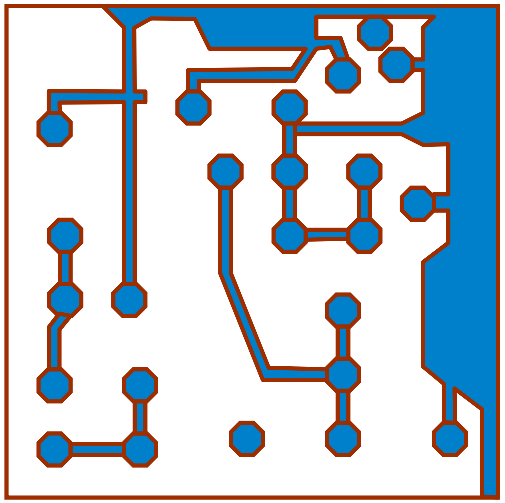
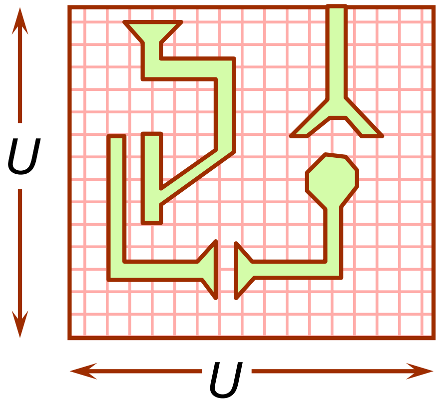
Geometric Data Structures for Computer Graphics

G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

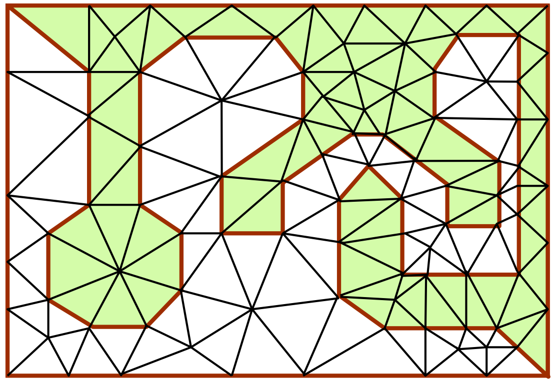
Meshing

- Wichtiger Preprocessing-Schritt in vielen Anwendungen
  - "Domain discretization" =
  - Komplexes Gebiet (2D oder 3D) wird in einfache Gebiete zerlegt (Dreiecke, Tetraeder)
- Anwendungen: FEM, CFD, VLSI = Simulation = Lösen von PDEs
  - PDEs lassen sich über regelmäßigem Gitter diskretisieren (über beliebige Gebiete nicht)

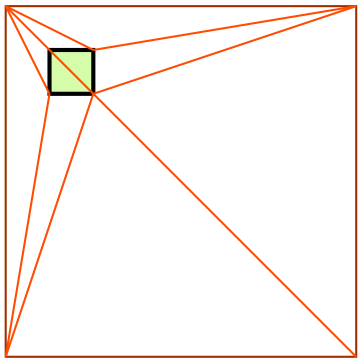




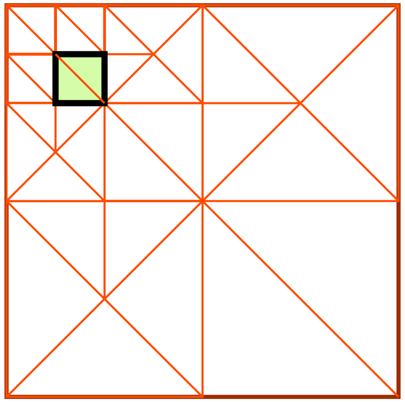
Uniform mesh, i.e. too many mesh elements



Non-uniform, conforming mesh that respect the input; well-shaped, too: bounded aspect ratio (e.g., angles ∈ [45°, 90°].
But needs so-called "Steiner points" (additional pts) ⟶ where/how to place them?

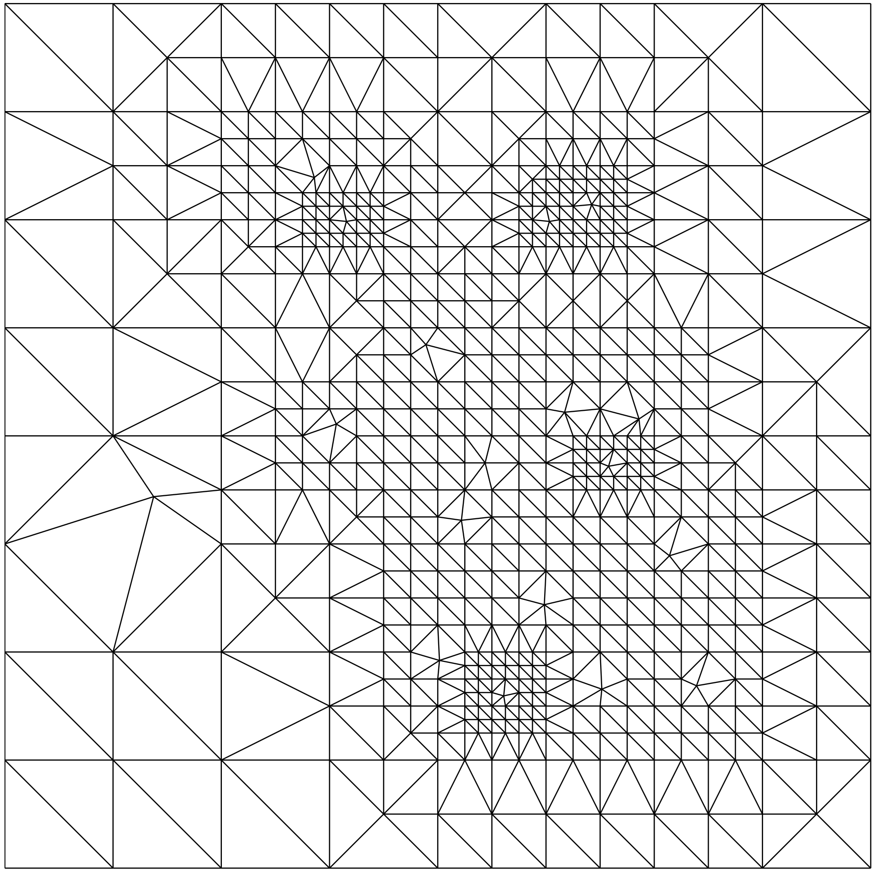


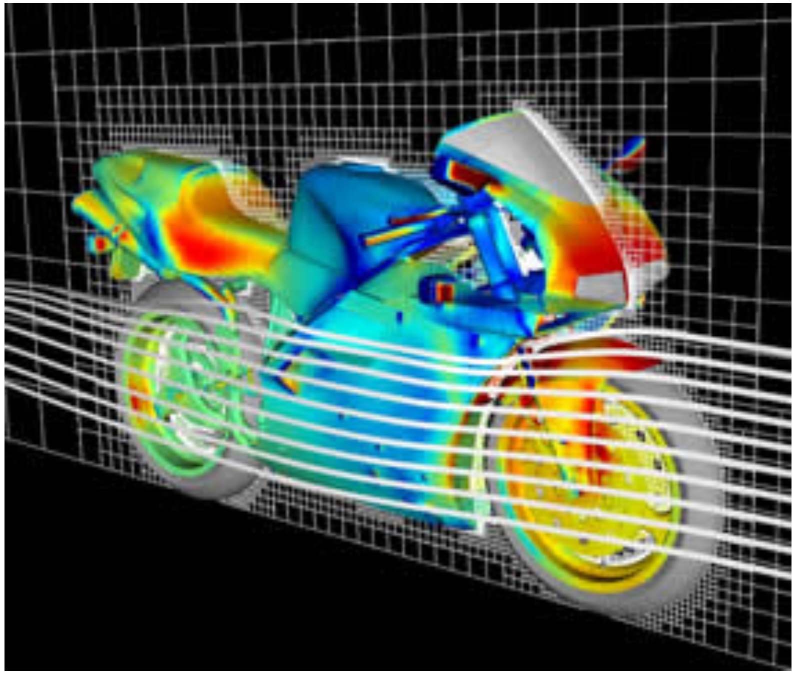
Non-uniform, conforming mesh that respect the input.
But acute triangles.

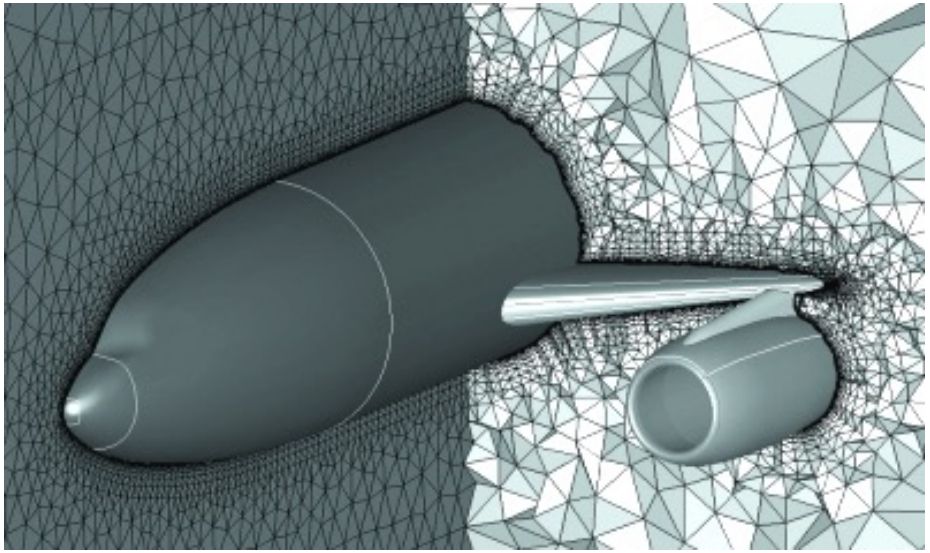


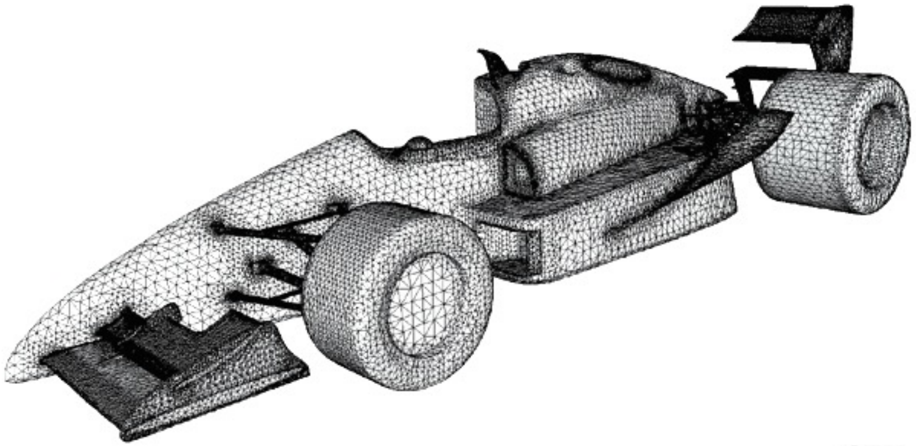
Mesh with all desired properties, based on quadtree.

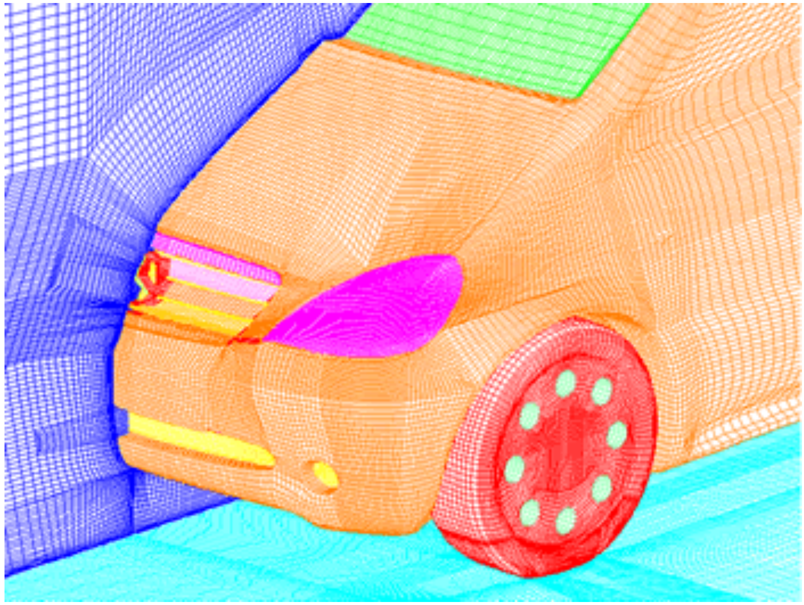
Example Result of Our Meshing Algorithm

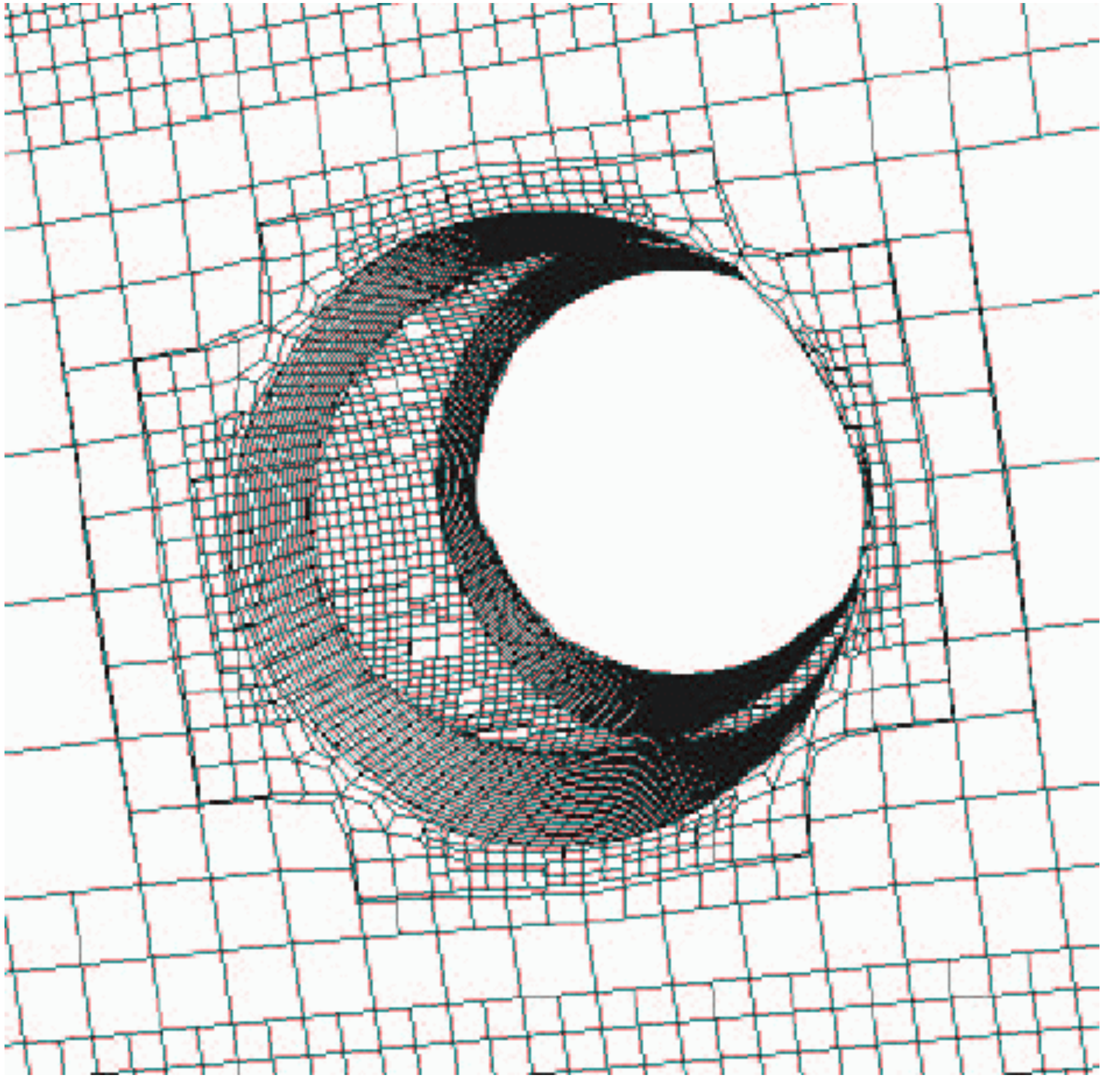




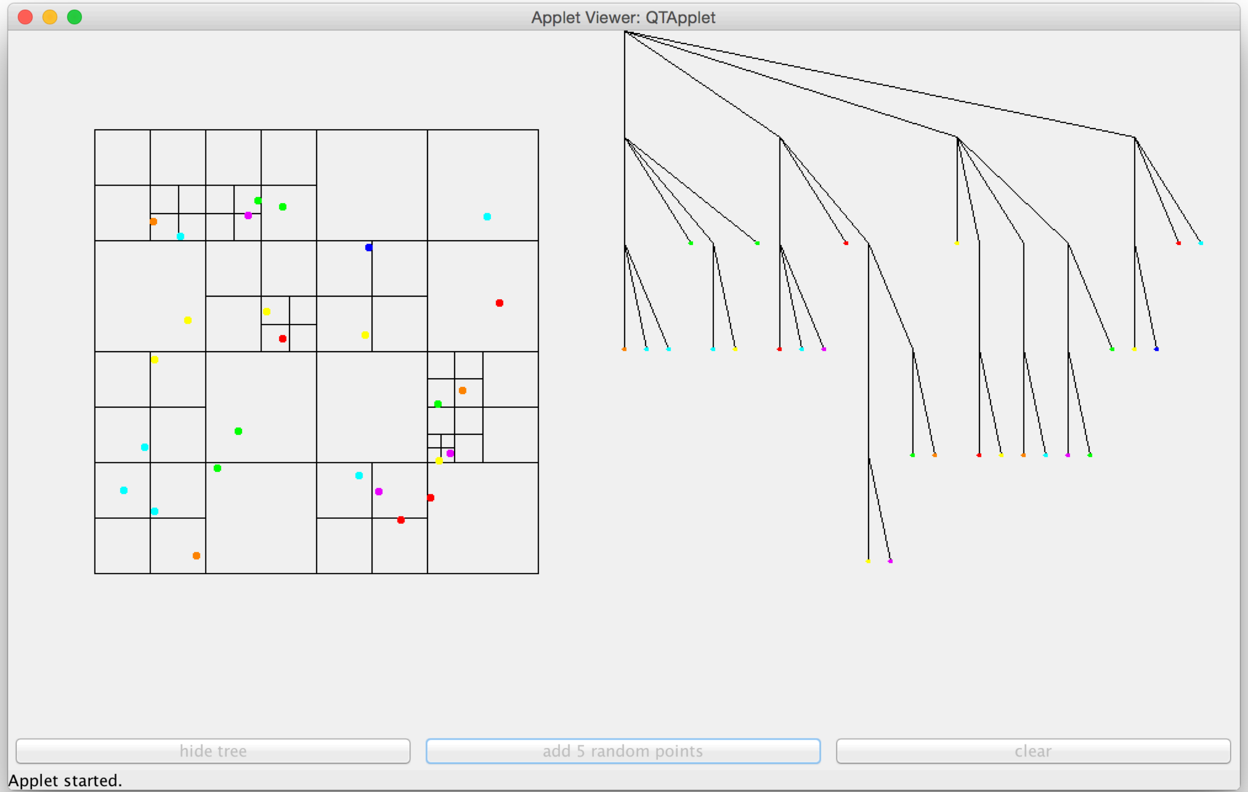






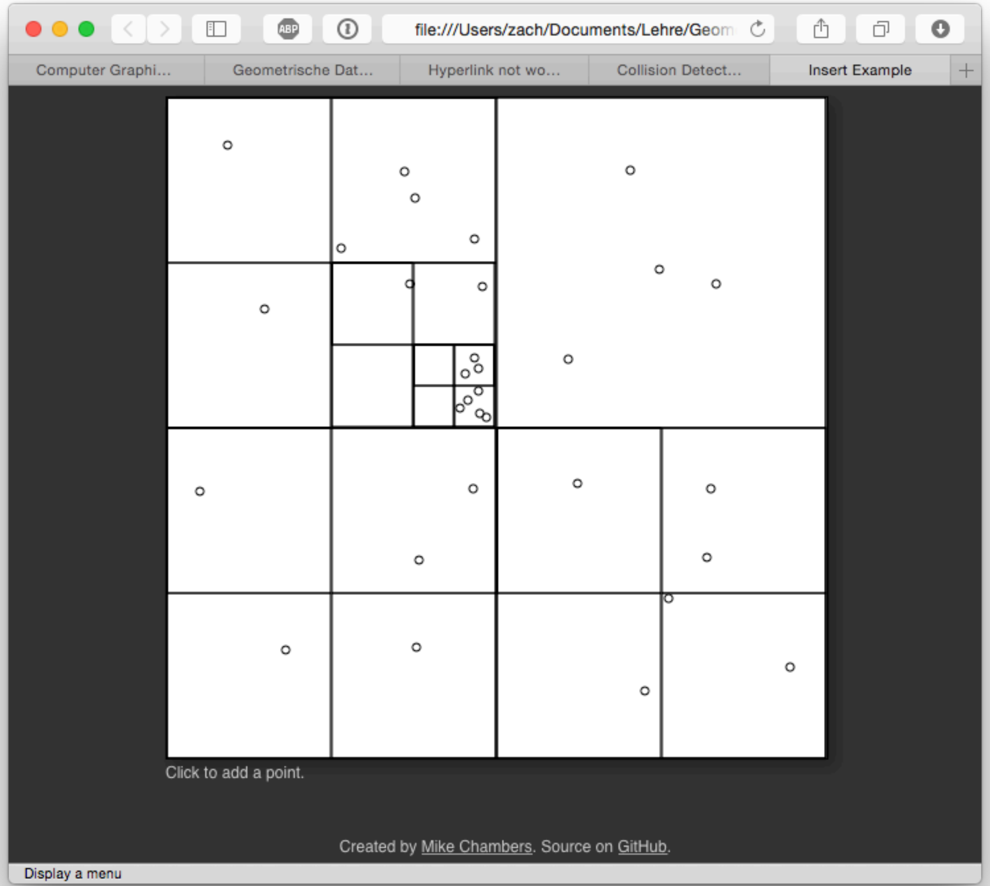


Point Quadtree Demo



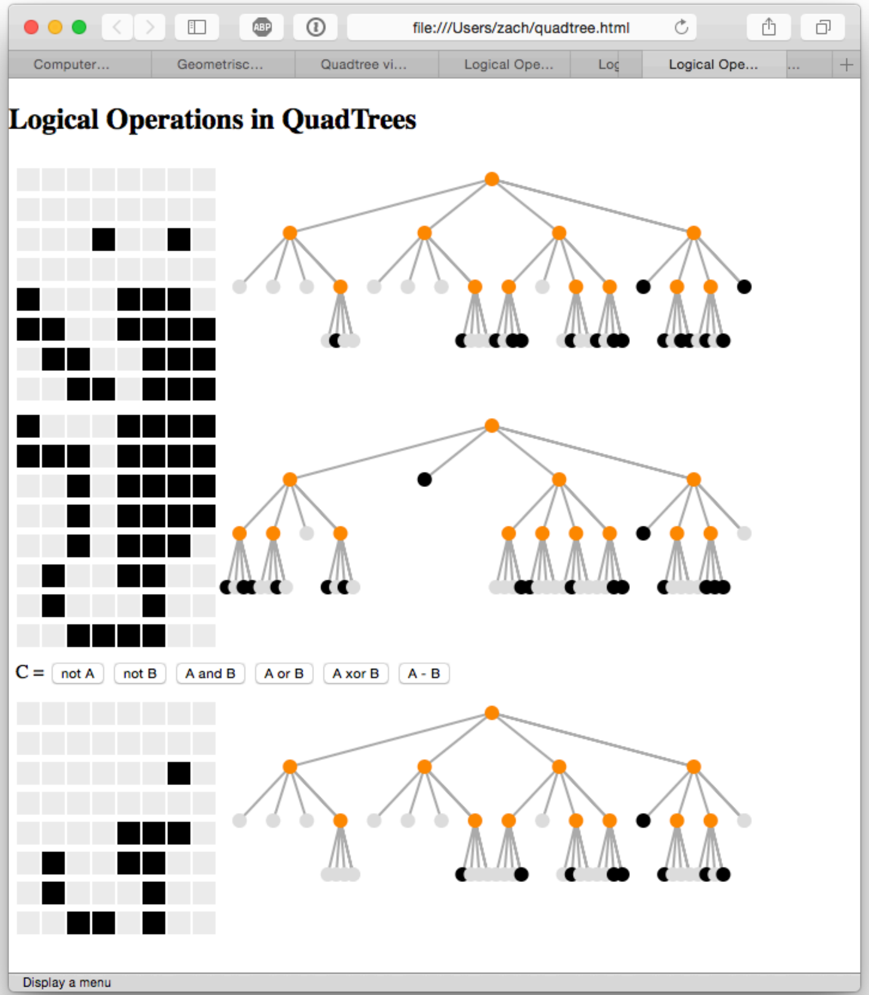
http://www.cs.utah.edu/~croberts/courses/cs7962/project/index.html

