\vec{v}(\mathbf{x}_n + \frac{1}{2}\vec{k}_1, t_n + \frac{1}{2}h)$$
$$\vec{k}_3 = h\vec{v}(\mathbf{x}_n + \frac{1}{2}\vec{k}_2, t_n + \frac{1}{2}h)$$
$$\vec{k}_4 = h\vec{v}(\mathbf{x}_n + \vec{k}_3, t_n + h)$$

$$\vec{x}_{n+1} = \vec{x}_n + \frac{1}{6}\vec{k}_1 + \frac{1}{3}\vec{k}_2 + \frac{1}{3}\vec{k}_3 + \frac{1}{6}\vec{k}_4 + O(h^5)$$



- Ba able to apply the algorithm if given the formula
- What is the error?

Accuracy

- RK4 requires 4 evaluations of the derivative (velocity) per step
- RK2 requires 2 evaluations per step
- Euler requires 1 evaluation per step

For RK4 to be superior it would need to be more accurate than

- RK2 using ½ the stepsize
- Euler using ¼ the stepsize

This is usually the case...but can vary by problem

- **Central difference formula for a numerical derivative**
  derivative: the rate of change.

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

h is called the step-size

- smaller step-size will result in less error
- error is on the order of $O(h^2)$

Appropriate when function is known. Not appropriate for sampled or noisy data.
Better approach for sampled data: fit an approximation function first, then take derivative of the function.

10. Vector Field Visualization

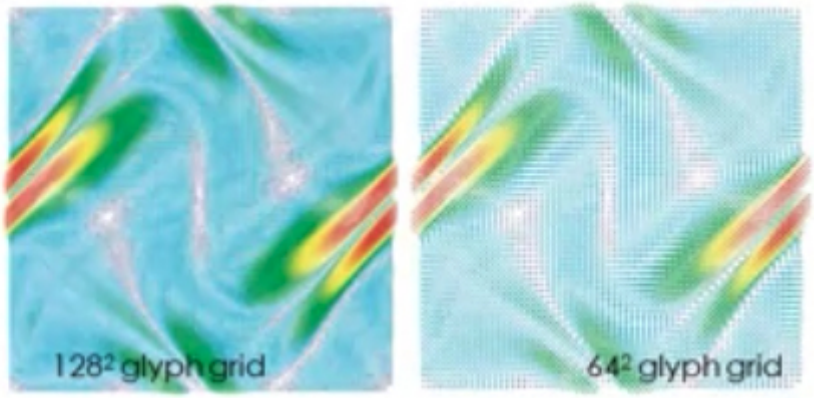
- **Glyphs**

Icons, or signs, for visualizing vector fields

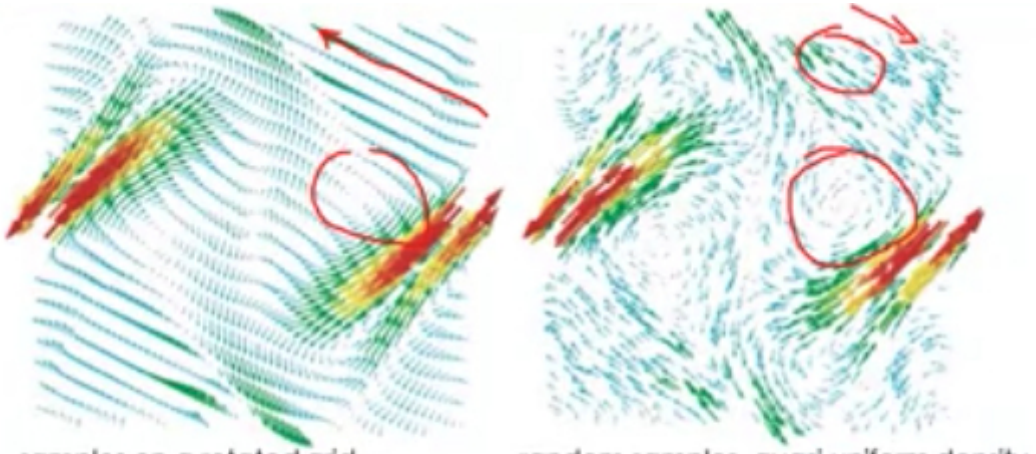
- placed by (sub)sampling the dataset domain
- attributes (scale, color, orientation) map vector data at sampl points

Simplest glyph: Line segment (hedgehog plots)

- for every sample point $x \in D$
  - draw line $(x, x + k\mathbf{v}(x))$
  - optionally color map $\|\mathbf{v}\|$ onto it



- more sample: more potential clutter
- fewer sample: higher clarity
- more line scaling: easy to see high speed region, more clutter;
- less line scaling: less clutter, hard to perceive direction.