Quadtree Demo

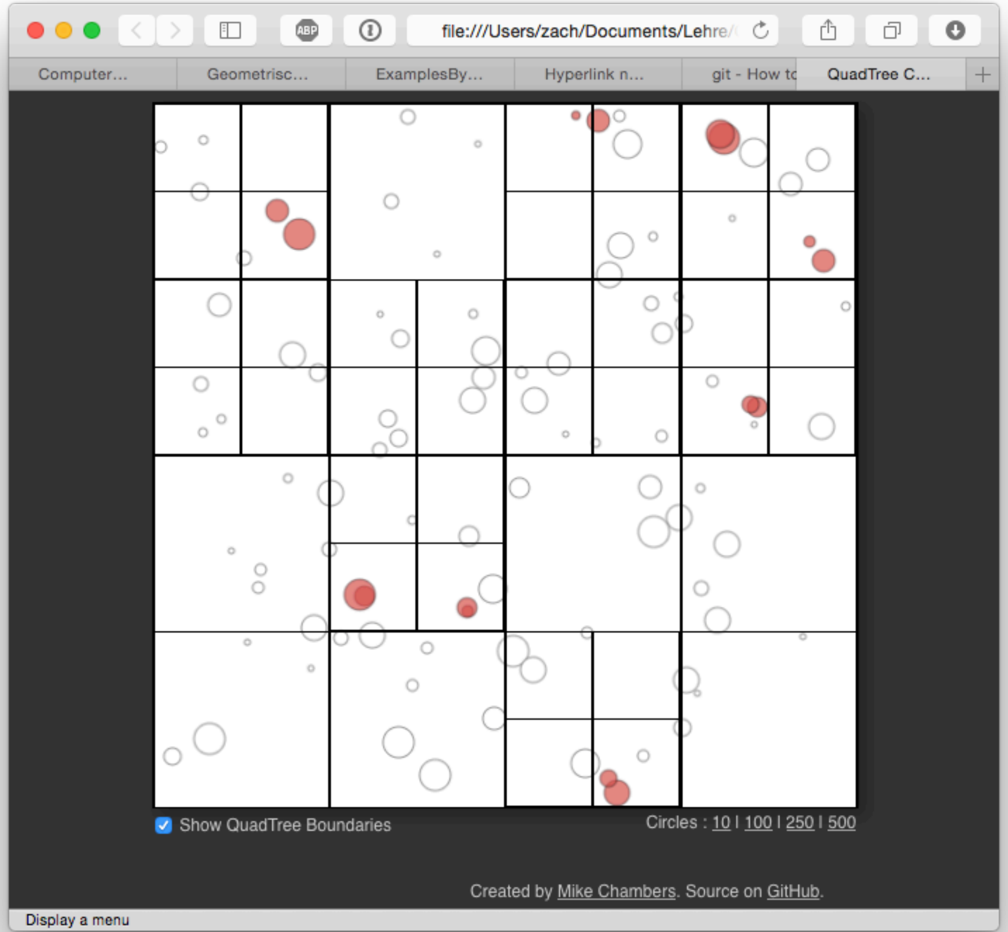


Recursion criterion here:
more than 4 points in a node

Logic Operations with Quadtrees

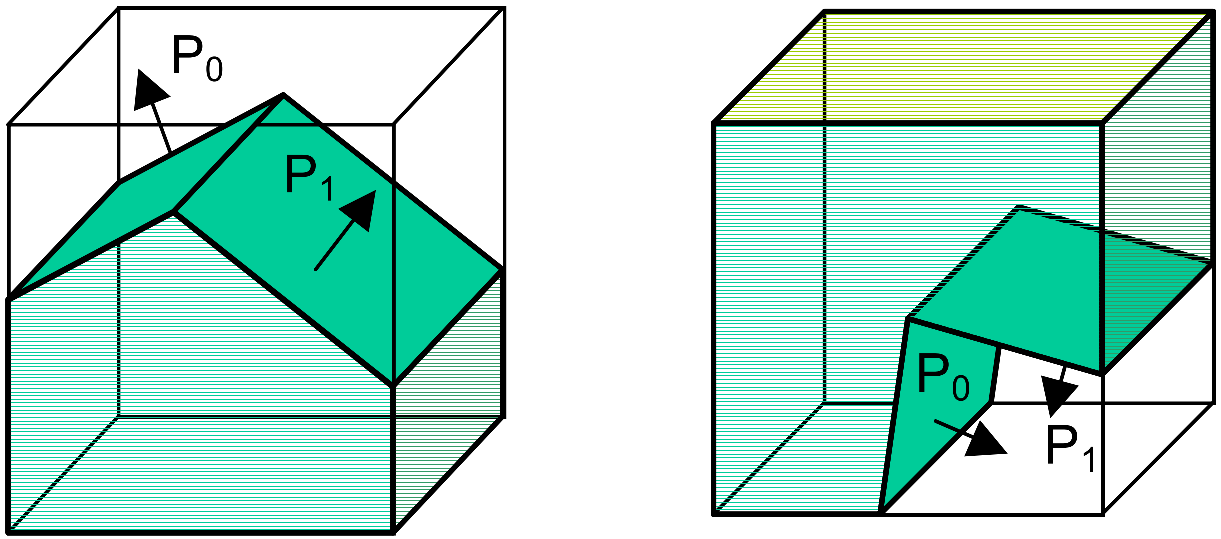


http://blog.ivank.net/quadtree-visualization.html



http://www.mikechambers.com/blog/2011/03/21/javascript-quadtree-implementation/

Exact Octrees (a.k.a. SP-Octrees)

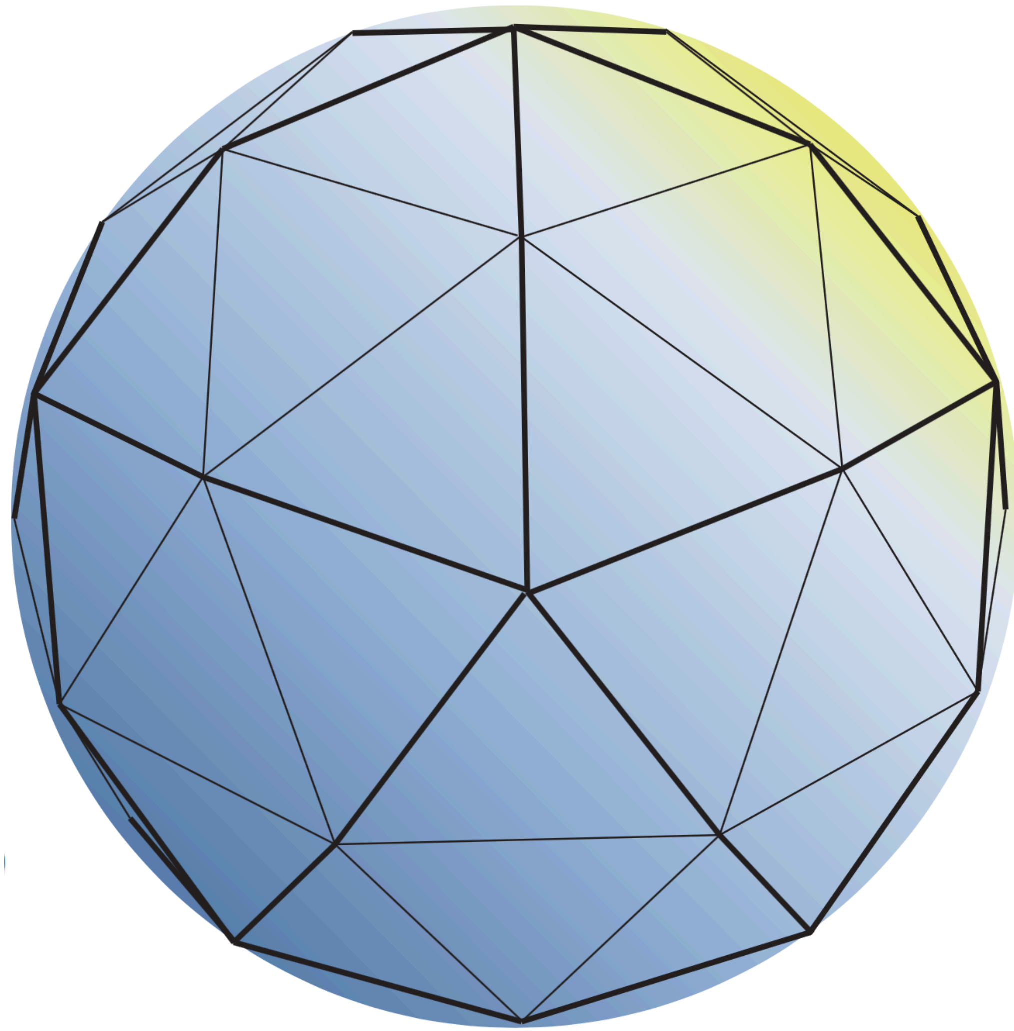


Boundary leaf nodes

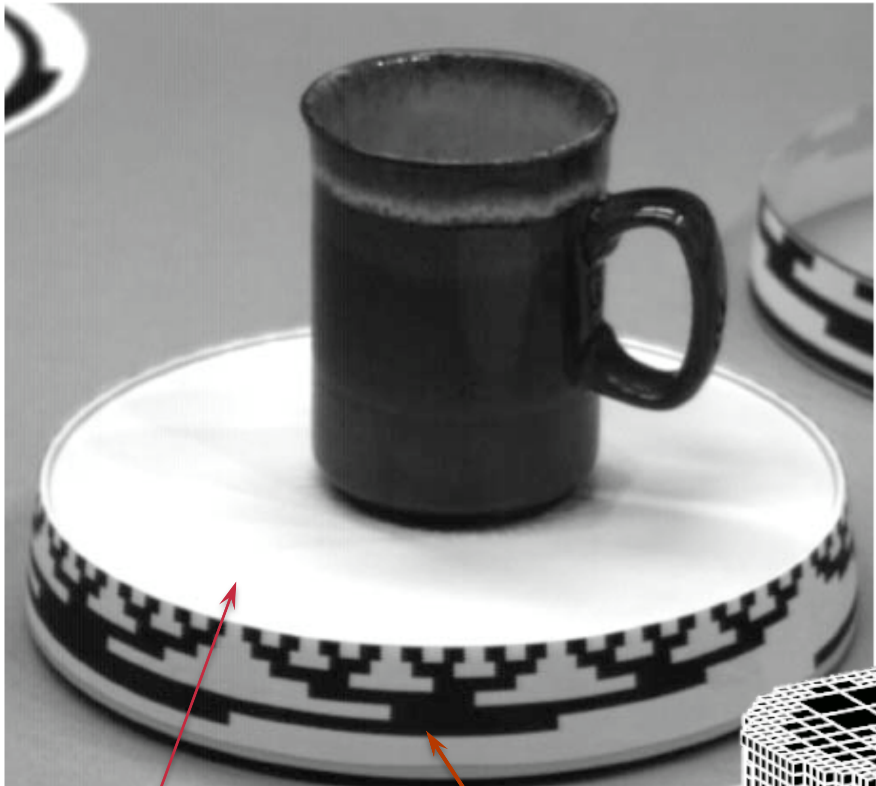
Other leaf nodes are black or white

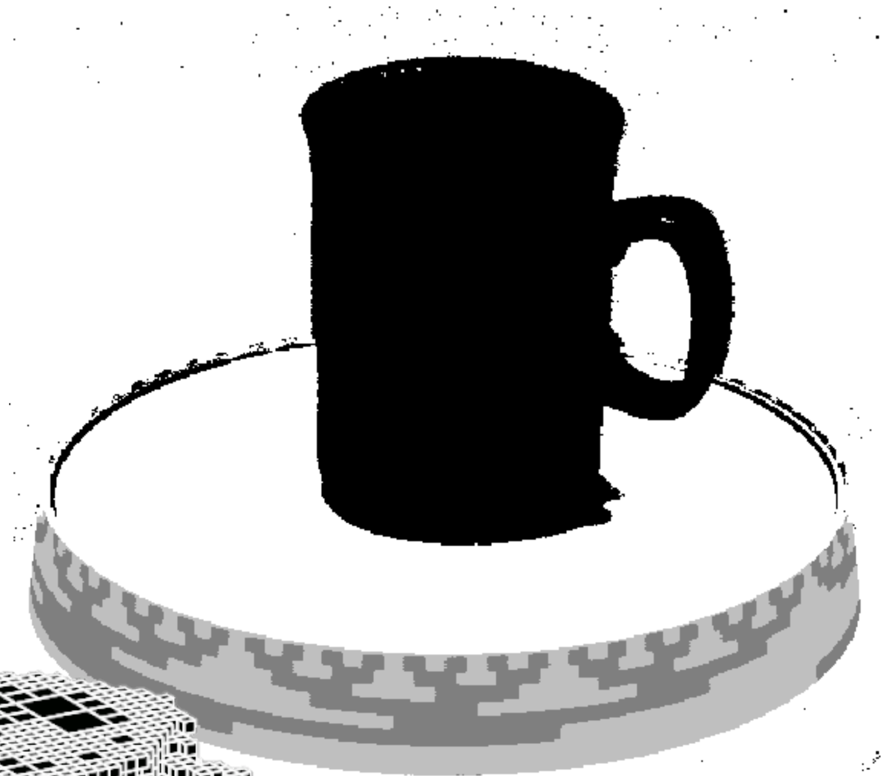
Geodesic Dome

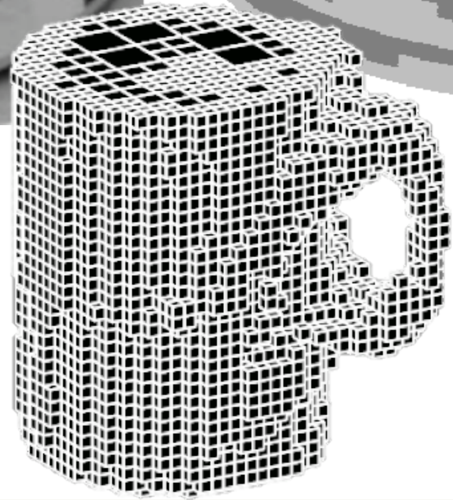
- Start with Icosahedron
- Subdivide each triangle by k^2 smaller triangles (recursively)
  - $\longrightarrow$ quadtree in each base triangle
- Navigation (finding neighbors of a node) in such an ensemble of quadtrees is a bit more complex



Octree Models from Images







Drehteller

Gray Code
(zur Erkennung der
Orientierung des
Drehtellers)

Example Models

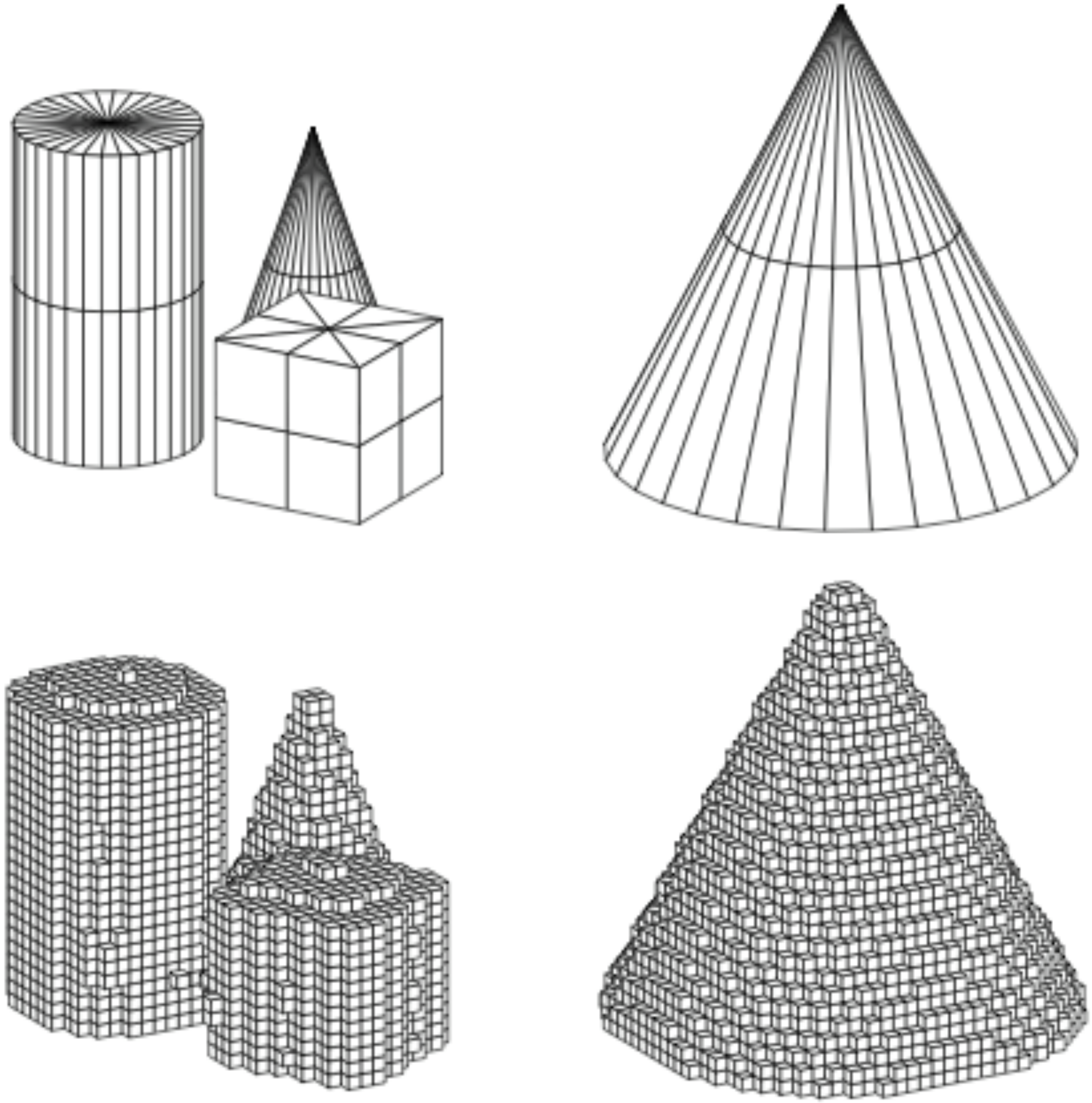
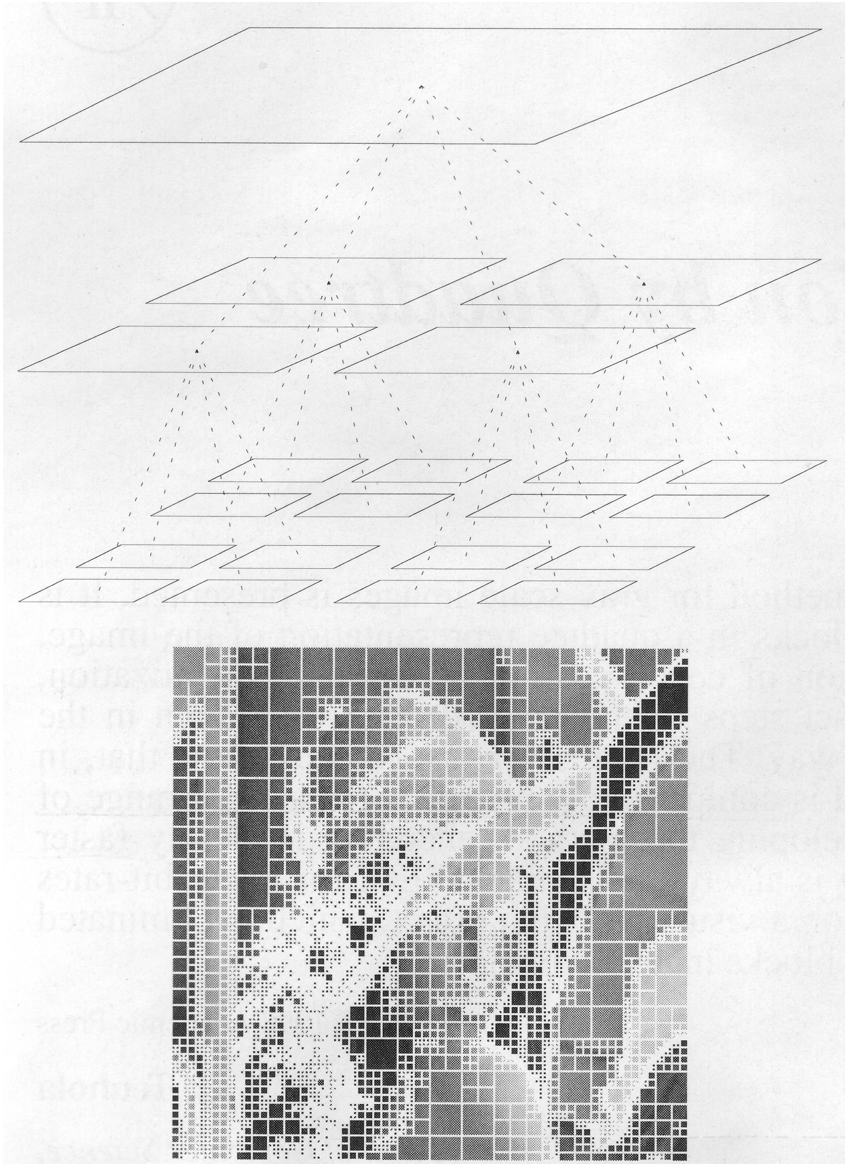


Image Compression using Quadtrees

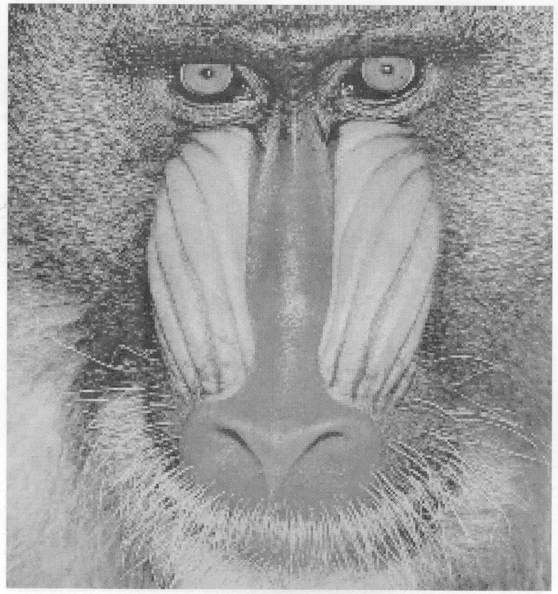


Die beiden Test-Bilder schlechthin

Resultate

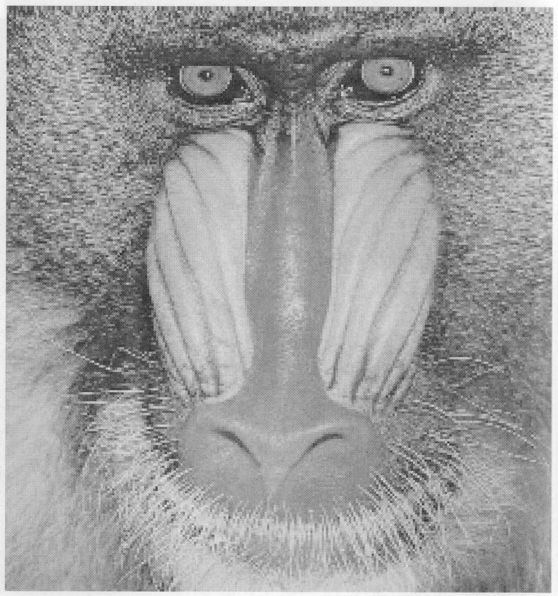
QP: 1.03 bits per pixel

QP: 1.95 bits per pixel

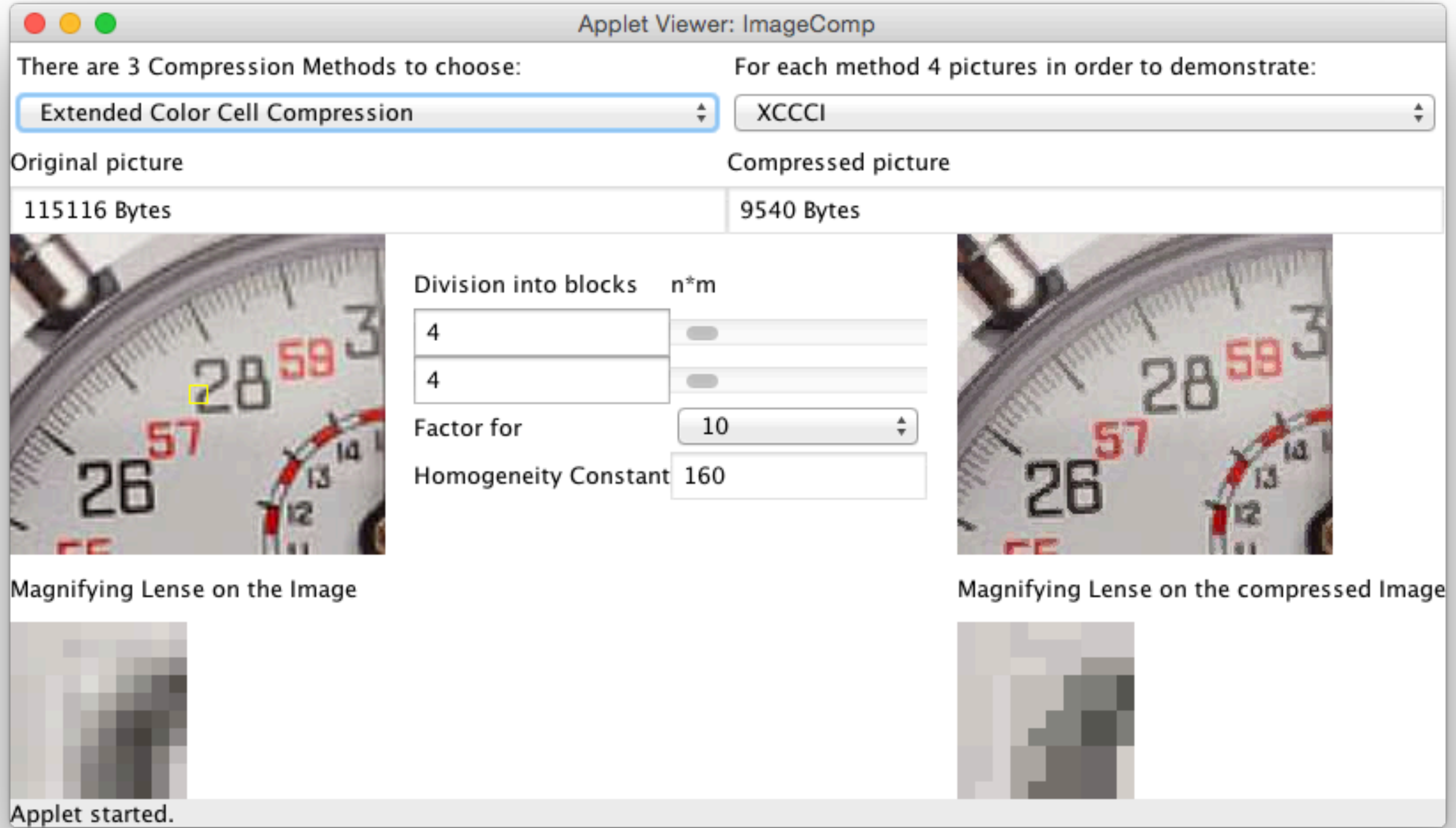


JPEG: 1.00 bits per pixel

JPEG: 1.99 bits per pixel



Demo for BTC and CCC Compression



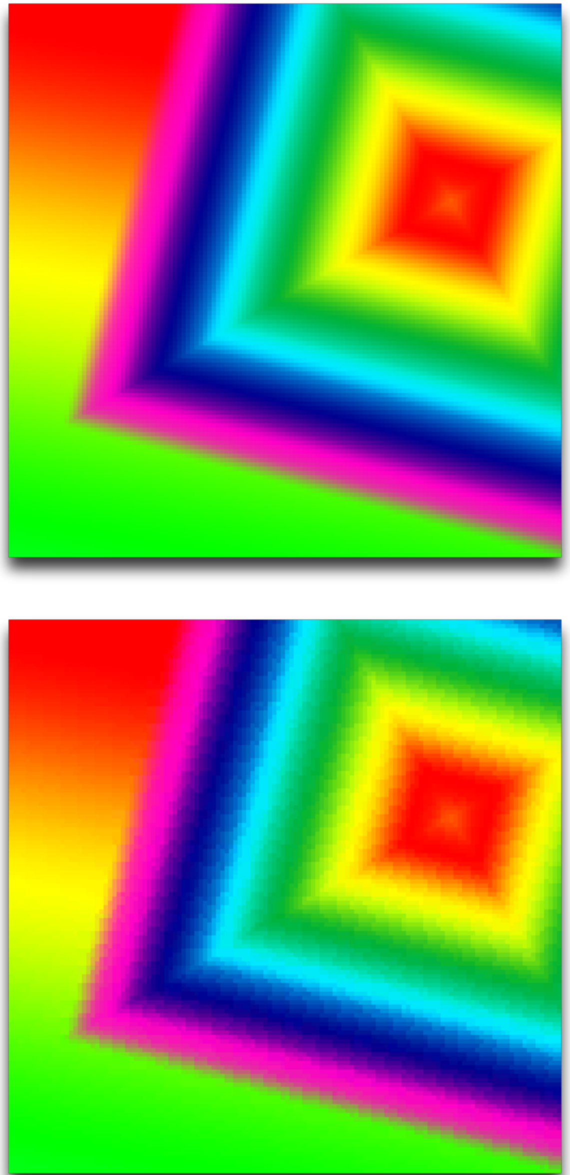
http://ls.wim.uni-mannheim.de/de/pi4/teaching/animations/

S3TC Texture Compression

- Vergleich:

DXT1 | Uncompressed

[Philipp Klaus Krause]



Uncompressed

DXT1

[Simon Brown]

- Vorteil: größere Texturen möglich → höhere Qualität
- Beispiel aus der Unreal Engine:

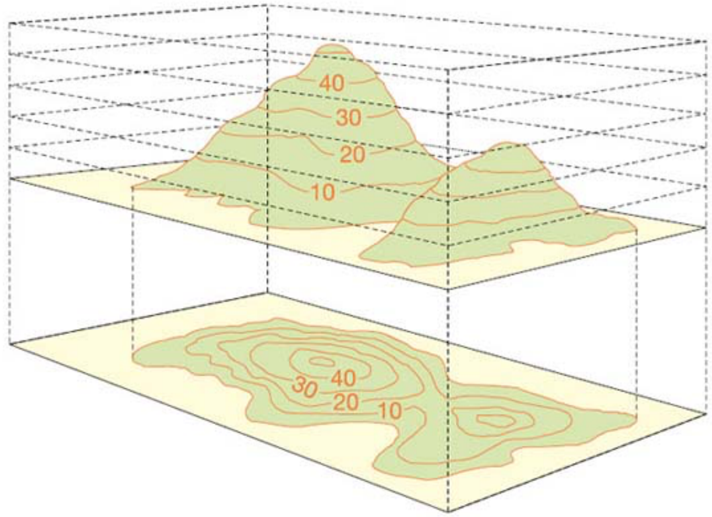
uncompressed

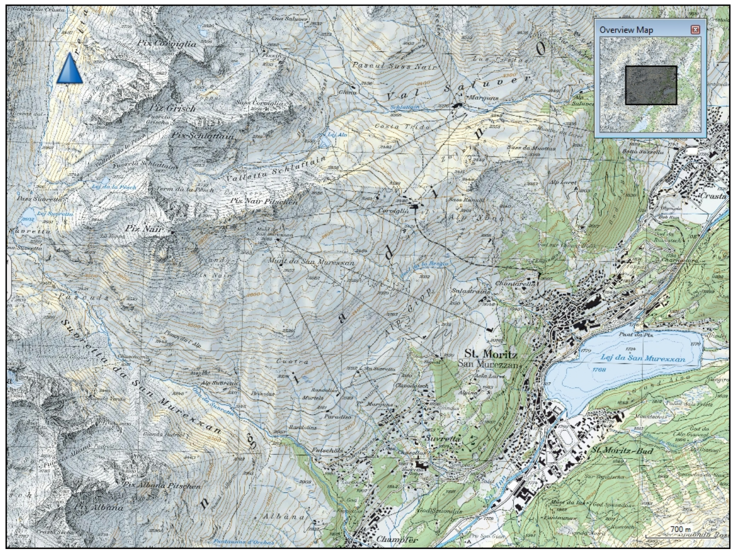
mit S3TC

Unreal Retexturing Project

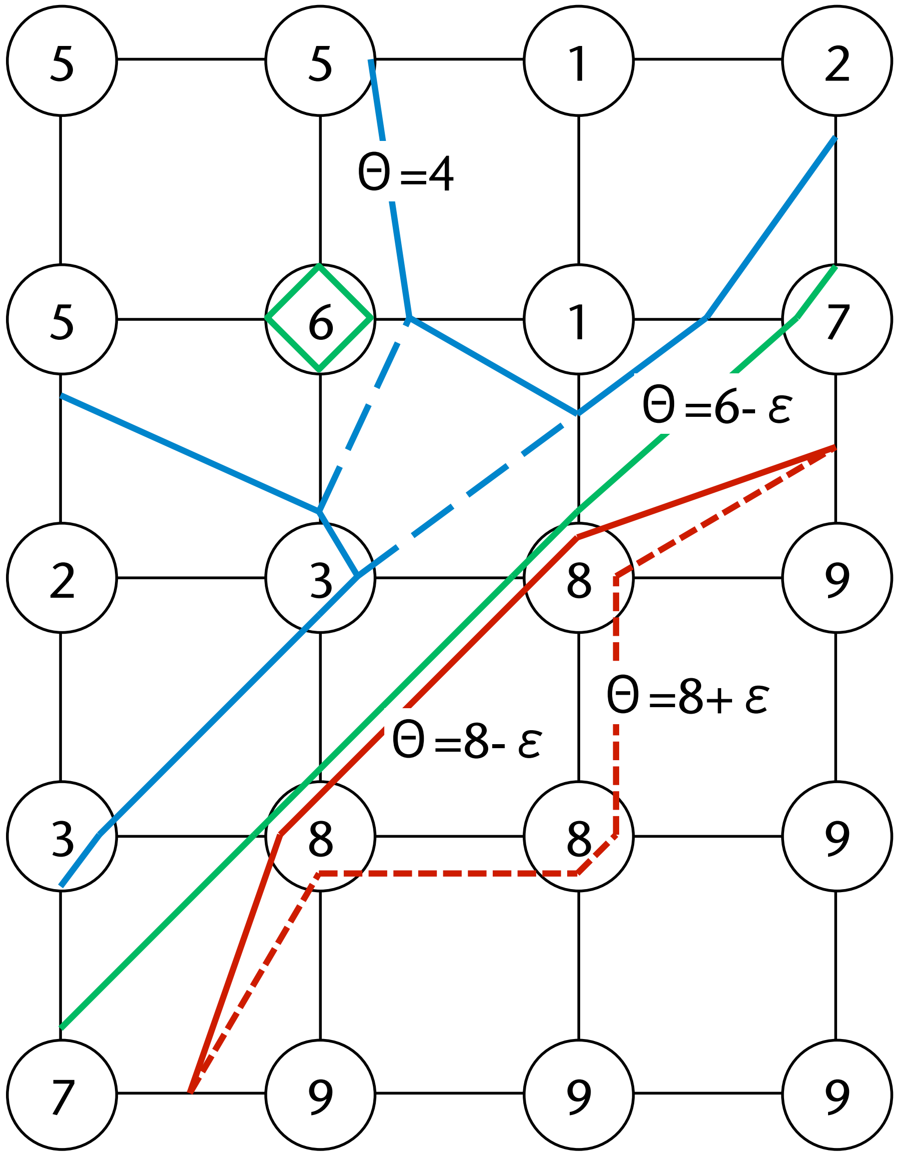
Isosurfaces

- Beispiel zur Motivation:
  - Gegeben ist ein 2D Höhenfeld
  - Gesucht ist eine Visualisierung (in 2D!), so daß man die Form / den Verlauf des Höhenfeldes gut "erkennt"
- Eine Möglichkeit: Höhenlinien = Konturen = Isolinien

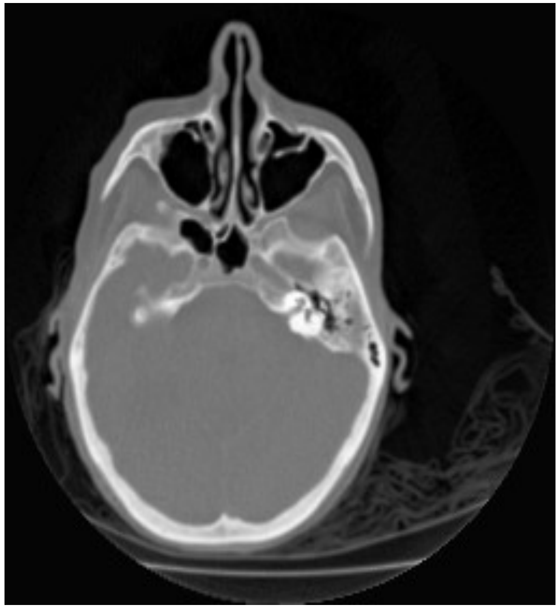




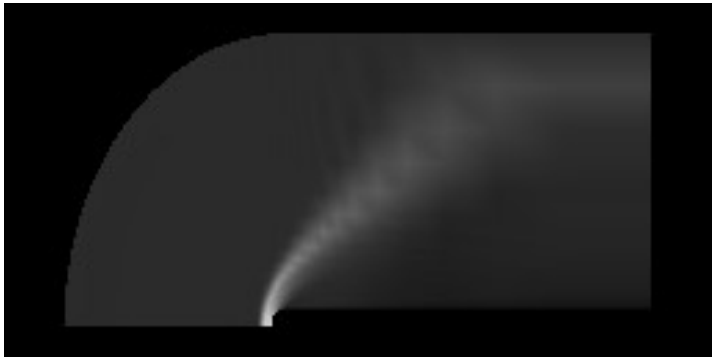
- Problems / challenges:
  - Plateaus ⟶ large "jumps" of the location of the isosurface when isovalue changes by ε
  - Singularities ⟶ isosurface contracts to a point, or appears "out of nowhere" when isovalue crosses that point
  - Ambiguities during tesselation



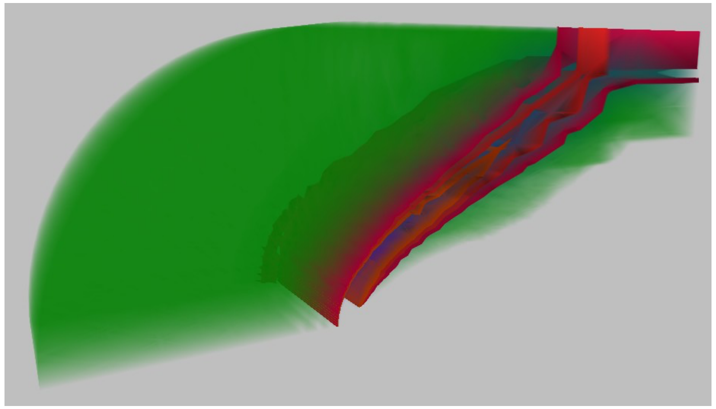
Beispiele für Volumendatensätze

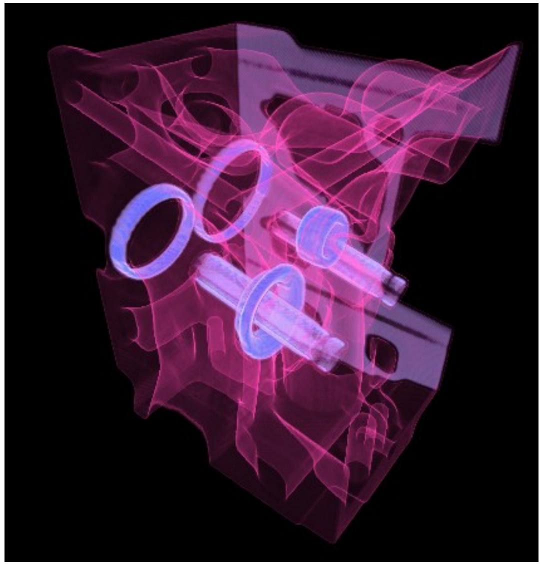



Chapel Hill CT Head



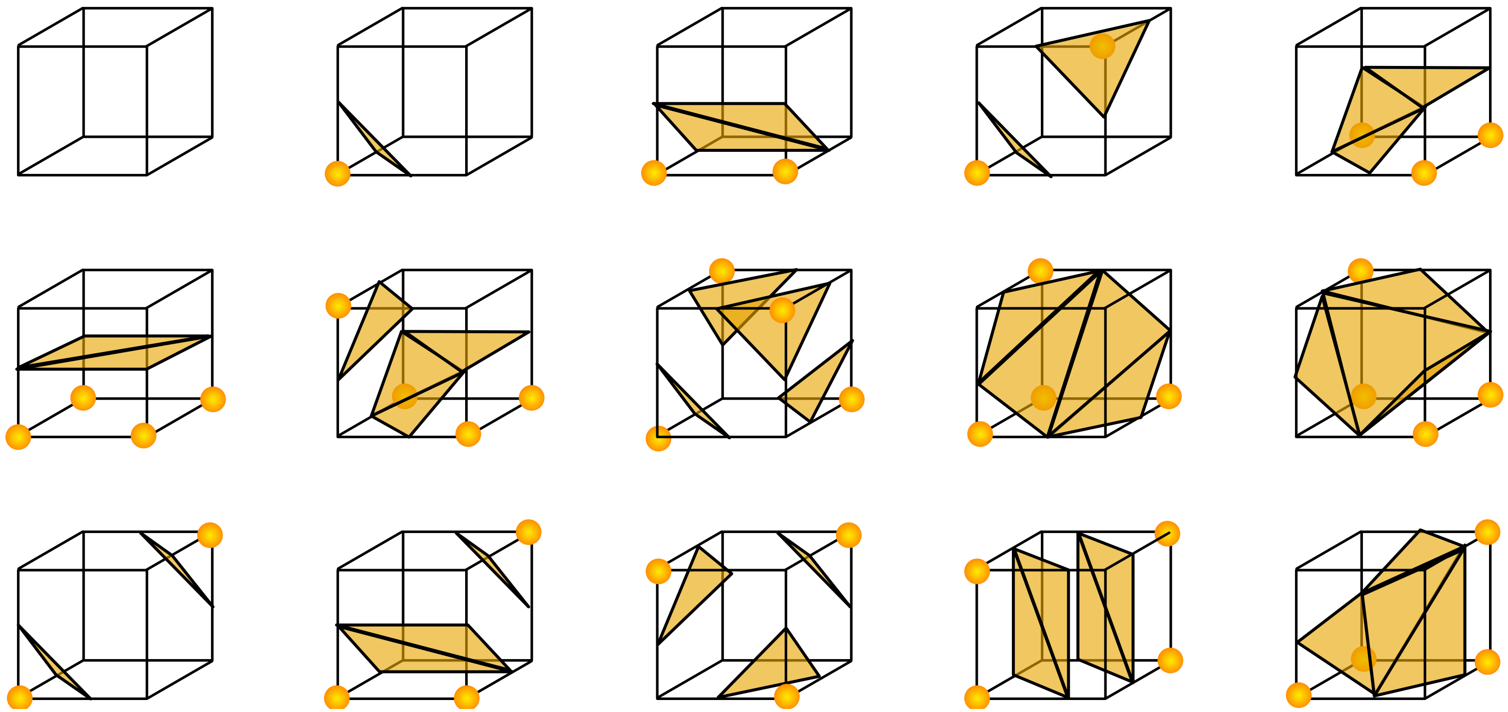
Blunt Fin



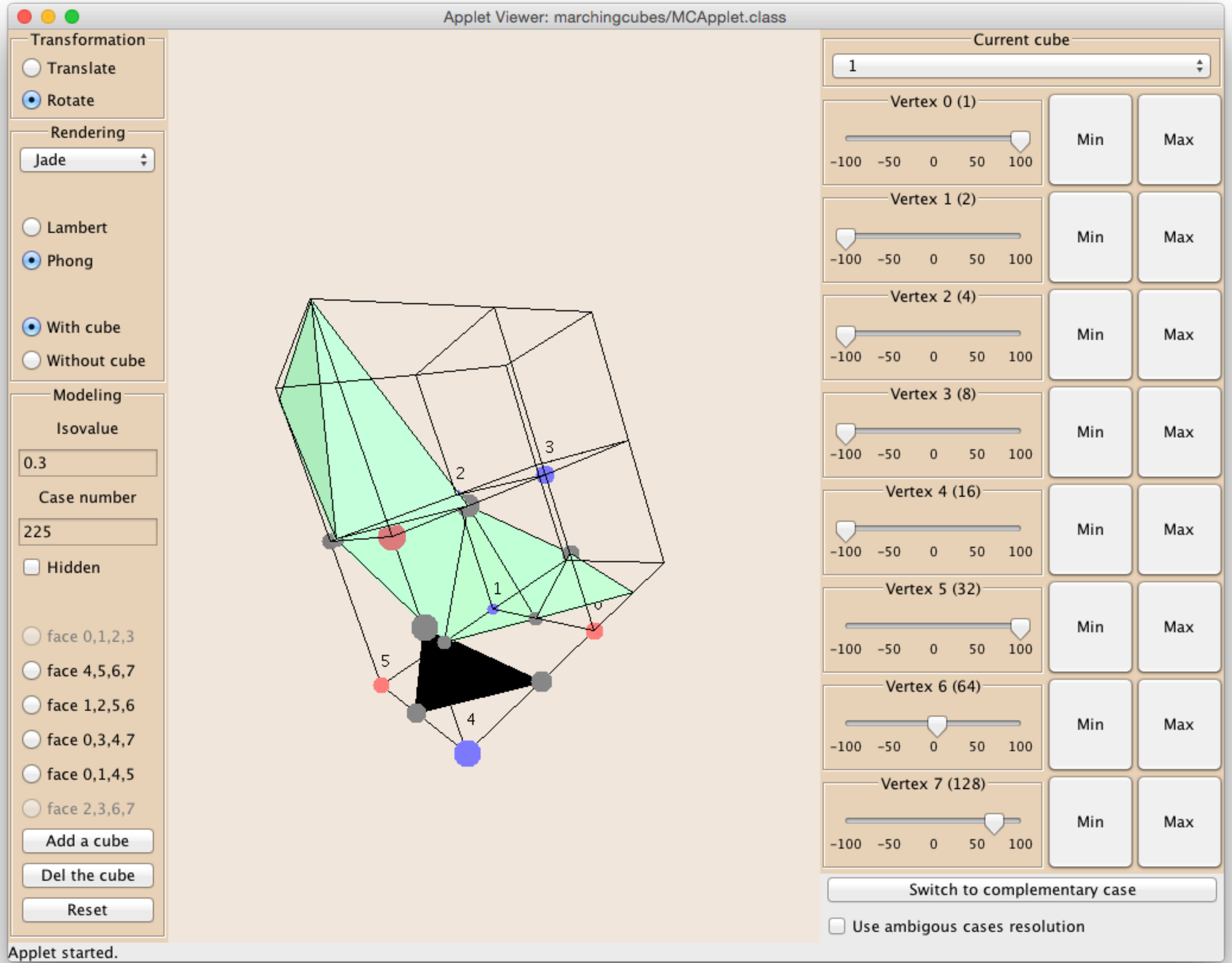



Engine Block

- Die 15 echt verschiedenen Fälle in 3D (module rotation & Spiegelung):