How to choose sample points

- avoid uniform grids!
- random sampling: generally OK

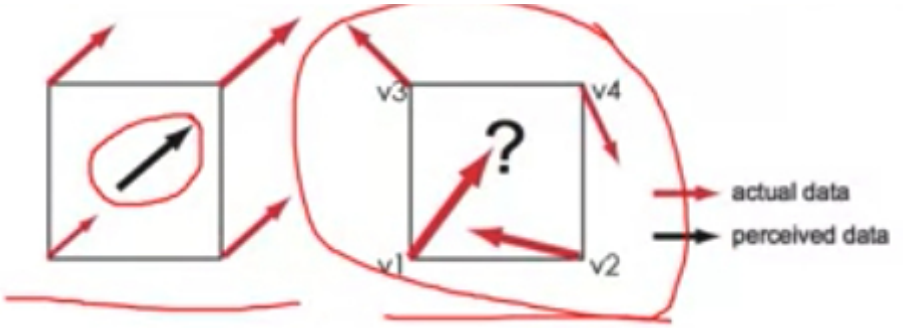
glyph with cones or arrows: show orientation better but take more space. Use shading (**not on lines**) to separate overlaping glyphs.

3D glyph: more data, occlusion, viewpoint selection.

1. Use alpha blending to reduce occlusion. Low-speed zones: highly transparent; high-speed zones: opaque and coherent.
2. Use Glyph on surfaces. Select certain isosurface with particular flow velocity.

Problems:

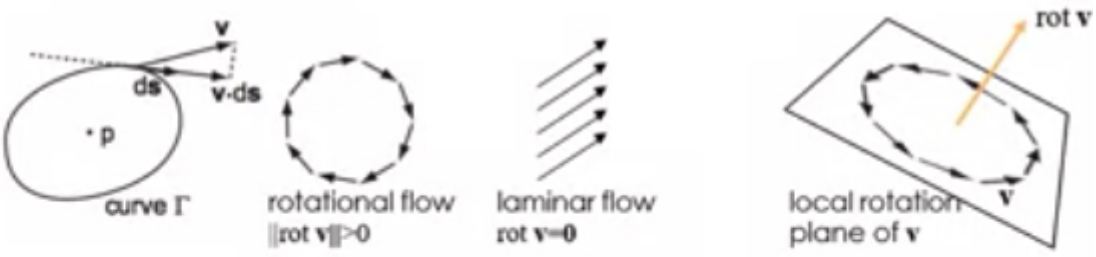
1. no interpolation in glyph space.



2. a glyph needs more space
3. human aren't good at visully interpolating arrows.
4. glyph plots are sparse, while scalar plots are dense.

- **Color coding**
- **Vorticity**
  - intuition: magnitude is how quickly the flow 'rotates' around each point?
  - given a field $\mathbf{v} : \mathbf{R}^3 \to \mathbf{R}^3$, rot $\mathbf{v} : \mathbf{R}^3 \to \mathbf{R}^3$ is

$$\text{rot } \mathbf{v} = \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z}, \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x}, \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right)$$



rot $\mathbf{v}$ is sometimes denoted as $\nabla \times v$

. rot v is othogonal to rotation plane. Magnitude tells how much spin.
2D vorticity: use $v(x, y) =< v_x, v_y, 0 >=< 0, 0, \partial v_y/\partial v_x - \partial v_x/\partial v_y >$

- **Divergence**

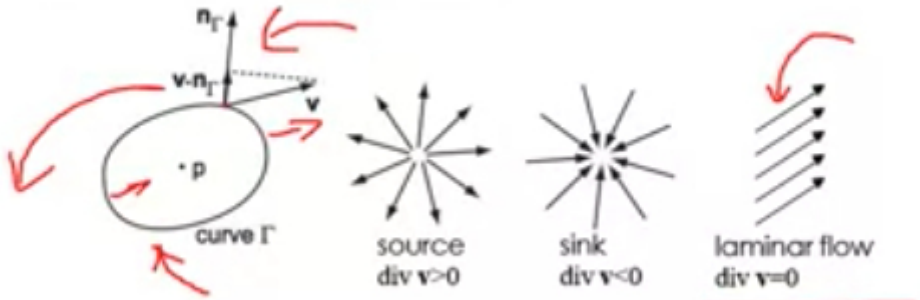
Divergence $\text{div } \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$ equivalent to $\text{div } \mathbf{v} = \lim_{\Gamma \to 0} \frac{1}{|\Gamma|} \int_\Gamma (\mathbf{v} \cdot \mathbf{n}_\Gamma) ds$