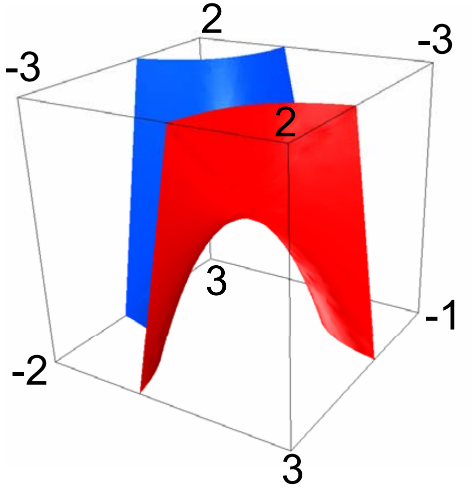


Demo

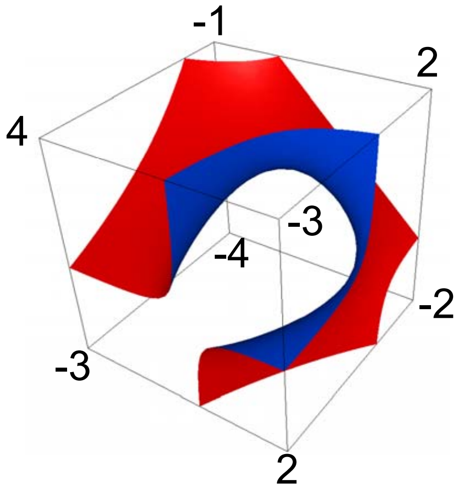


http://users.polytech.unice.fr/~lingrand/MarchingCubes/applet.html

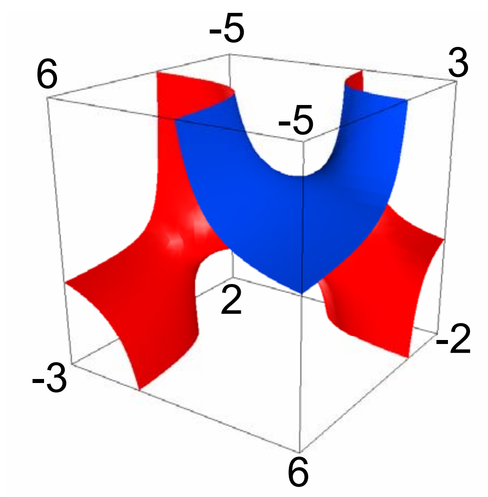
Knifflige Fälle für jeden Isosurface-Algorithmus



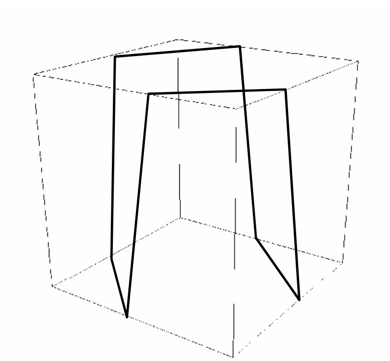
8-sided polygon



9-sided polygon



12-sided polygon

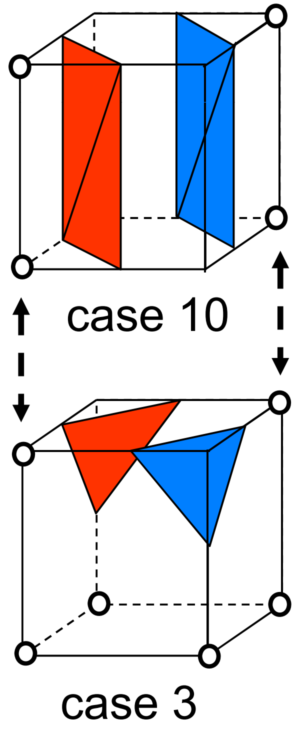


The 8-sided polygon has no valid triangulation!

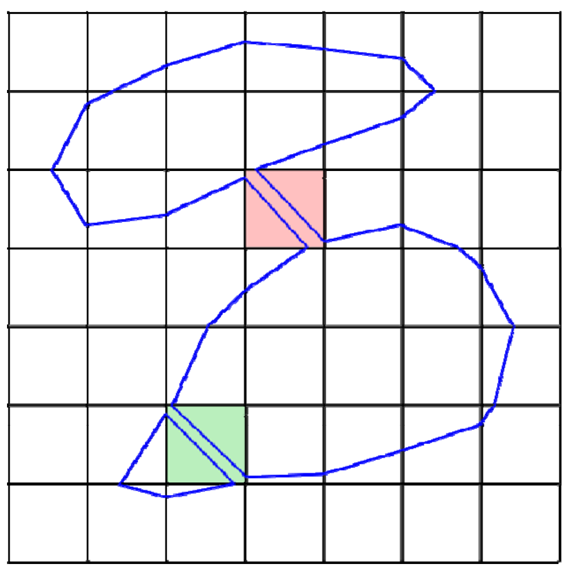
- either some triangles lie on faces of the cell
- or an extra vertex has to be used

~/avs/networks/SciVis/AD*net

- Manchmal passen die Dreiecke der benachbarten Zellen nicht zusammen:

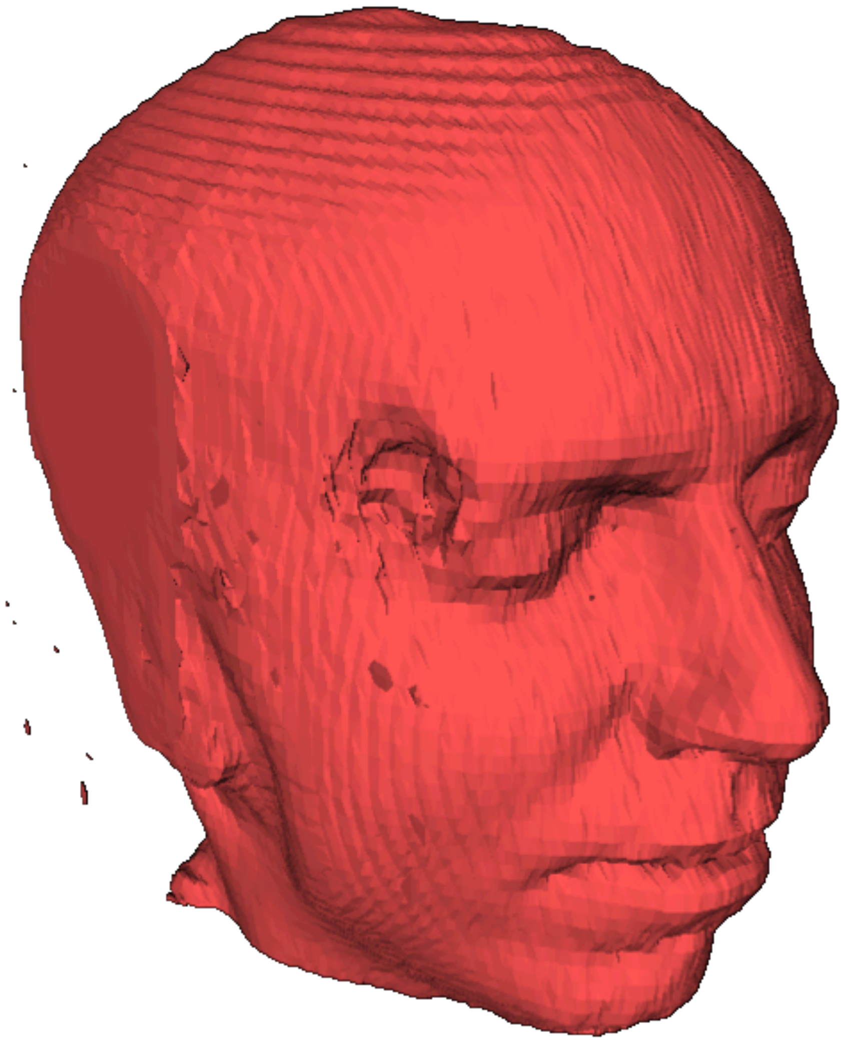


- Uneindeutiger Fall im 2D:

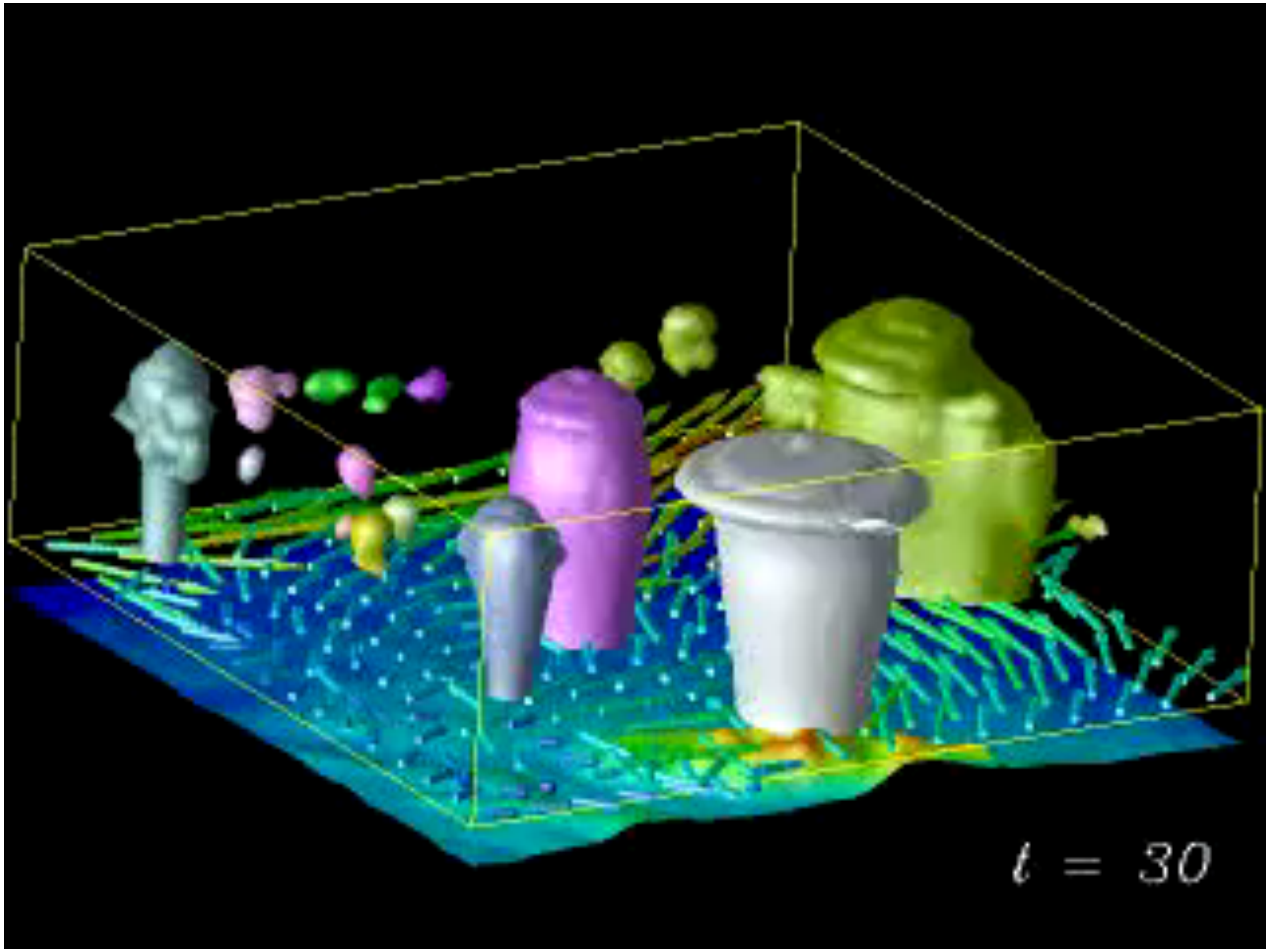


- More on that ⟶ Advanced Computer Graphics

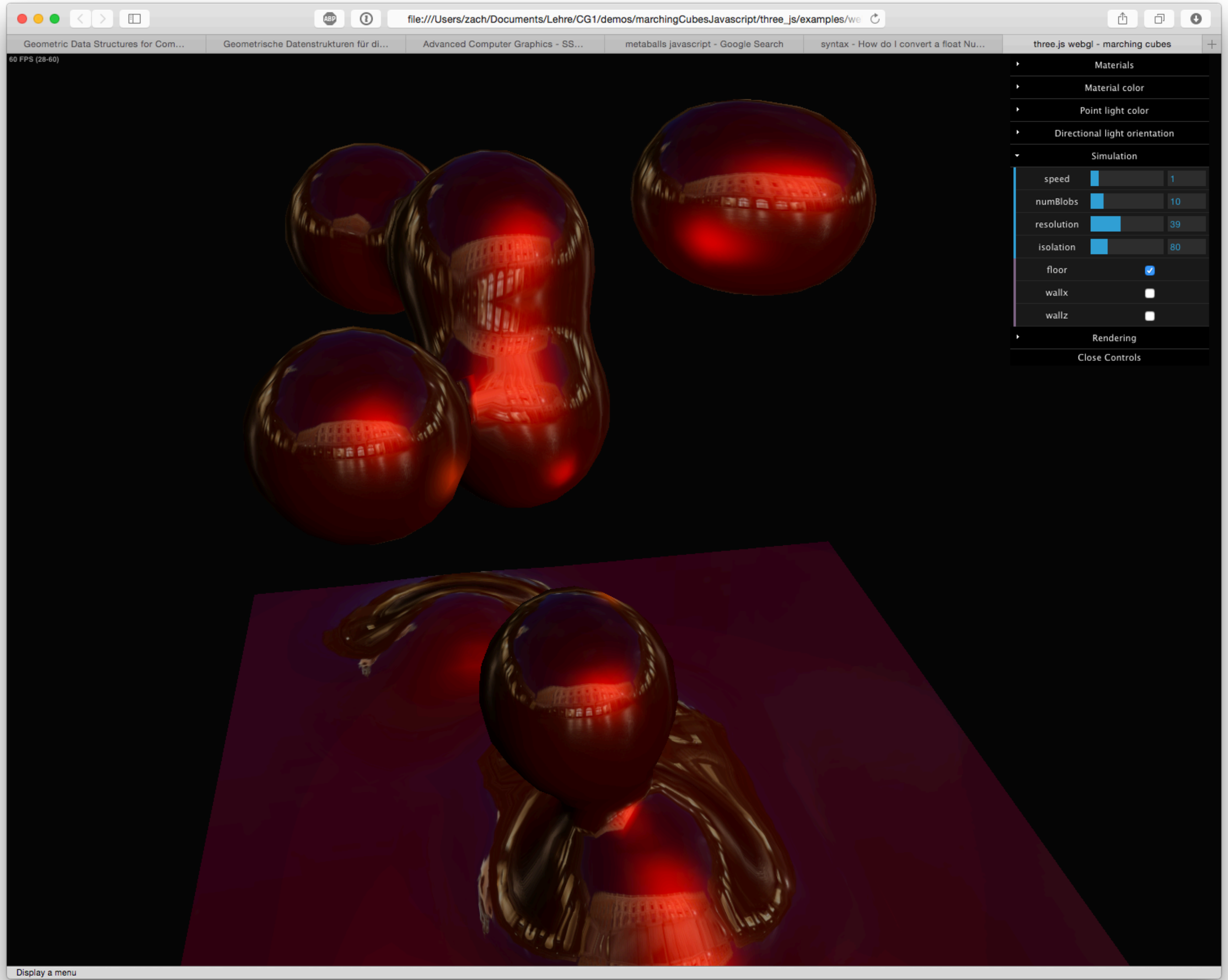
- Output eines einfachen Marching-Cube-Algorithmus':



Beispiel aus einer Wetter-Simulation

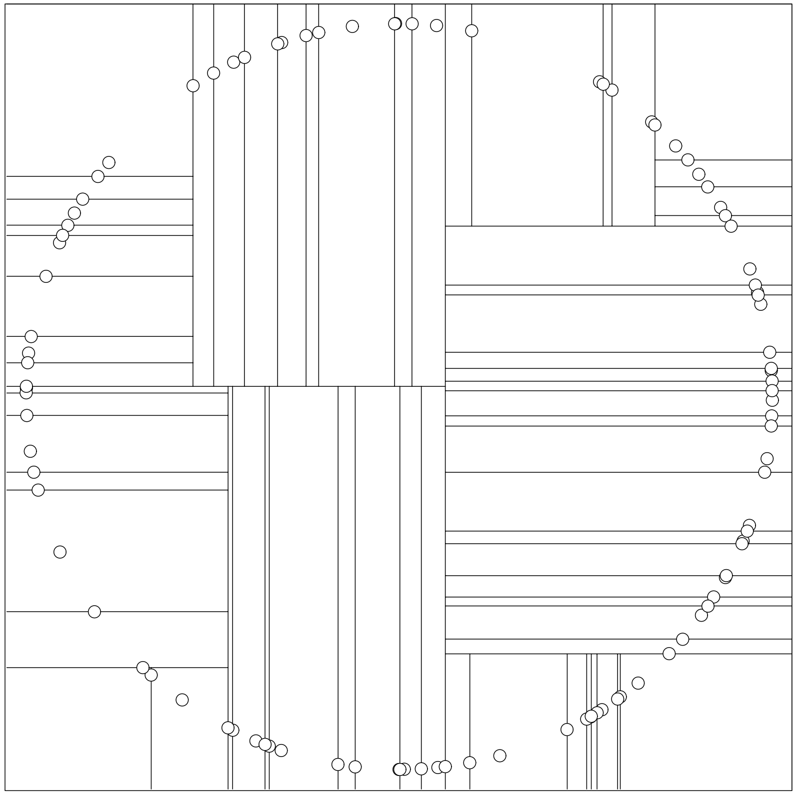


Another Metaballs Demo

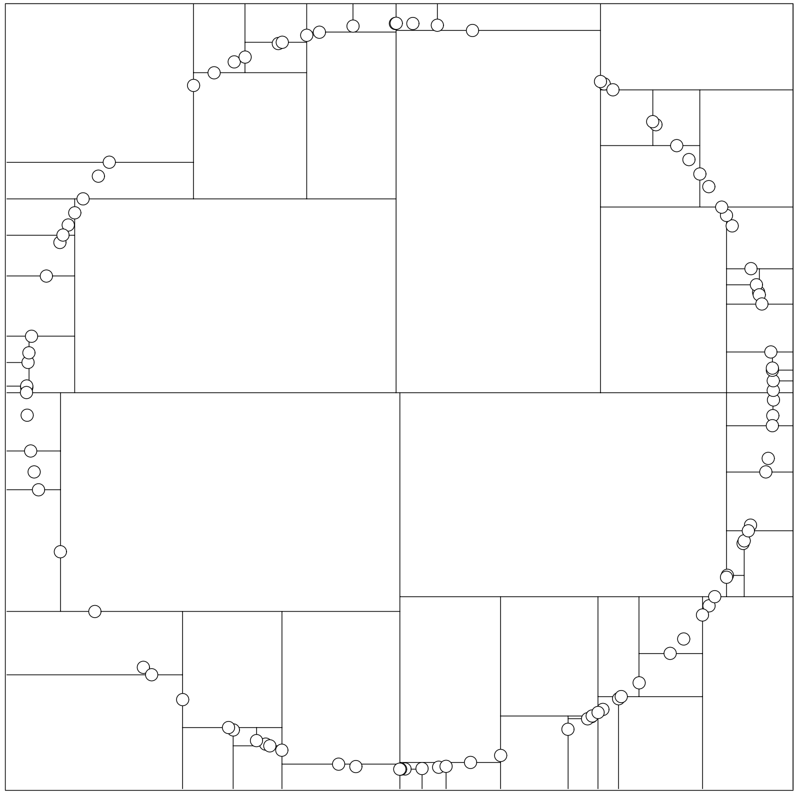


http://threejs.org/

Pivot-Strategien beim Aufbau von kd-Trees

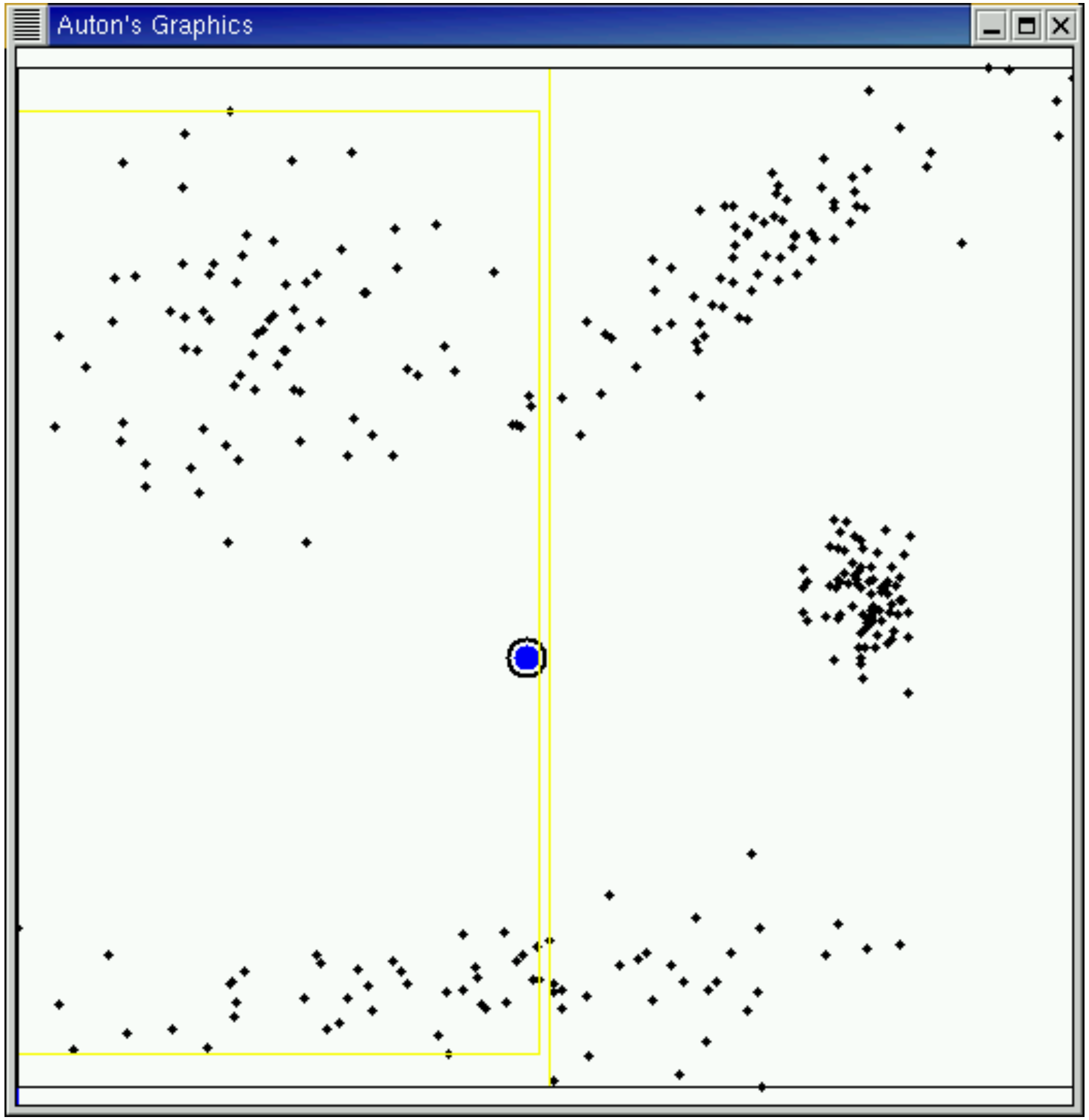


Median along the dimension with the widest spread of the points

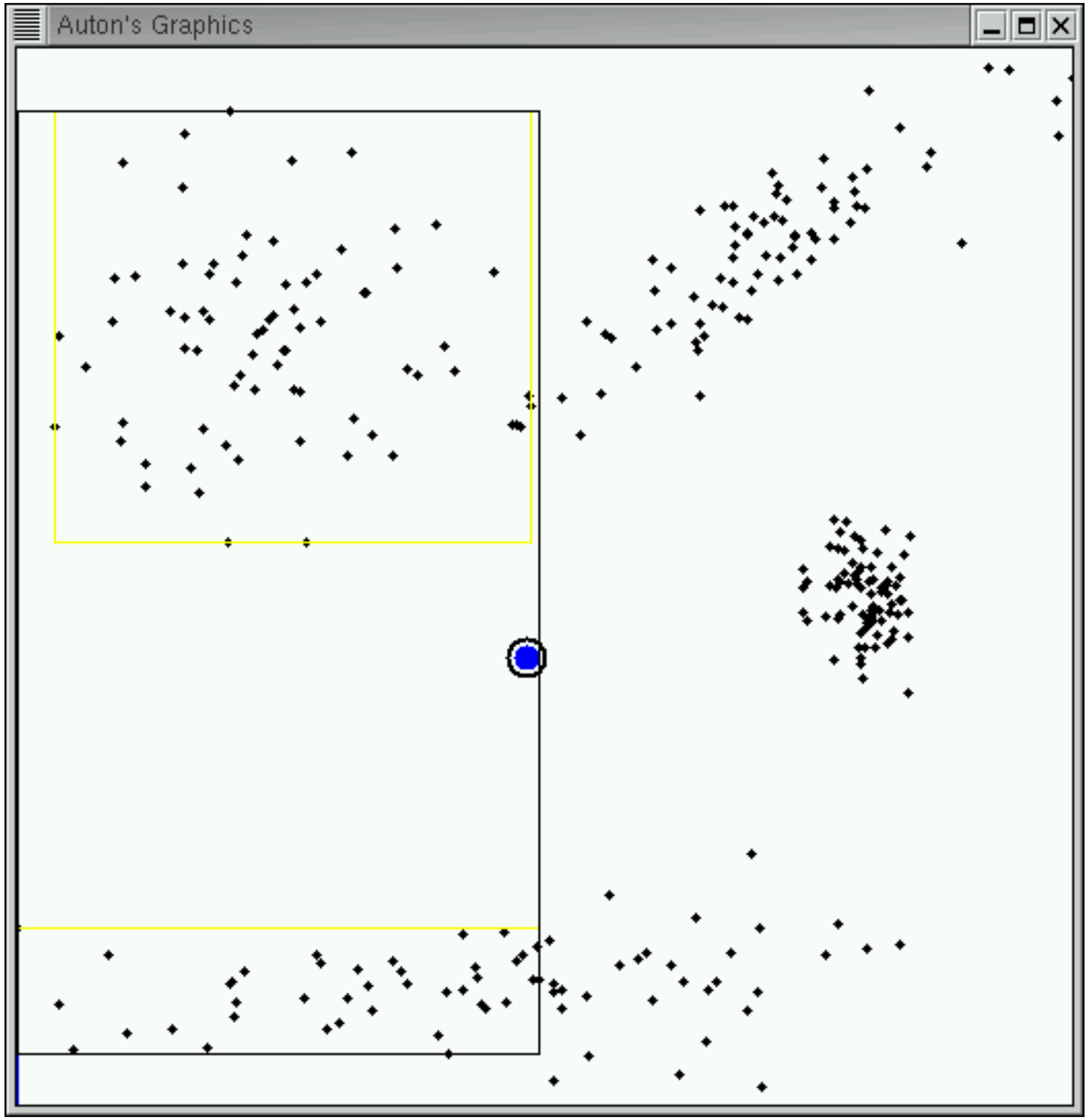


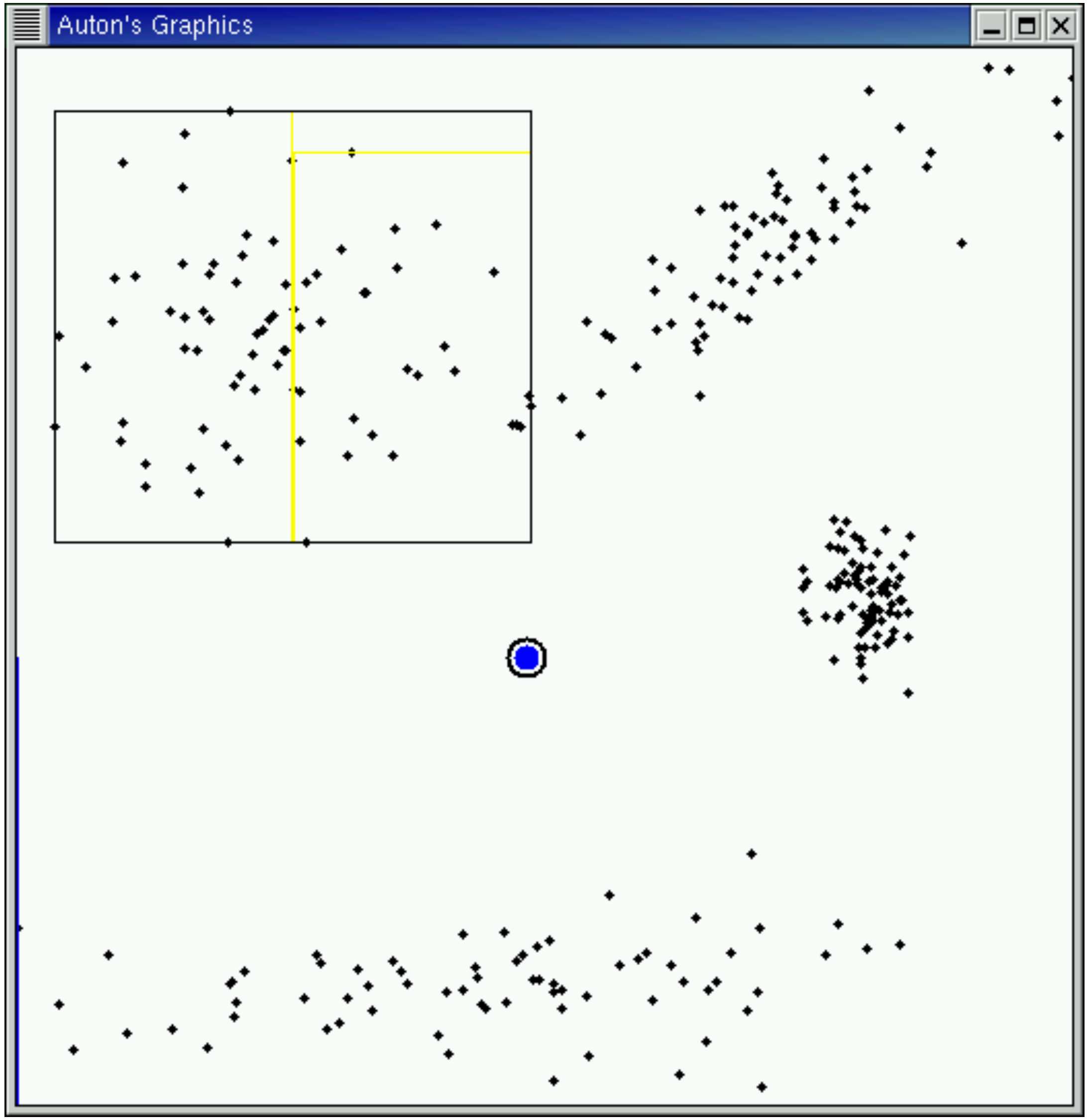
The point closest to the center along the dimension with longest side of the region

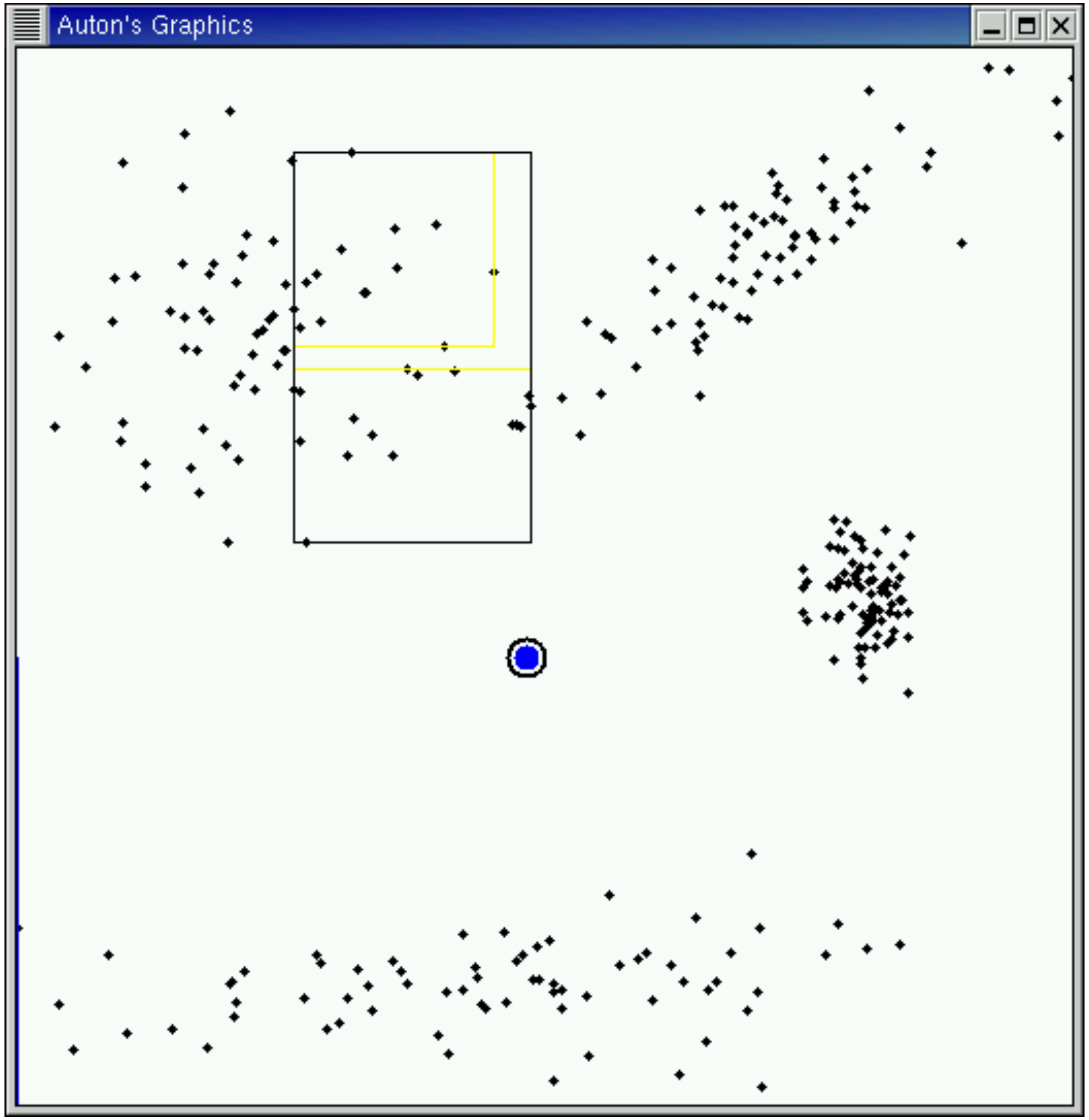
Animation of Nearest-Neighbor using kd-Trees