Example:

$$v(x,y,z) = \langle xy^2, xy^2, zy \rangle$$
$$v(1,2,3) = \langle 4,4,6 \rangle$$

$$\text{div } v(x,y,z) = y^2 + 2xy + y$$
$$\text{div } v(1,2,3) = 4 + 4 + 2 = 10$$

- intuition: degree to which the field converges or diverges at a point
- given a field $\mathbf{v} : \mathbf{R}^3 \to \mathbf{R}^3$, $\text{div } \mathbf{v} : \mathbf{R}^3 \to \mathbf{R}$ is



$\text{div } \mathbf{v}$ is sometimes denoted as $\nabla \cdot v$

Give impression where the flow enters and exits.

- **Stream-based visualization**
  - Streamline
    particle trajectory in steady (unchanging) vector field
  - Pathline
    trajectory of particle in an unsteady flow
  - Timeline
    connect particles released simultaneously at discrete time-steps
  - Streakline
    continuously inject particles at a point, connect consecutive particles
    - Timelines
    - Streaklines
      For unsteady flow
    - **Streamlines**
      For steady flow

Streamlines

- assume that $\mathbf{v}$ is not changing in time (steady-states)
- for each seed $p_o \in D$
  - the streamline S seeded at p_o is given by

$$S = \{p(\tau), \tau \in [0, T]\}, p(\tau) = \int_{t=0}^{\tau} \mathbf{v}(p)dt, \quad \text{where } p(0) = p_0$$

integrate p_o in vector field $\mathbf{v}$ for time T

Stream object show trajectory for longer time interval; vector glyphs is over a short time. 2D starts from few positions. 3D has too many streamlines and scene is cluttered. Use samplings to keep appropriate positions to start.

- numerically integrate

$$S = \{p(\tau), \tau \in [0, T]\}, p(\tau) = \int_{t=0}^{\tau} \mathbf{v}(p)dt, \quad \text{where } p(0) = p_0$$

- discretizing time yields

$$\int_{t=0}^{\tau} \mathbf{v}(p)dt = \sum_{i=0}^{\tau/\Delta t} \mathbf{v}(p_i)\Delta t \quad \text{where } p_i = p_{i-1} + \mathbf{v}_{i-1}\Delta t$$

(simple Euler integration)

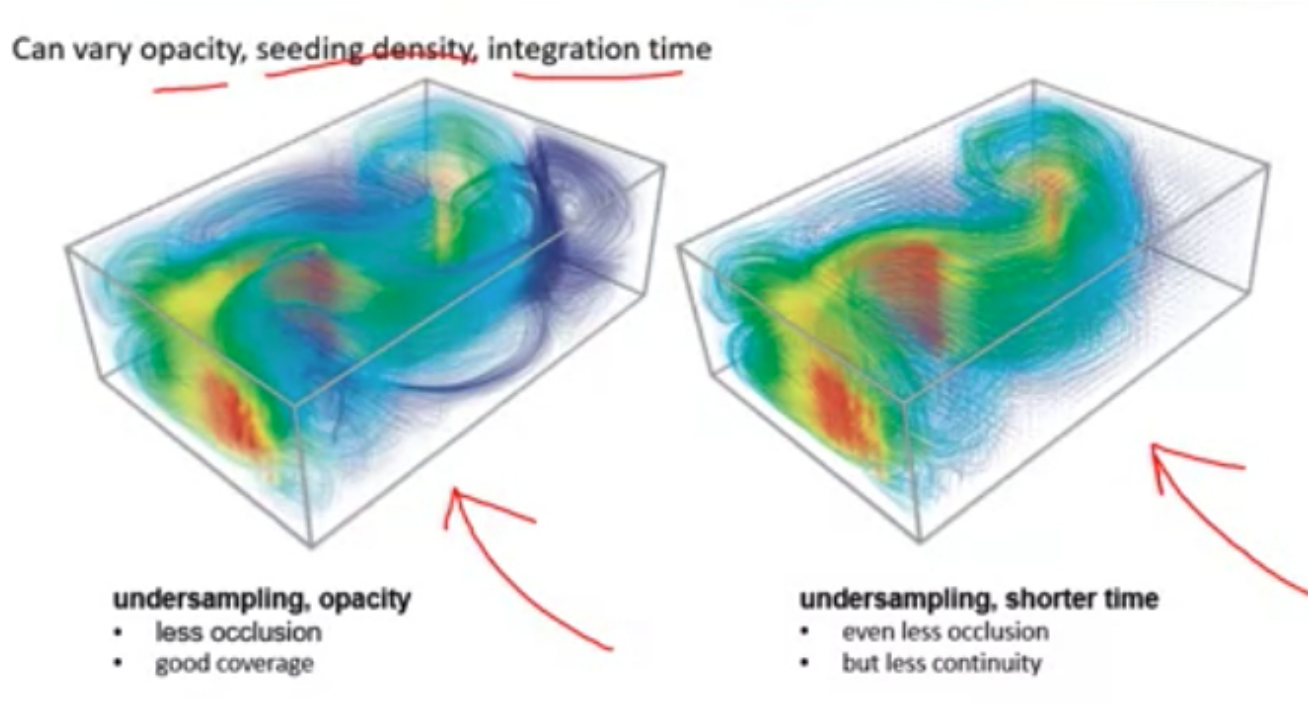
RK4 would be better

- Euler integration
  - we consider $\mathbf{v}$ constant between two sample points p_i and p_{i+1}
  - we compute $\mathbf{v}(p)$ by linear interpolation within the cell containing p
  - variant: use $\mathbf{v}(p)/\|\mathbf{v}(p)\|$ instead of $\mathbf{v}(p)$ in integral
  - S will be a polyline, $S = \{p_i\}$
- stop when $\tau = T$ or $\mathbf{v}(p) = 0$ or $p \notin D$
  - what does $\tau = T$ mean when we use $\mathbf{v}(p)/\|\mathbf{v}(p)\|$?

Use euler's method to compute location.

Stream tubes: hyperstreamlines to represent extra data with tube thickness.

Adjust by varying **opacity (occlusion), seeding density (coverage), integration time (continuity)**.

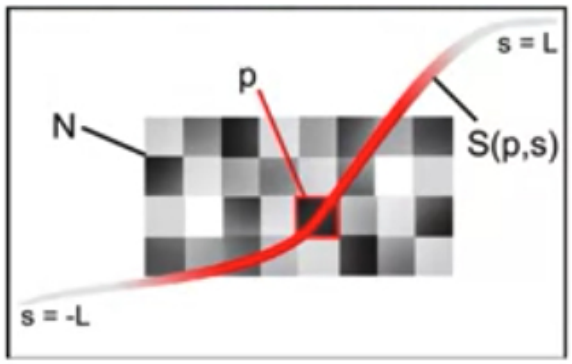


3D stream tube: stream tubes traced from inlet to outlet, must reduce number of seeds to reduce occlusion.

Stream ribbons: how vector field twists. Choose **pairs** of close seeds, trace streamlines and construct strip by connecting closest points.

- **LIC (Line Integral Convolution) principle**

  highly coherent along streamlines; highly contrasting across the streamlines.



LIC: Line Integral Convolution

$$T(p) = \frac{\int_{-L}^{L} N(S(p,s))k(s)ds}{\int_{-L}^{L} k(s)ds}$$

$$k(s) = e^{-s^2}$$

N : noise texture

$S(p,s)$: streamline of seed point P

$k(s)$: weighting or blurring function

L : width of blurring function

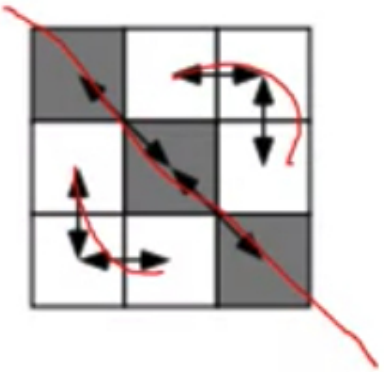
Take each pixel of the image, trace a streamline from upstream and downstream, blend all streamlines, pixel-wise

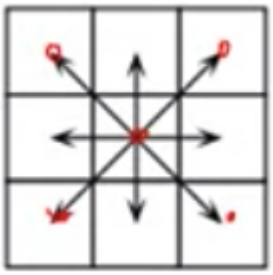
11. Classification and Segmentation

Solution: Different Rules for Different Classifications

- Usual solution:
  - Use 4-connectivity for the foreground
  - Use 8-connectivity for the background

  or
  - Use 8-connectivity for the foreground
  - Use 4-connectivity for the background





8-connectivity.