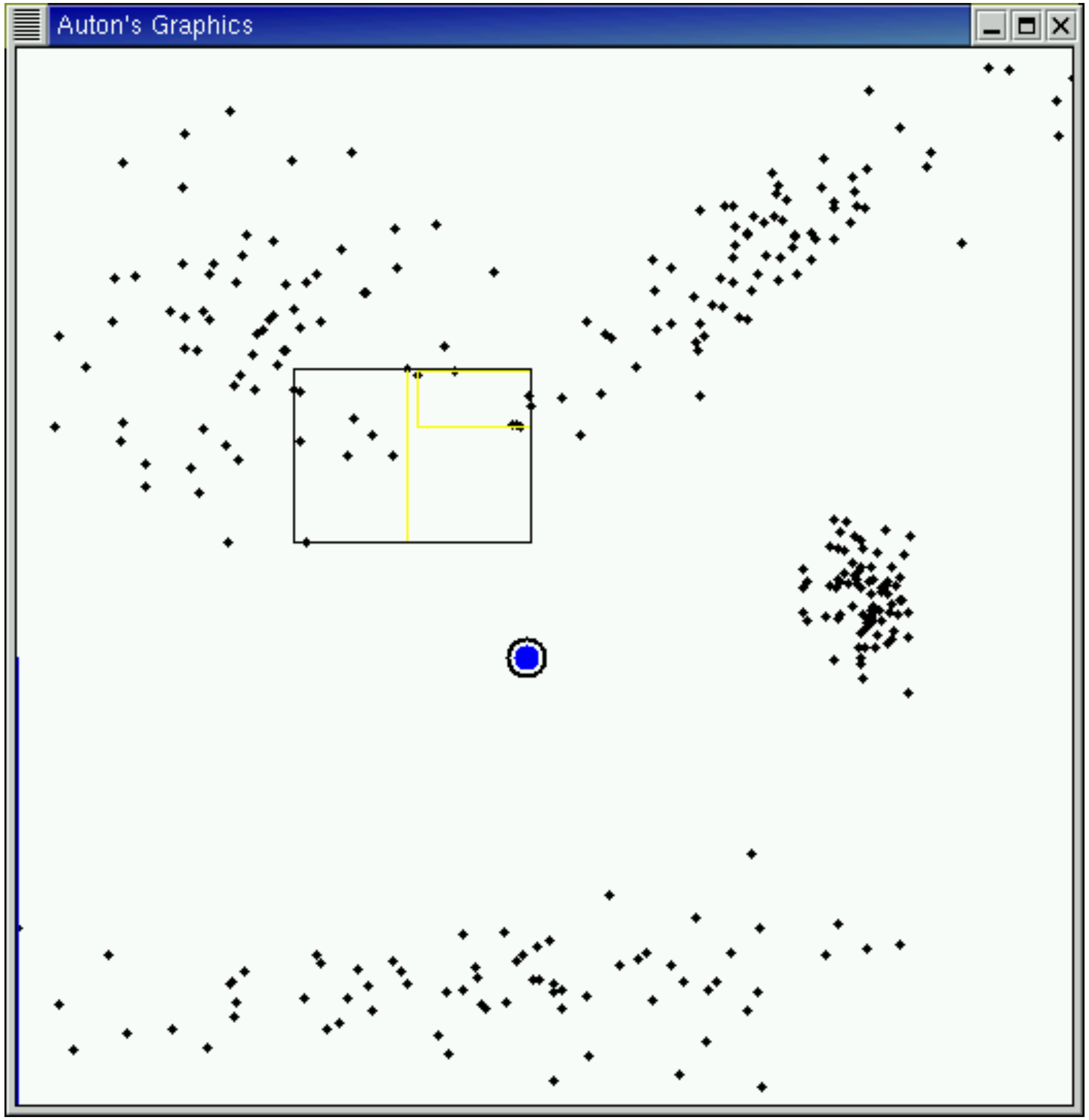


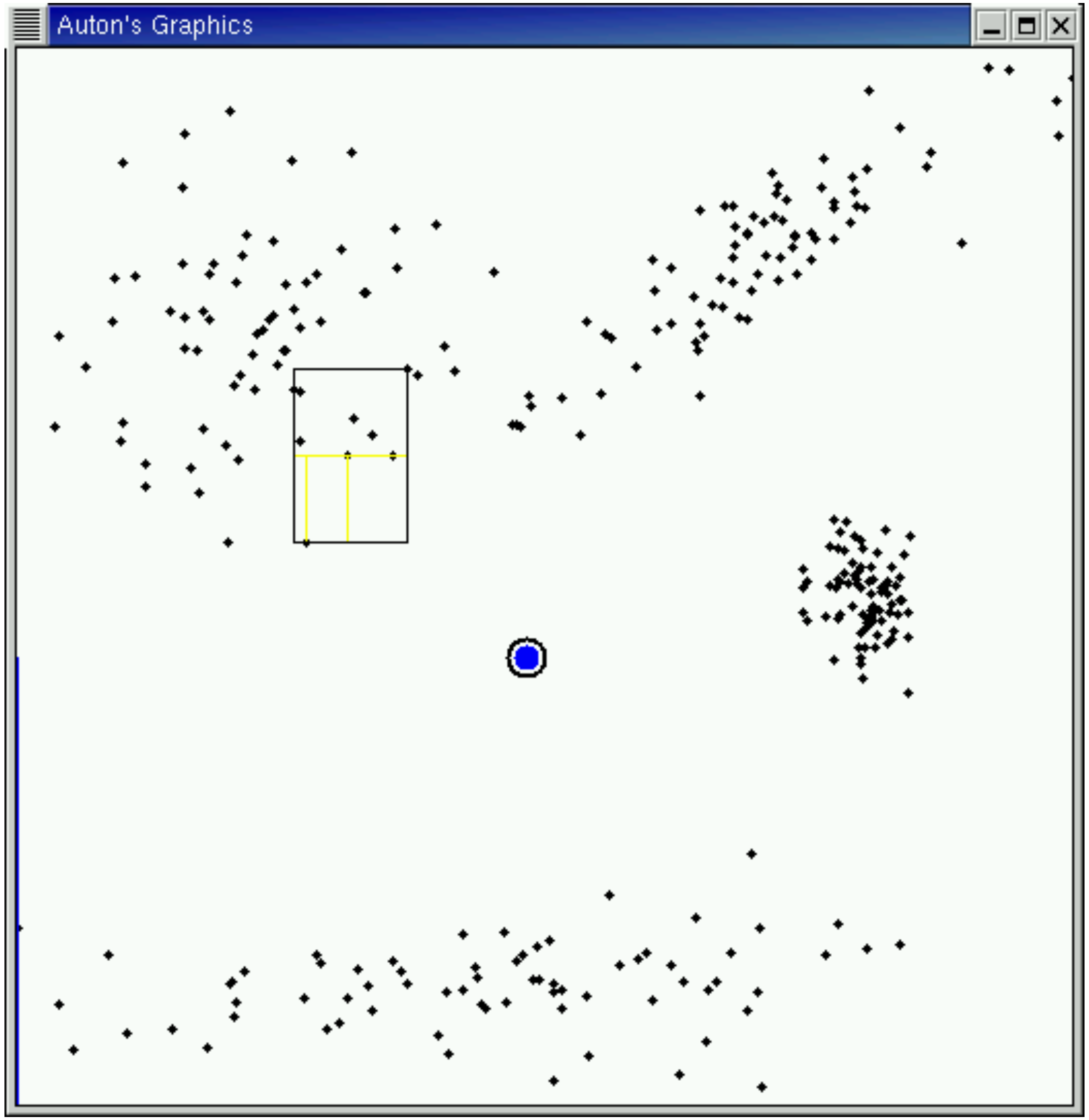
Andrew Moore, CMU

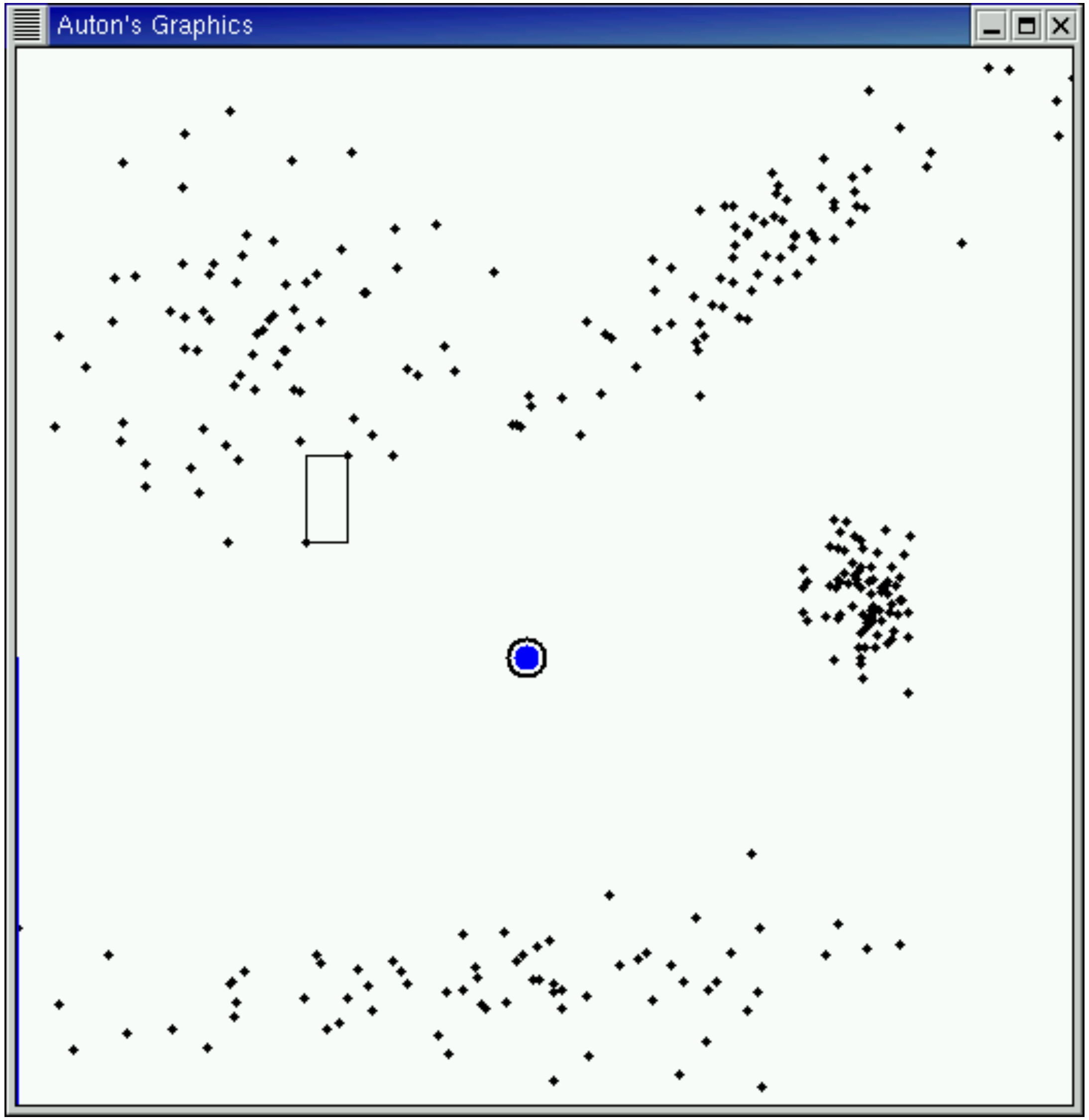

Auton's Graphics

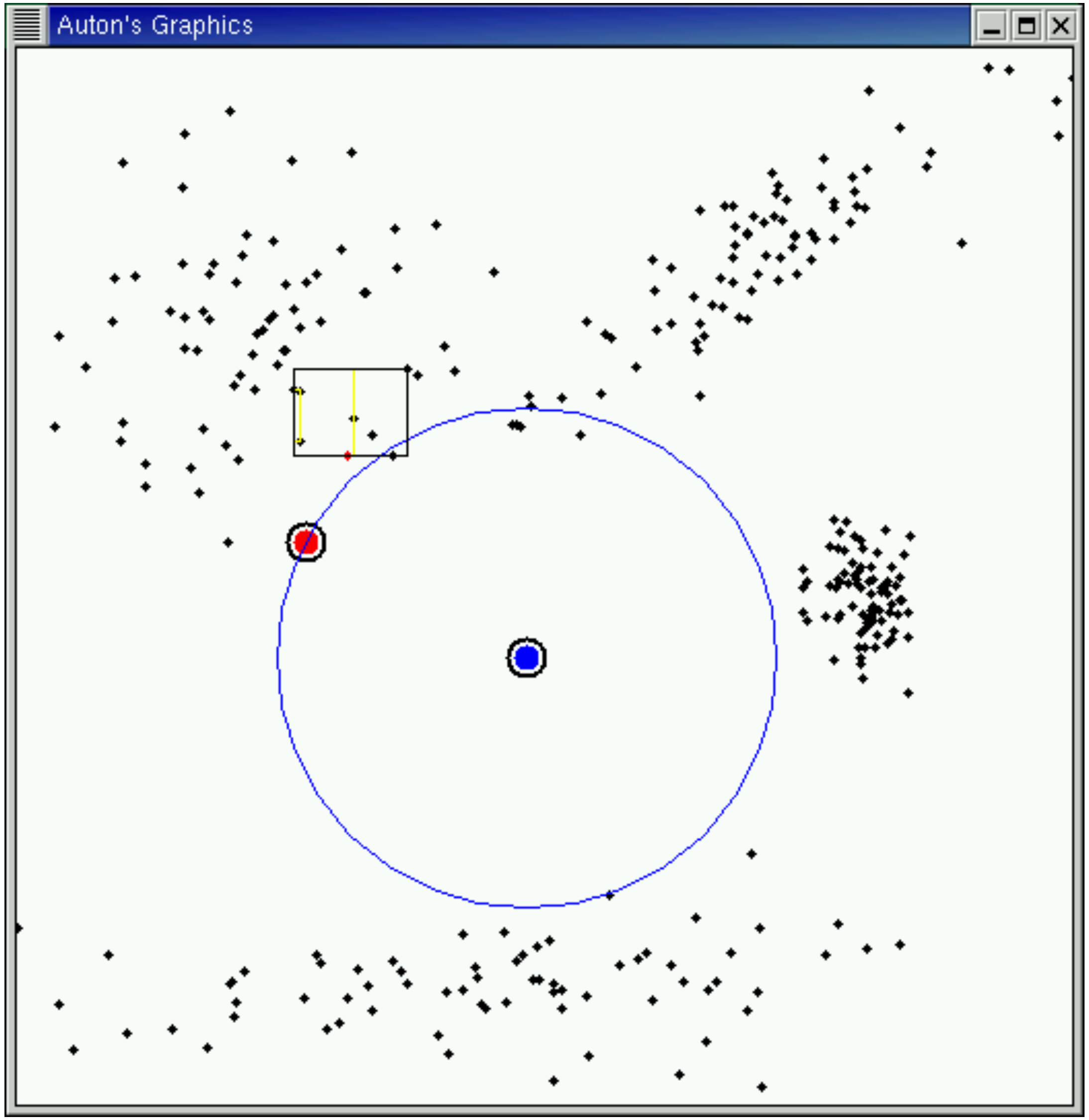

Auton's Graphics

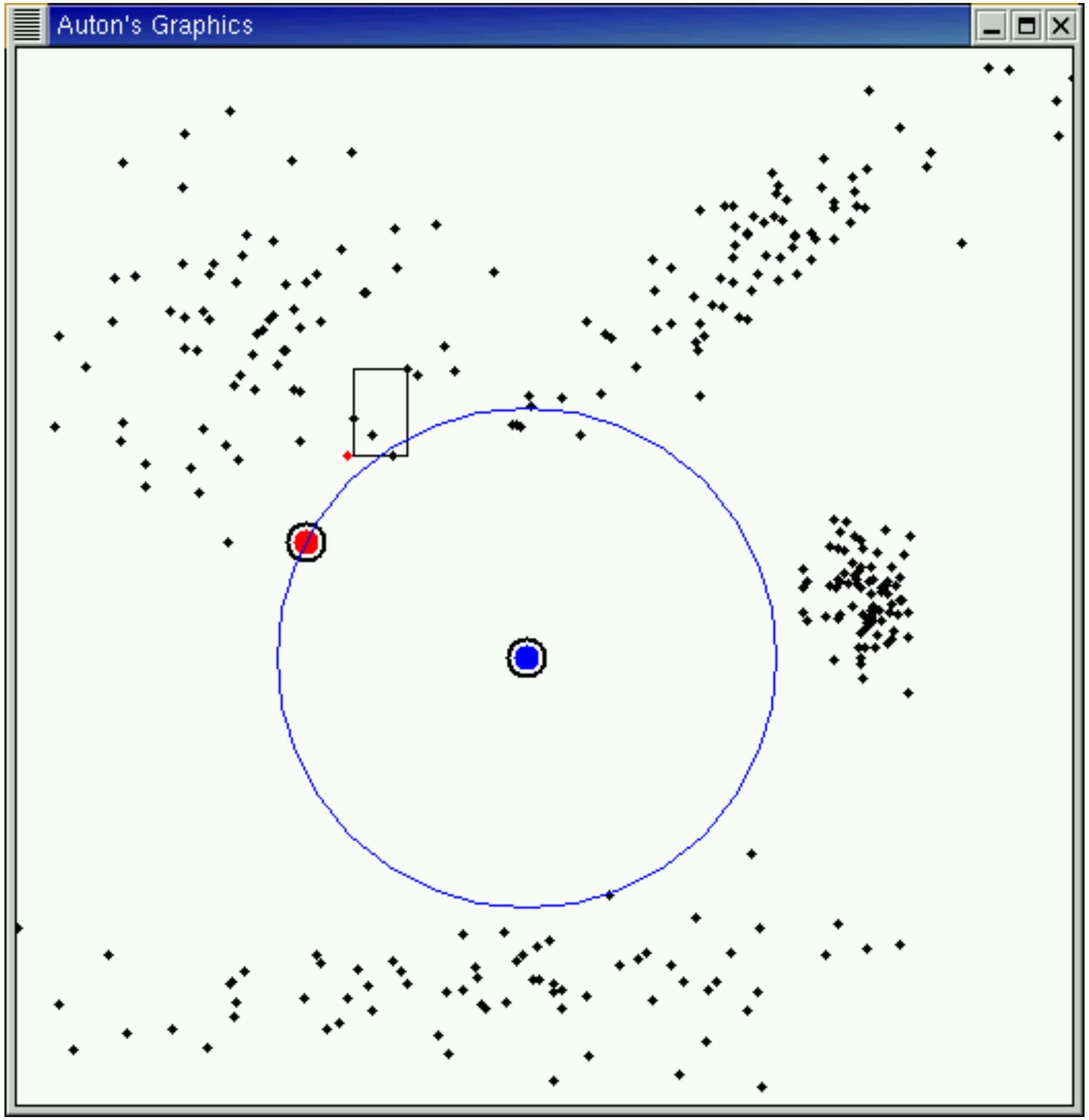

Auton's Graphics

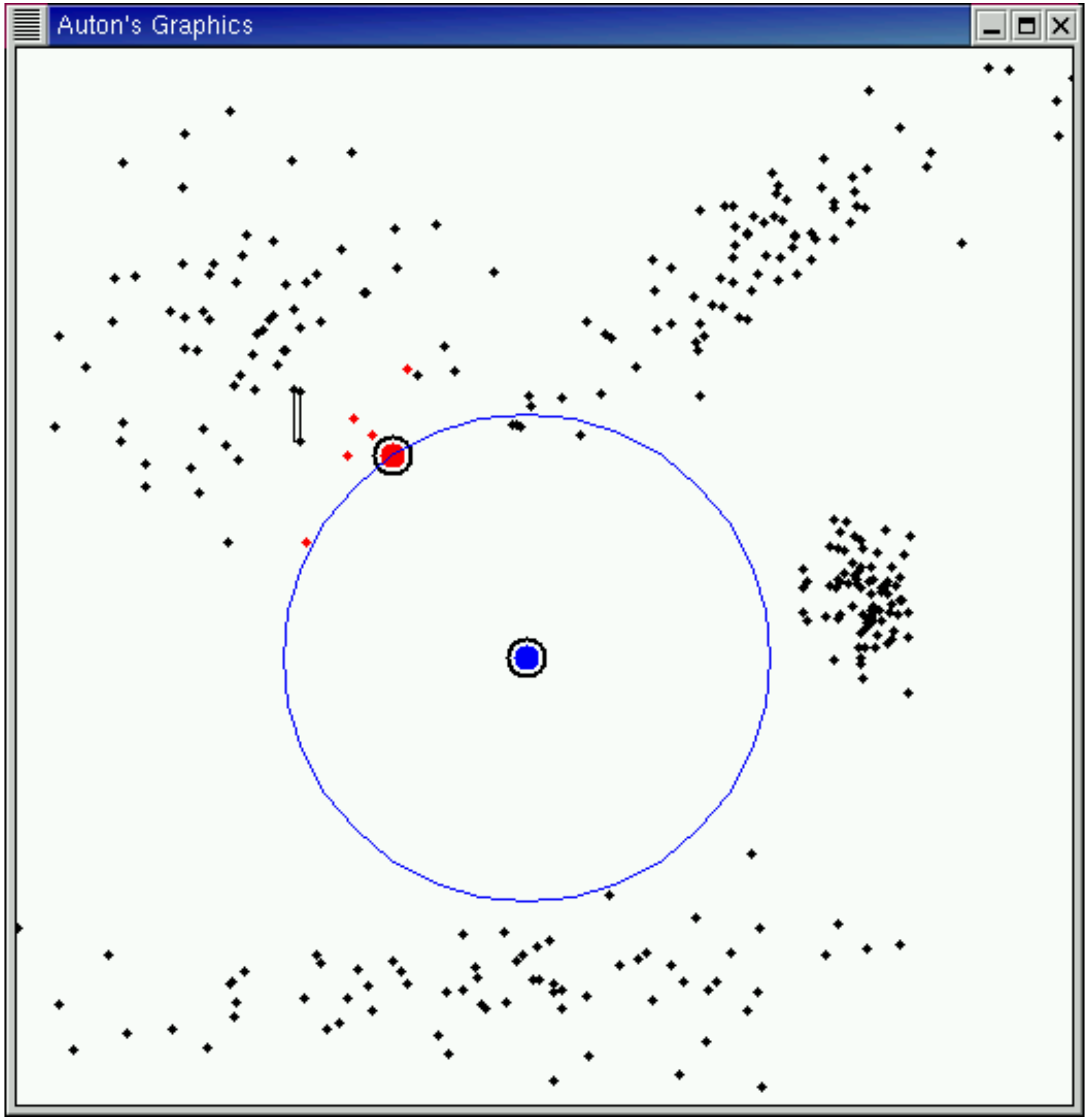

Auton's Graphics

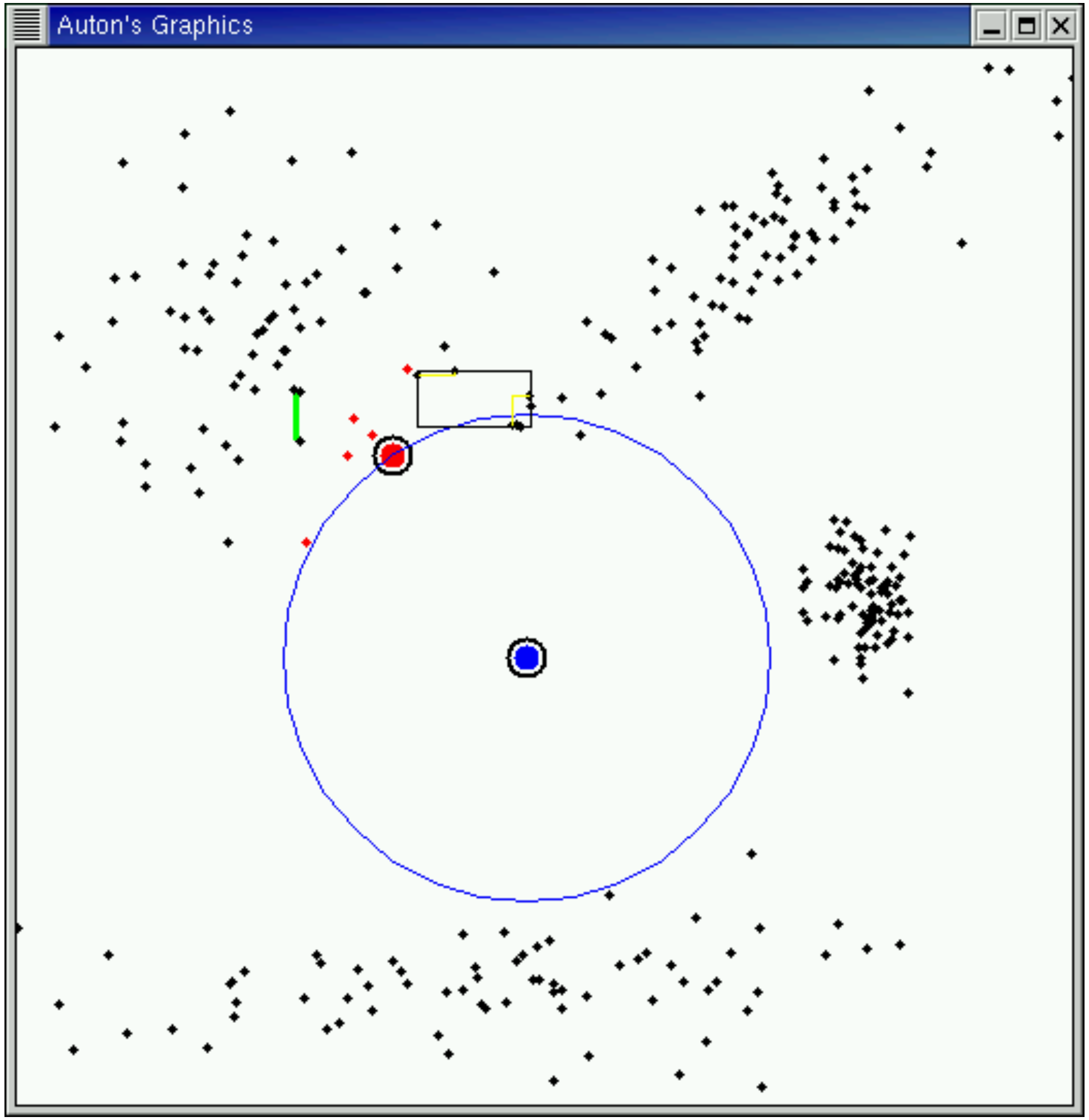

Auton's Graphics

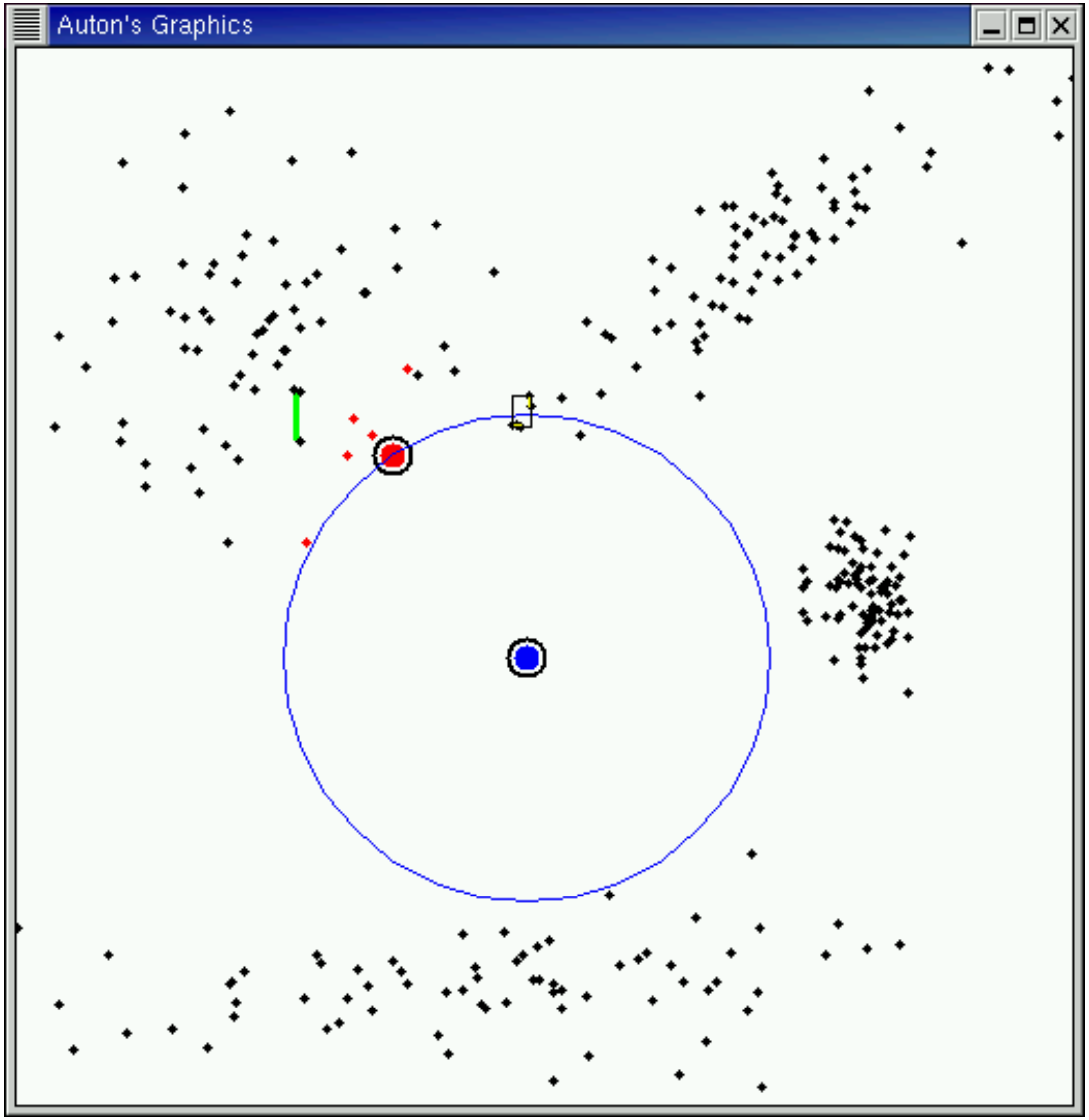