- **Graph cuts algorithm**

  Similarity matrix.

  Idea: break graph into segments by deleting links. Break links that have low cost (low similarity). Similar pixels should be in the same segment; dissimilar pixels should be in different segment.

  Not use all edges (not feasible). Use **neighboring pixels** (4 or 8) with direct links.

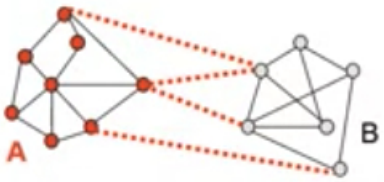
$$affinity(p_i, p_j) = e^{-\frac{\|f(p_i)-f(p_j)\|^2}{2\sigma^2}}$$

- p_i, p_j are pixels
- $f(p_i)$ metric for a pixel we can use in a distance function
- $2\sigma^2$ is "normalization" factor...a tweakable parameter

f could be location distance, color, intensity, texture.

Definition: Cut

- set of links whose removal makes a graph disconnected
- cost of a cut: $cut(A, B) = \sum_{p \in A, q \in B} c_{p,q}$



The sum of edge weights. Find minimum cut.

Source and Sink Edge Weights

Connect source to all pixels

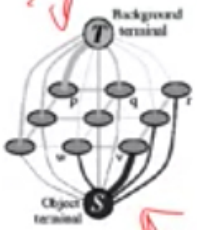
Foreground scribble pixels to foreground(src) vertex = infinity (max float)

Foreground scribble pixels to background(sink) vertex = 0

Connect the sink to all pixels

Background scribble pixels to background vertex(sink) = infinity (max float)

Background scribble pixels to foreground vertex(src) = 0

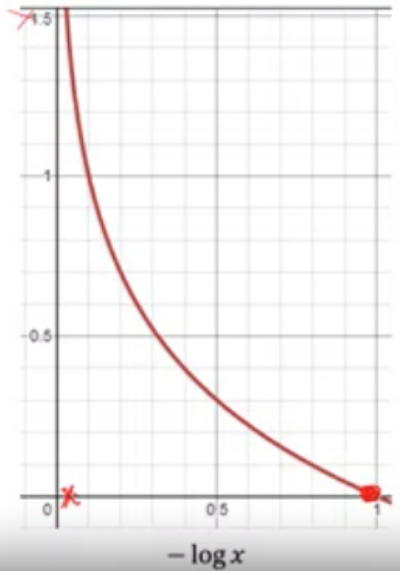


Edge Weights

Source and Sink to Non-Scribble Vertices p

Weight to foreground $affinity_{foreground}(p) = -\lambda \log P_B(p)$

Weight to background $affinity_{background}(p) = -\lambda \log P_F(p)$



Connect neighboring image vertices with edges (4 or 8 connected)

Edge weights:

$$affinity(p_i, p_j) = e^{-\frac{\|f(p_i)-f(p_j)\|^2}{2\sigma^2}}$$

$f(p_i)$ is intensity in original paper

Same intensity → $affinity(p_i, p_j) = 1$

Different intensity → $affinity(p_i, p_j) \approx 0$

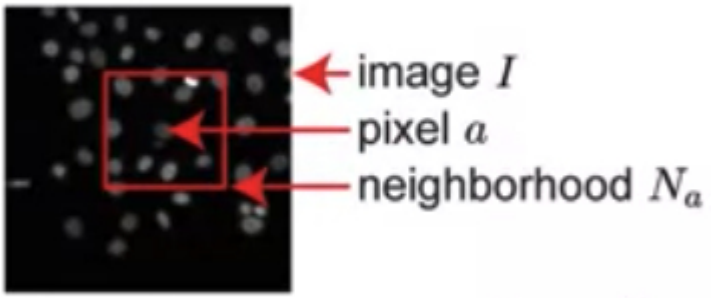
- **Thresholding**

  Intensity of a pixel: $i(x) = (r + g + b)/3$. Good to segment single object.

  Problem: same object has different intensity at different locations in an image.

  Solution: local thresholding.