Auton's Graphics

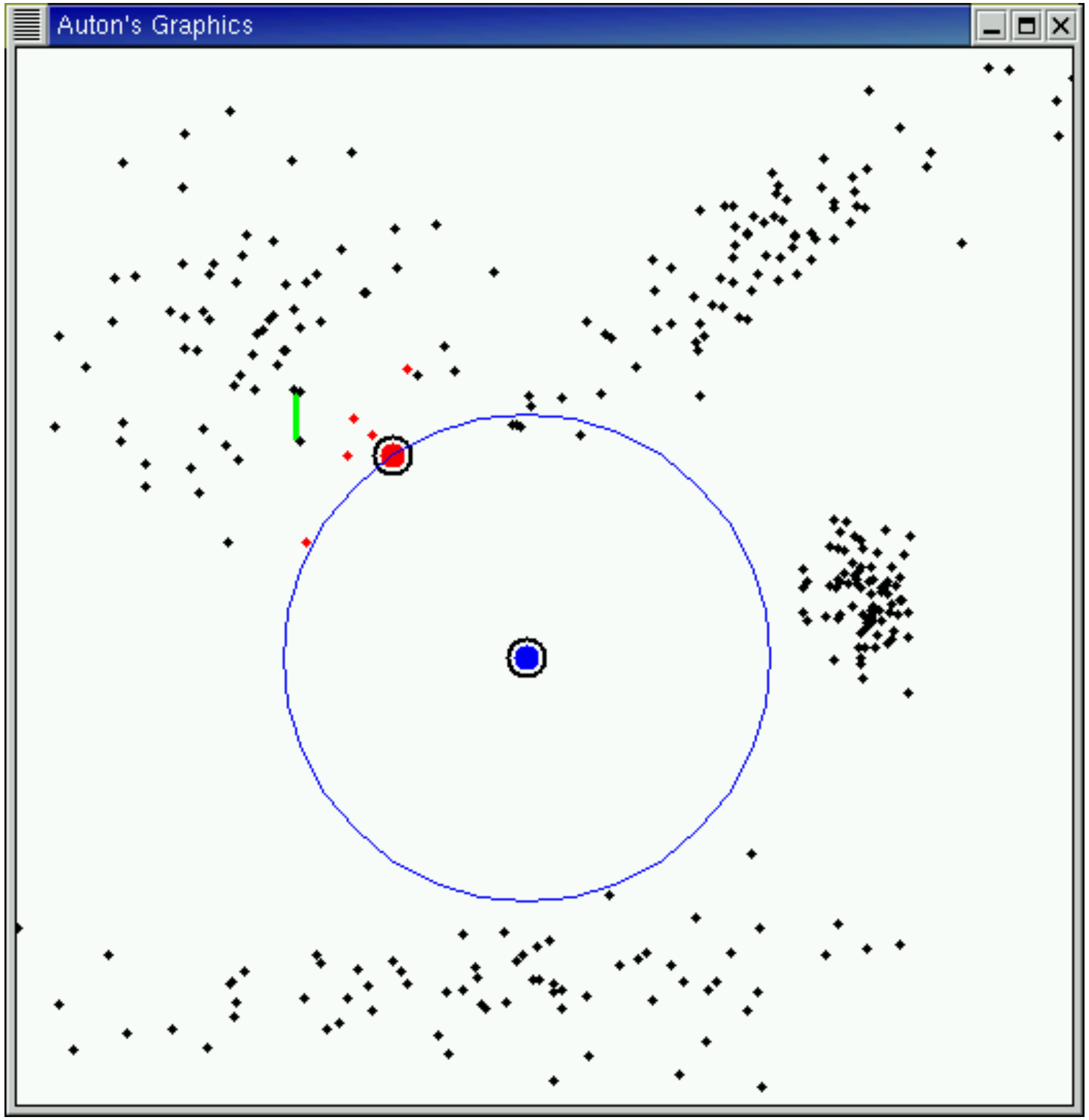

Auton's Graphics

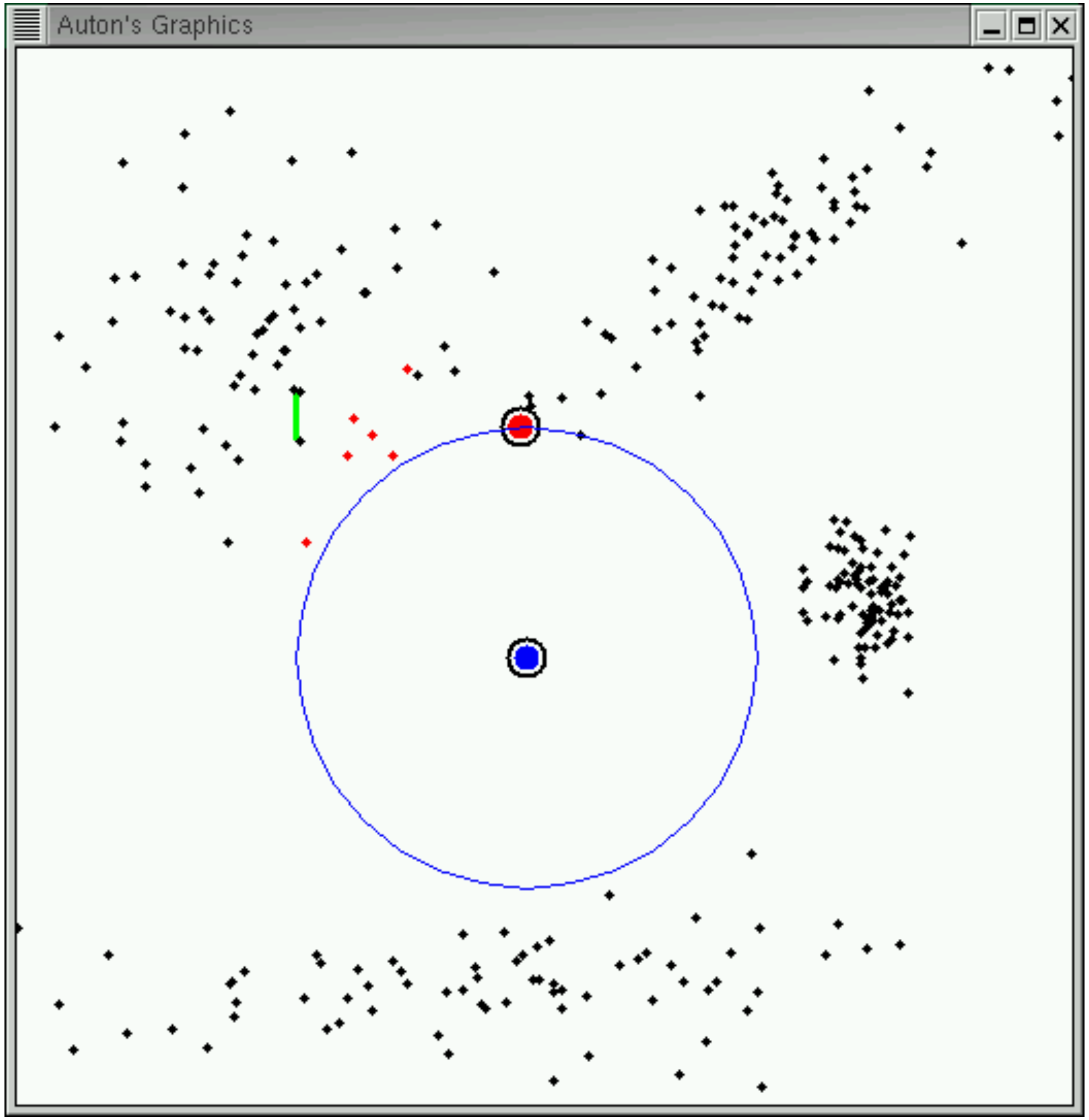

Auton's Graphics

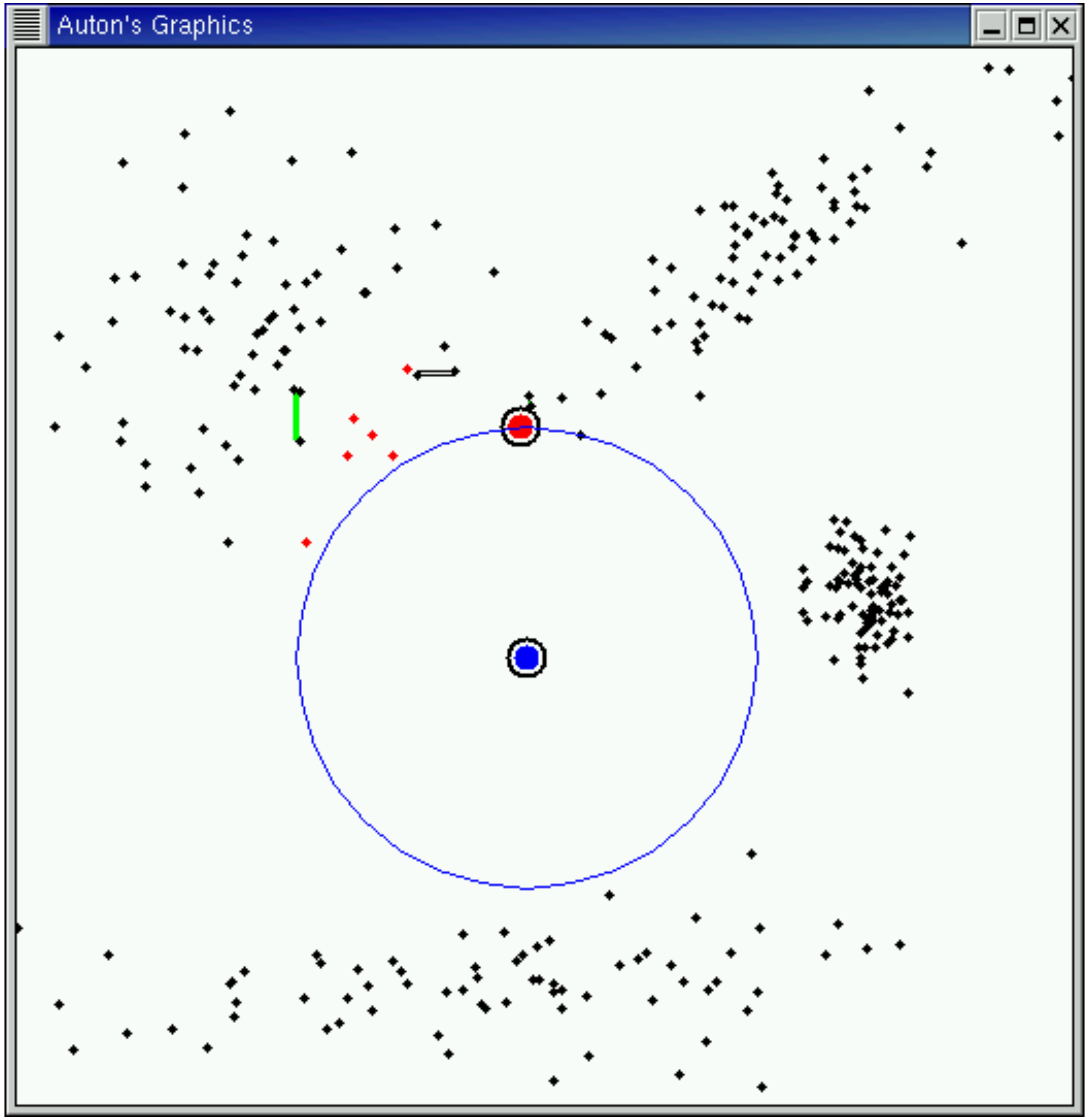

Auton's Graphics

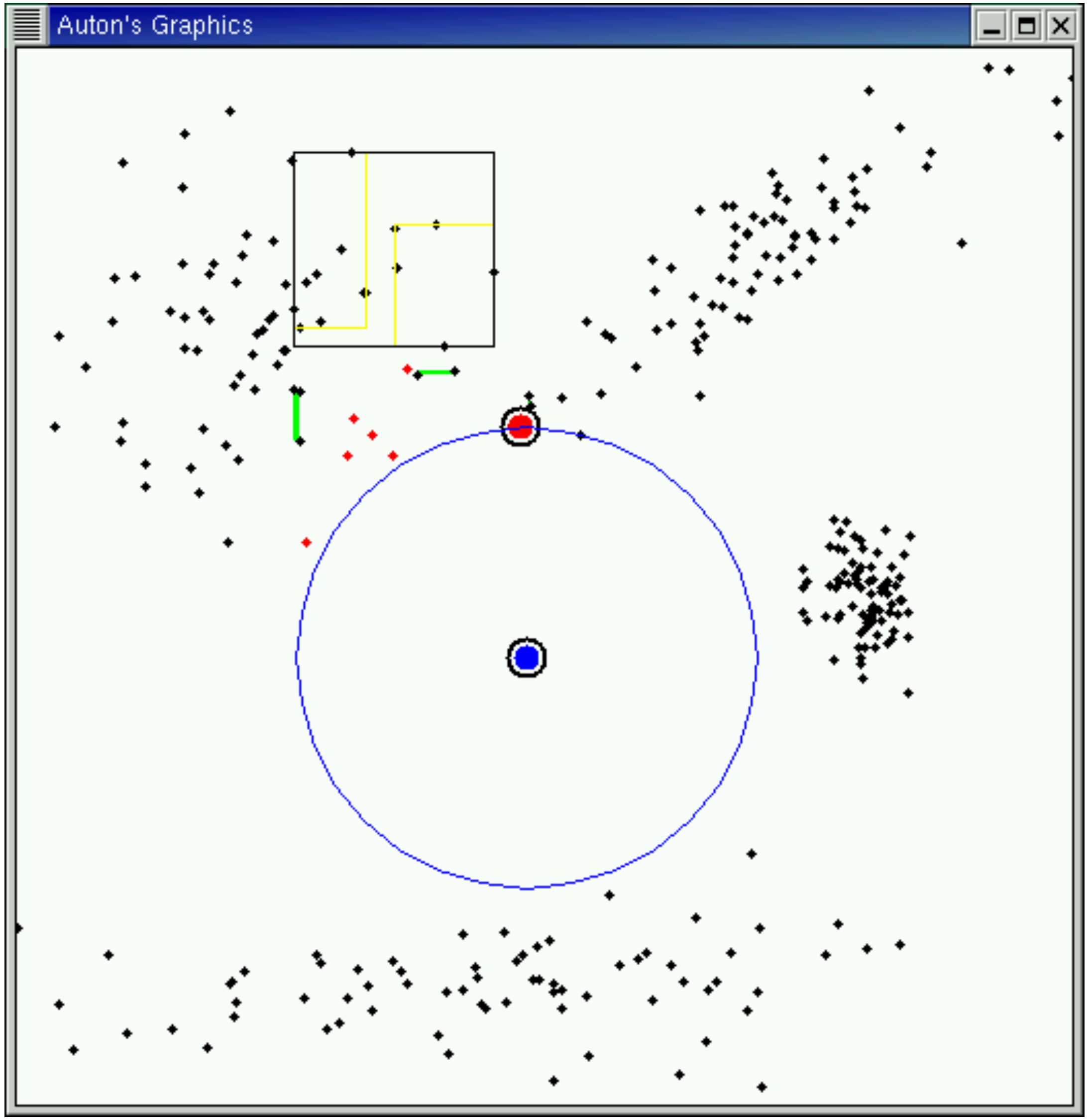

Auton's Graphics

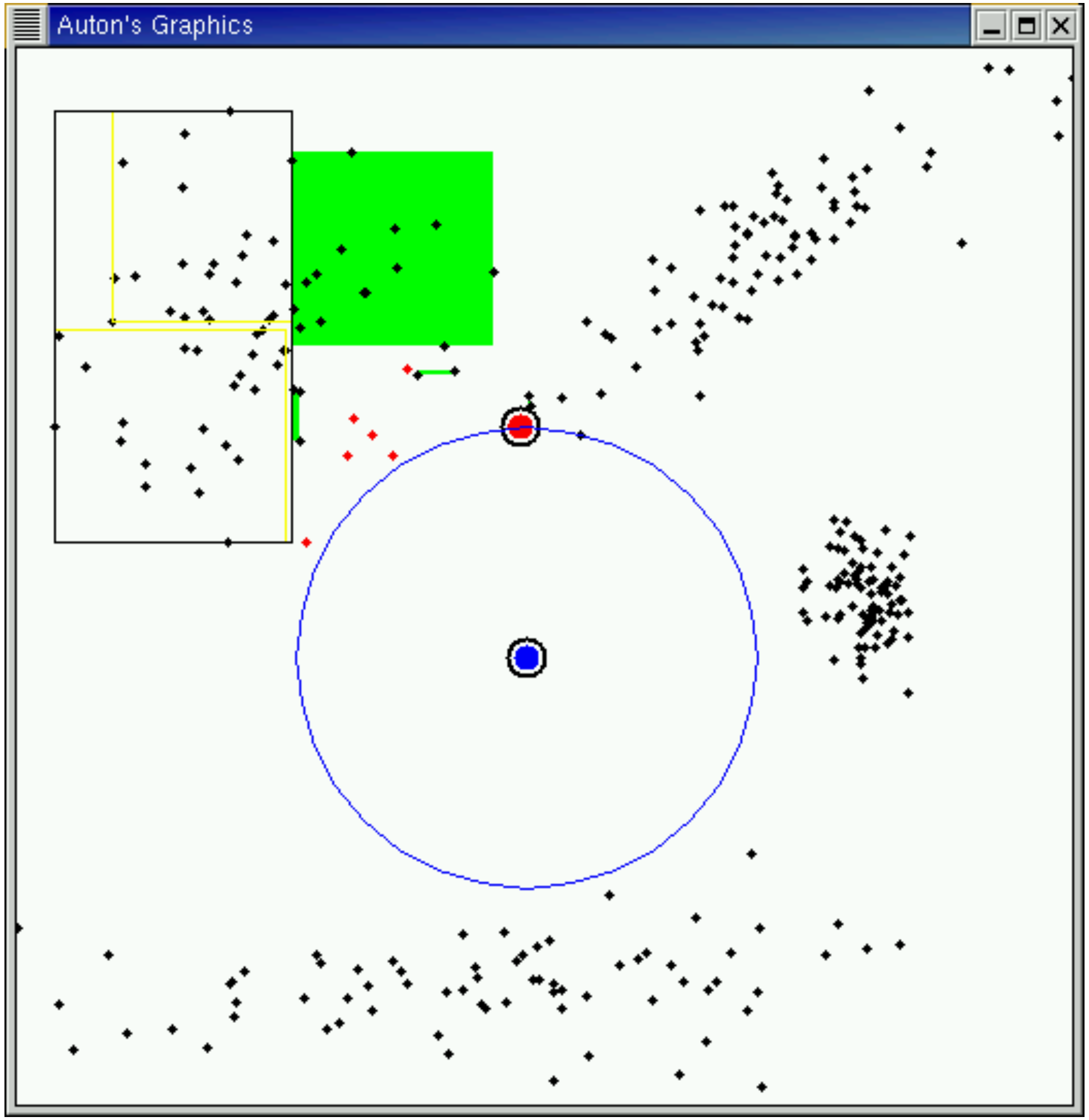

Auton's Graphics

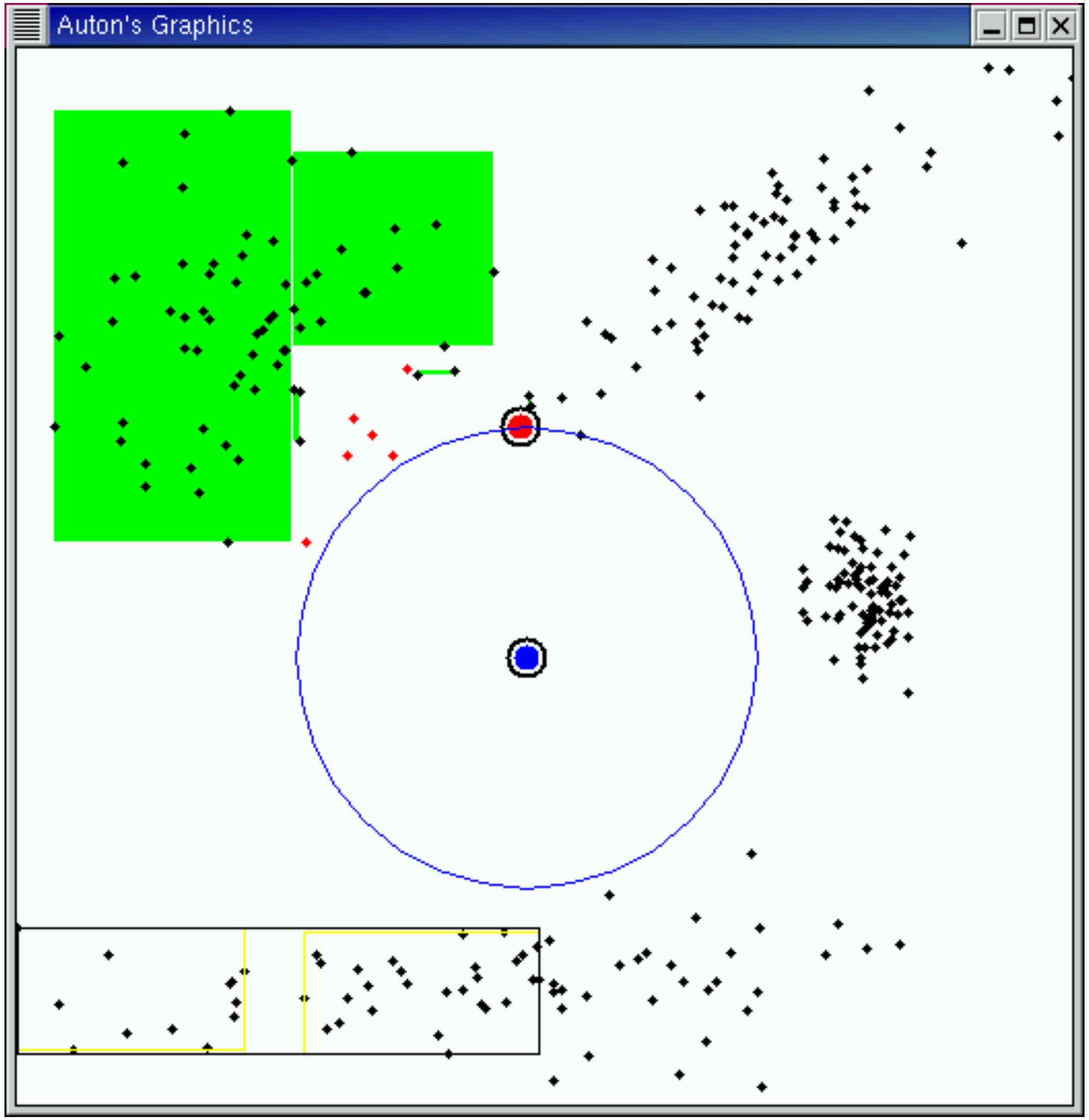

Auton's Graphics

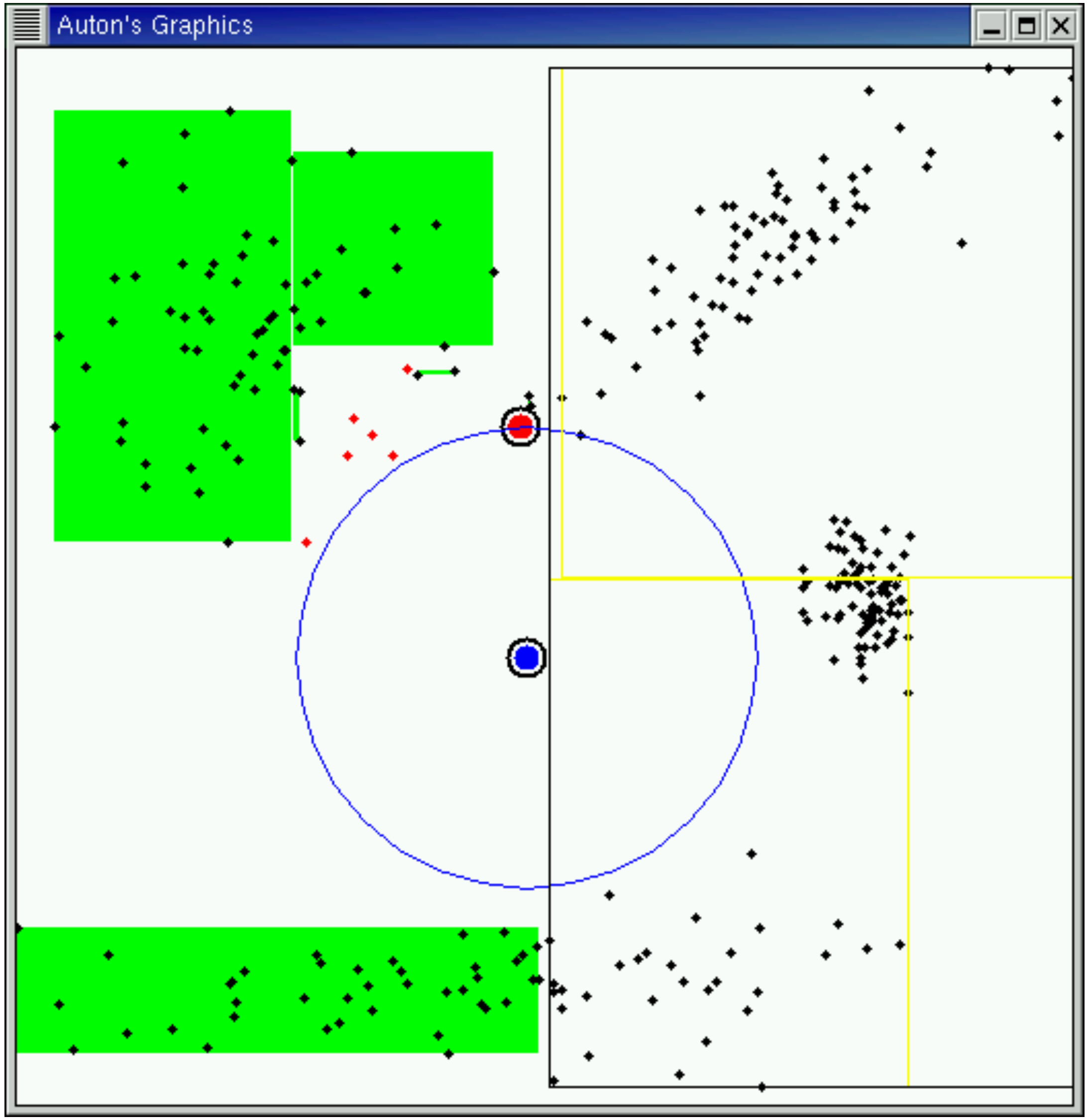