  - Tessellate the image into rectangular blocks
    - Use different threshold parameter(s) for each block
  - Use a moving window



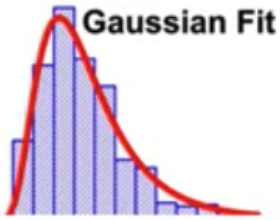
Global threshold: $\tau = f(\{i \in I\})$

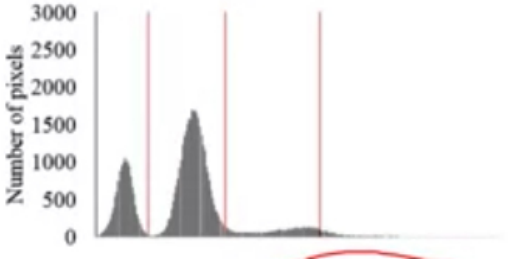
Local threshold: $\tau_a = f(\{i \in N_a\})$

Computing threshold:

1. expect contrast: $T = i_{avg} + \epsilon$. Use histogram analysis.

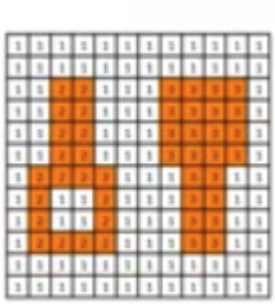
- Idea: Choose a threshold between the peaks
- Histograms often require pre-processing
- How to find the peaks?
  - Try fitting Gaussians to the histogram
  - Use their intersection(s) as the threshold(s)
  - How many Gaussians to try to fit?
    - Requires prior knowledge…or maybe Machine Learning





. Filter outliers in pre-processing.

Thresholding + connected component analysis for multiple object segmentation.



Thresholding
+
Connected
Component
Analysis

Recursive region growing to do connected component analysis.

Algorithm

Input: A threshold segmentation with object pixels as black in f

1. Search for an unlabeled black pixel; that is, $L(x, y) = 0$
   If you find one, choose a new label number for this region, call it N
   If all pixels have been labeled, stop
2. $L(x, y) \leftarrow N$
3. Push unlabeled neighboring object pixels onto the stack
   - If $f(x\text{-}1, y)$ is black push $(x\text{-}1, y)$ onto the stack.
   - If $f(x\text{+}1, y)$ is black push $(x\text{+}1, y)$ onto the stack.
   - If $f(x, y\text{-}1)$ is black push $(x, y\text{-}1)$ onto the stack.
   - If $f(x, y\text{+}1)$ is black push $(x, y\text{+}1)$ onto the stack.
4. If the stack is empty, go to 1
5. Else choose a new (x, y) by popping the stack and go to 2.

- **Otsu's method**

The algorithm searches for the threshold that minimizes

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

the intra-class variance, defined as a weighted sum of variances of the two classes

Weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t ,and σ_0^2 and σ_1^2 are variances of these two classes.

$Var(X) = \sum p_i(x_i - \mu)^2$

It's equivalent to maximizing the gap:

For 2 classes, minimizing the intra-class variance is equivalent to maximizing inter-class variance

..search for best gap…find t maxing:

$$\omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \qquad \mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \qquad \mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

1. Compute histogram and probabilities of each intensity level
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1, \ldots$ maximum intensity
    1. Update ω_i and μ_i
    2. Compute $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum $\sigma_b^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \qquad \mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \qquad \mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

For multipke object segementing, use Expectation Maximization (EM) to fit GMM.

- **Understand what a transfer function is and its purpose**
  Transfer function: map the information from every point along the ray to a color and opacity. A TF defines (1) which parts of the data are essential to depict and (2) how to depict these, often small, portions of the volumetric data.
  6 categories: 1D data-based, gradient 2D, curvature-based, size-based, texture-based, and distance-based.
  Traditional attributes of volum data: scalar value (density), gradient magnitude, and object/label ID (in the case of segmented data).
  Different segmented objects can have different TFs, different rendering modes (such as DVR or MIP), and different compositing modes. The latter capability enables two-level volume rendering, which comprises one local compositing mode per object, and a second global compositing level that combines the contributions of different objects.

Formulas

Memorize the following

- Euler's method
- Halton sequence
- Hammersley sequence
- Central difference formula

Given the following formulas, understand what the symbols mean:

- Betweeness centrality
- Quadric Error metric
  - Memorize how to compute optimal point
  - Memorize how to add quadrics together
- Tensor anisotropy measures
- RK-4
- LIC
- Vorticity
- Divergence
- Gradient