Auton's Graphics

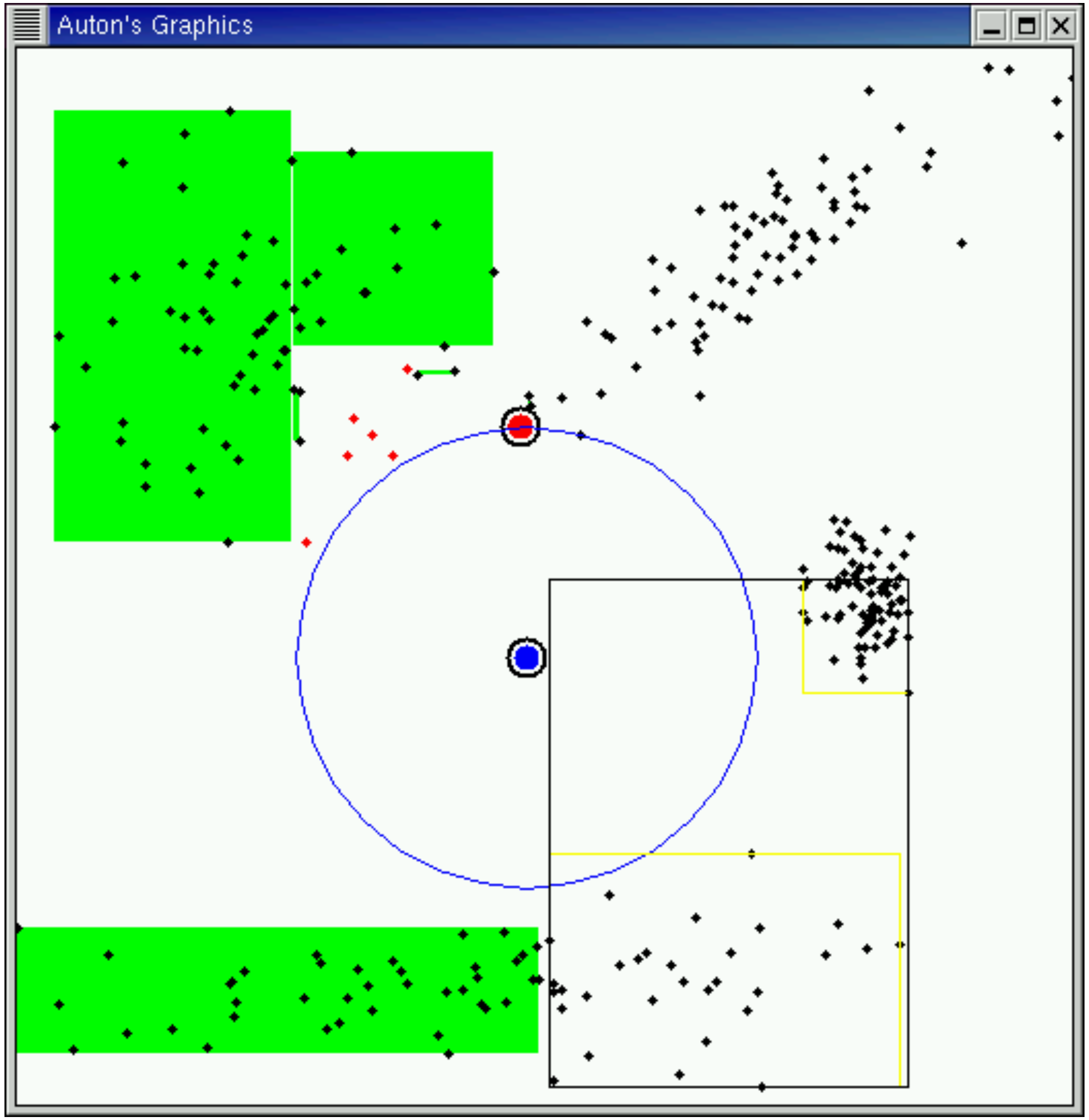

Auton's Graphics

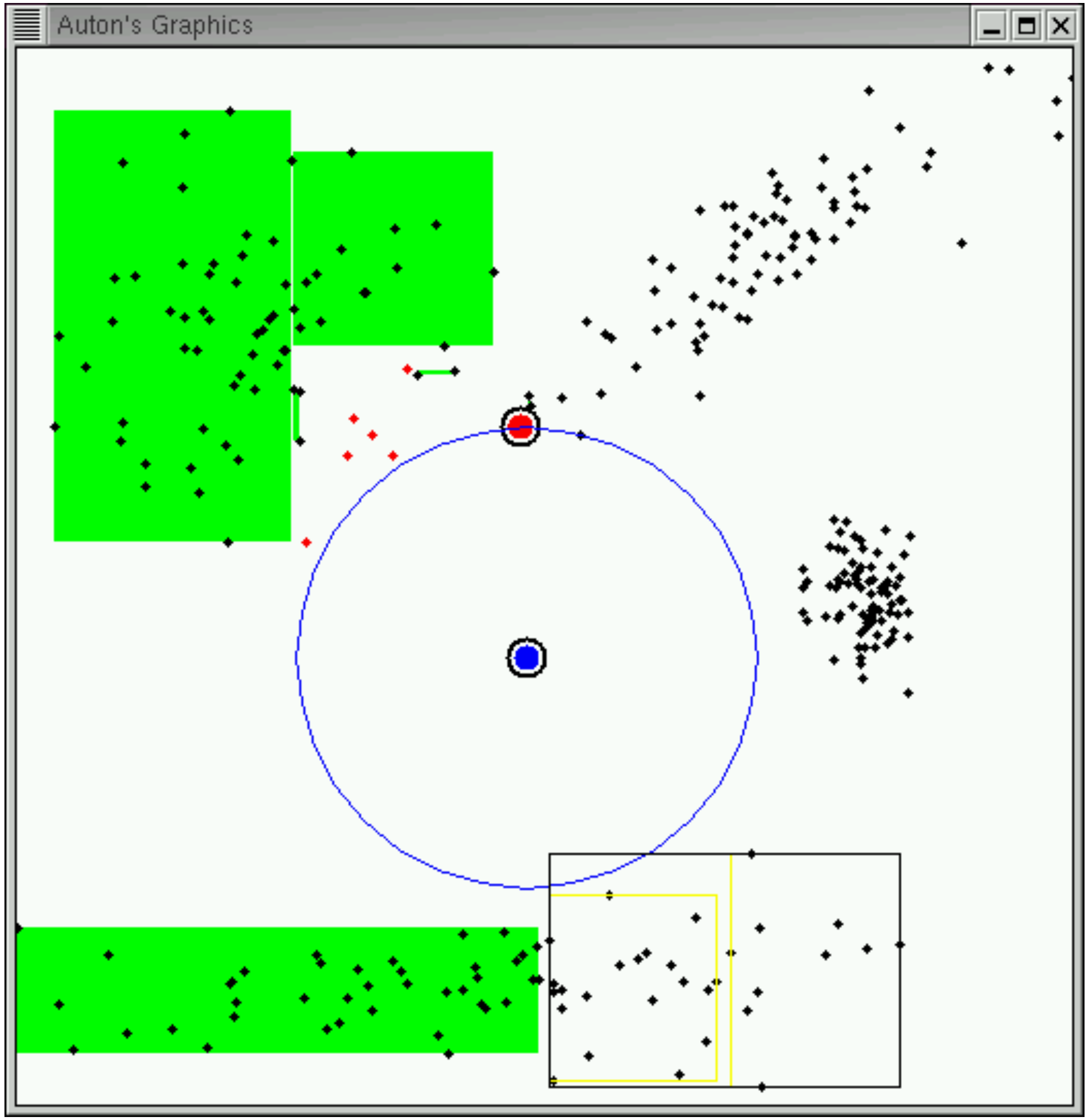

Auton's Graphics

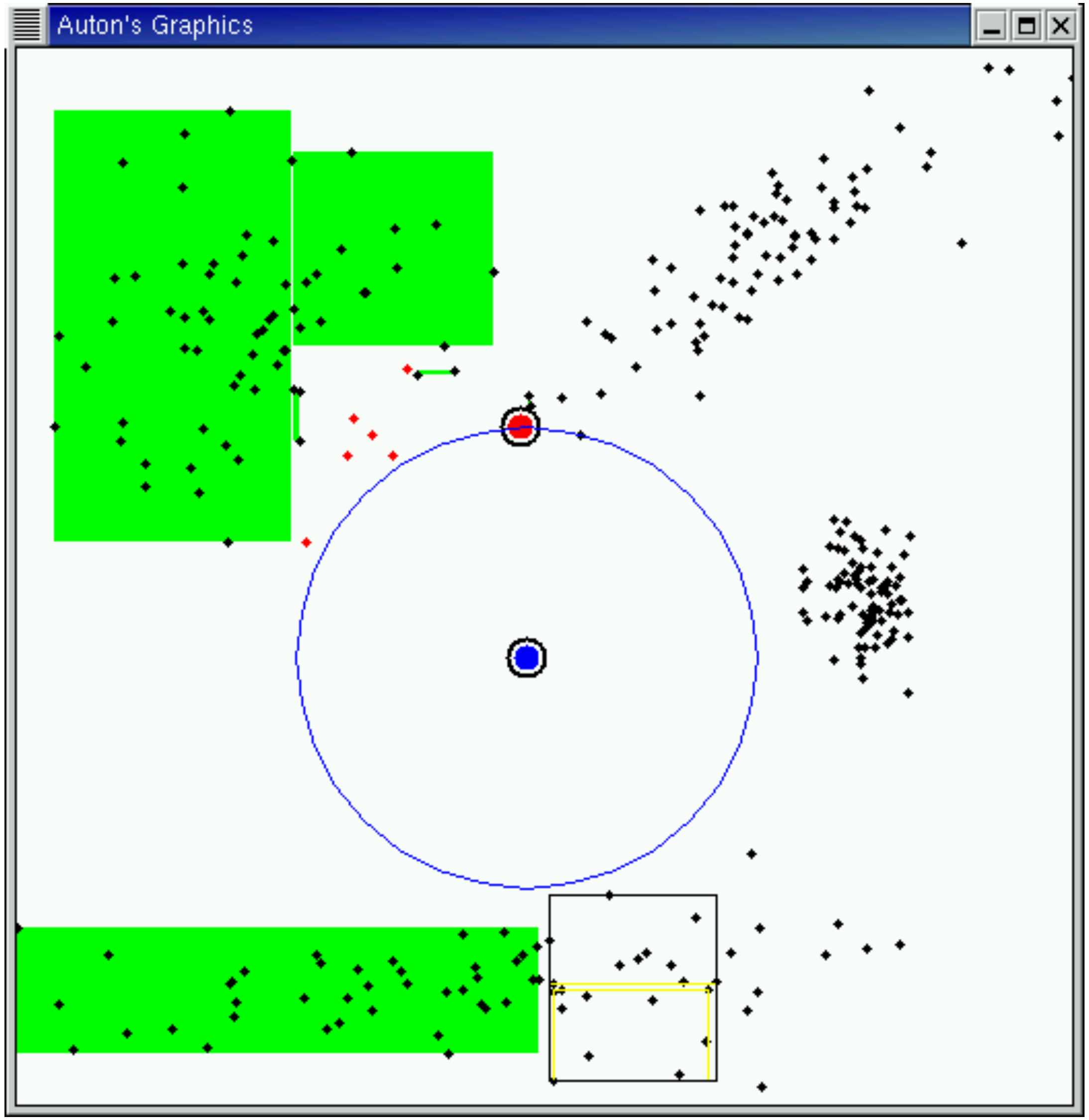

Auton's Graphics

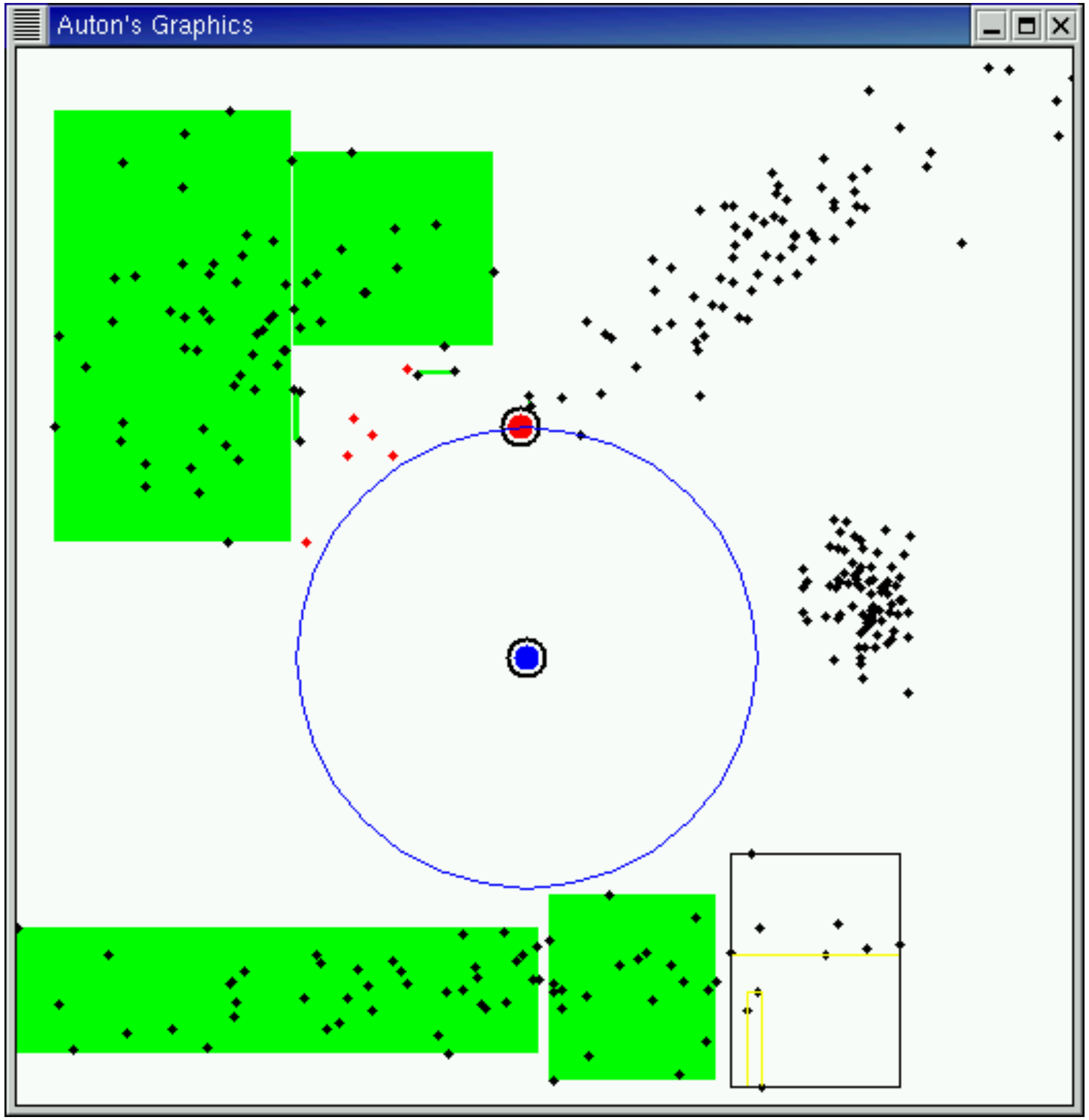

Auton's Graphics

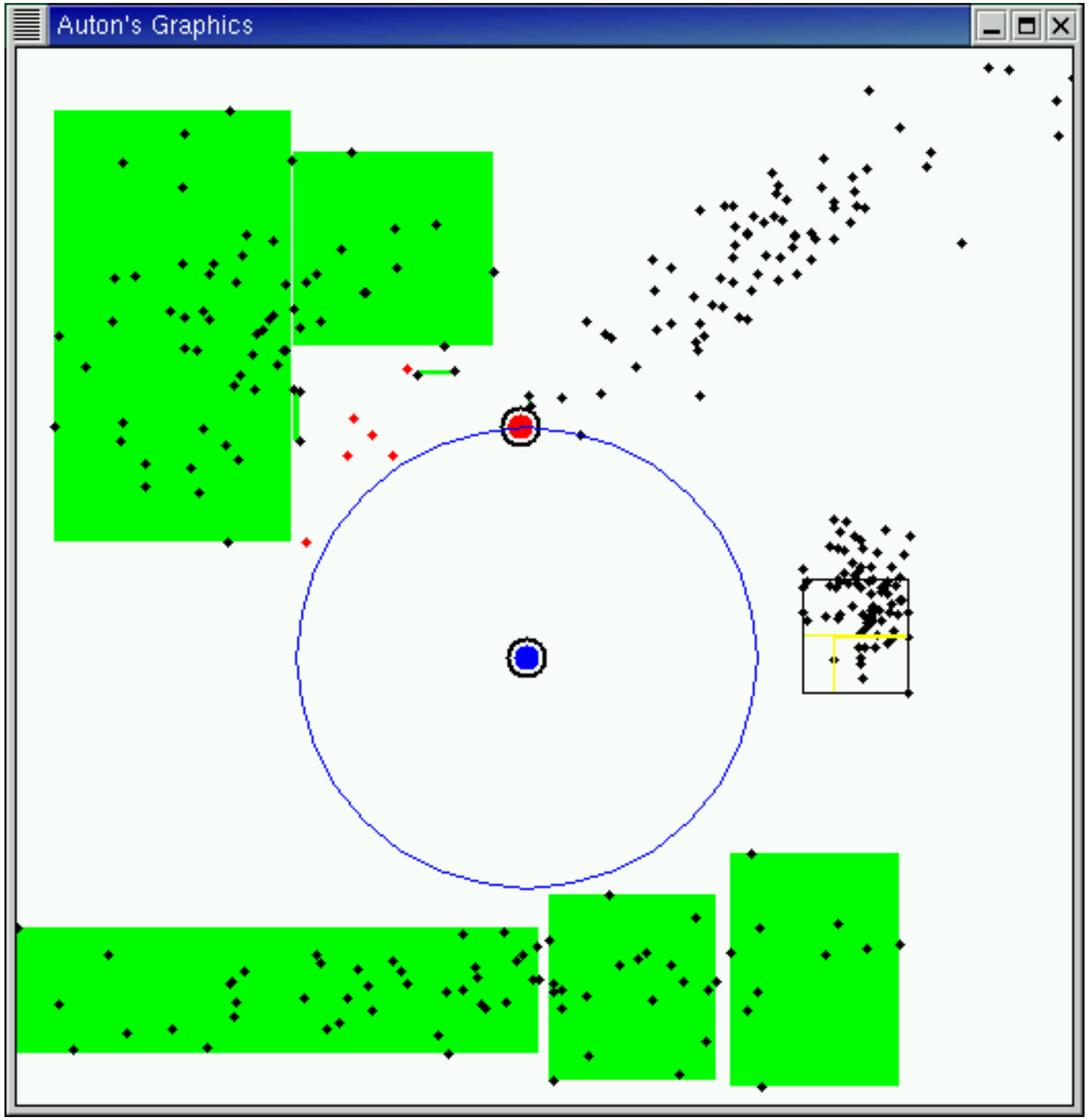

Auton's Graphics

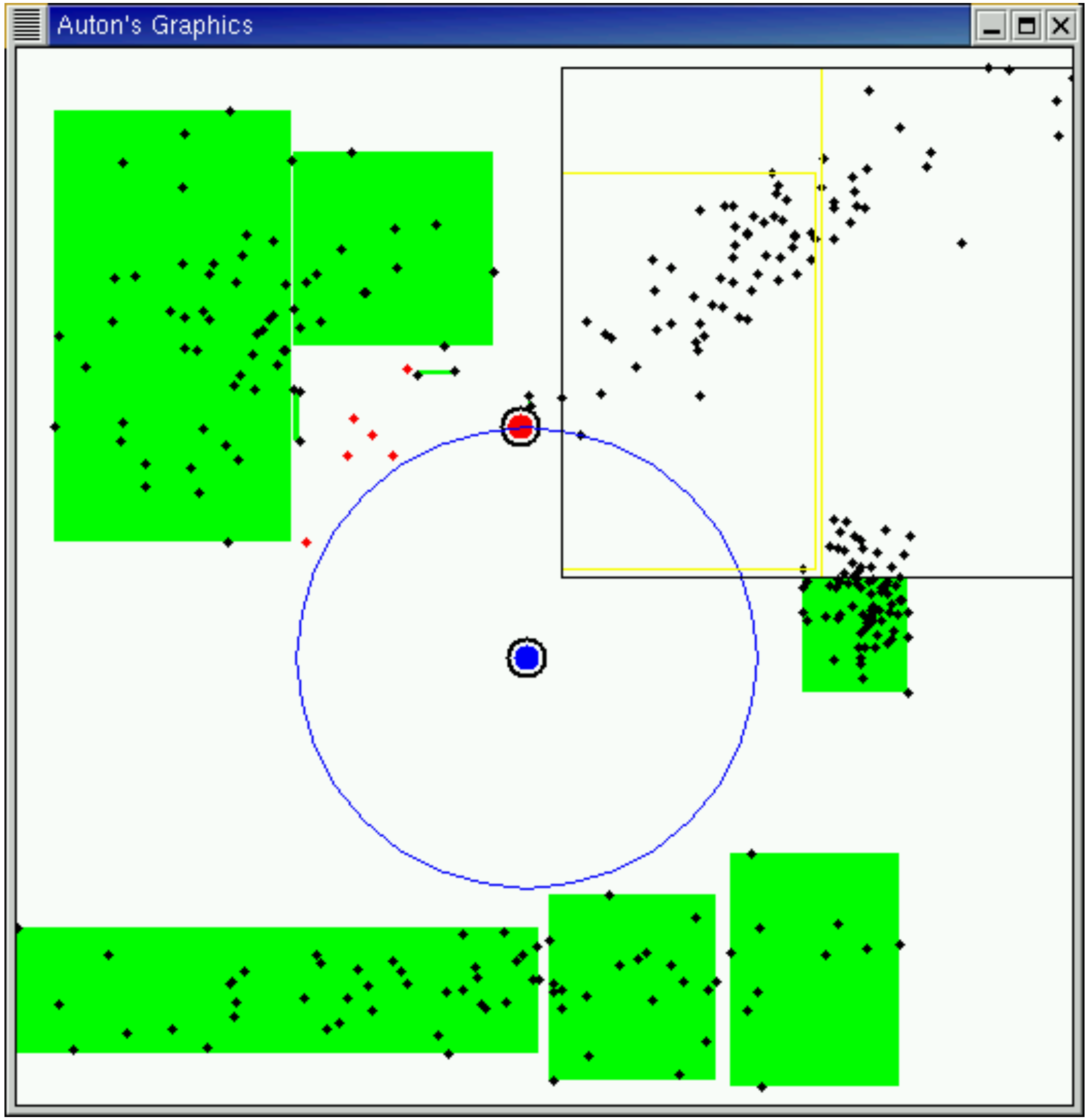

Auton's Graphics

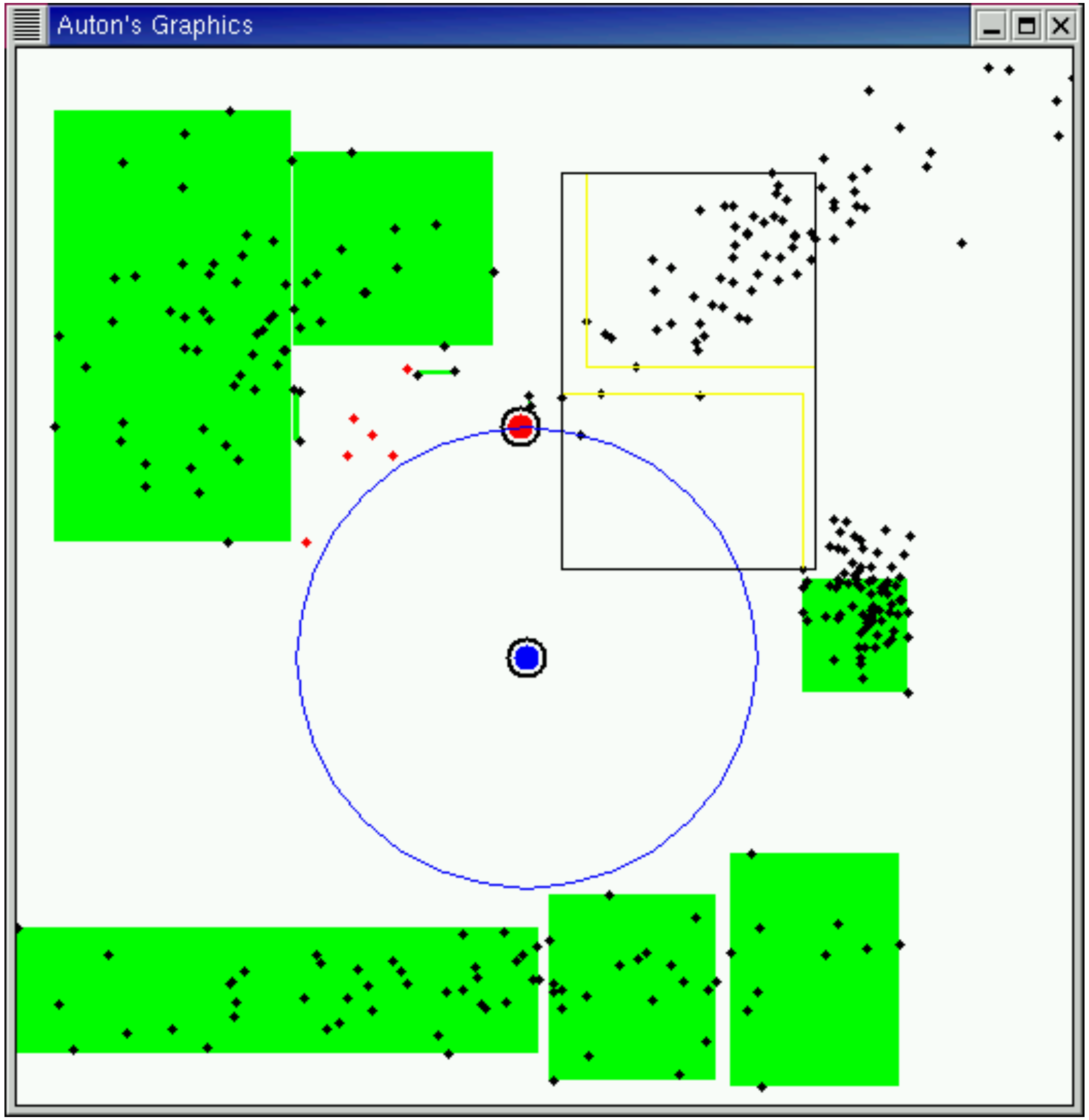

Auton's Graphics

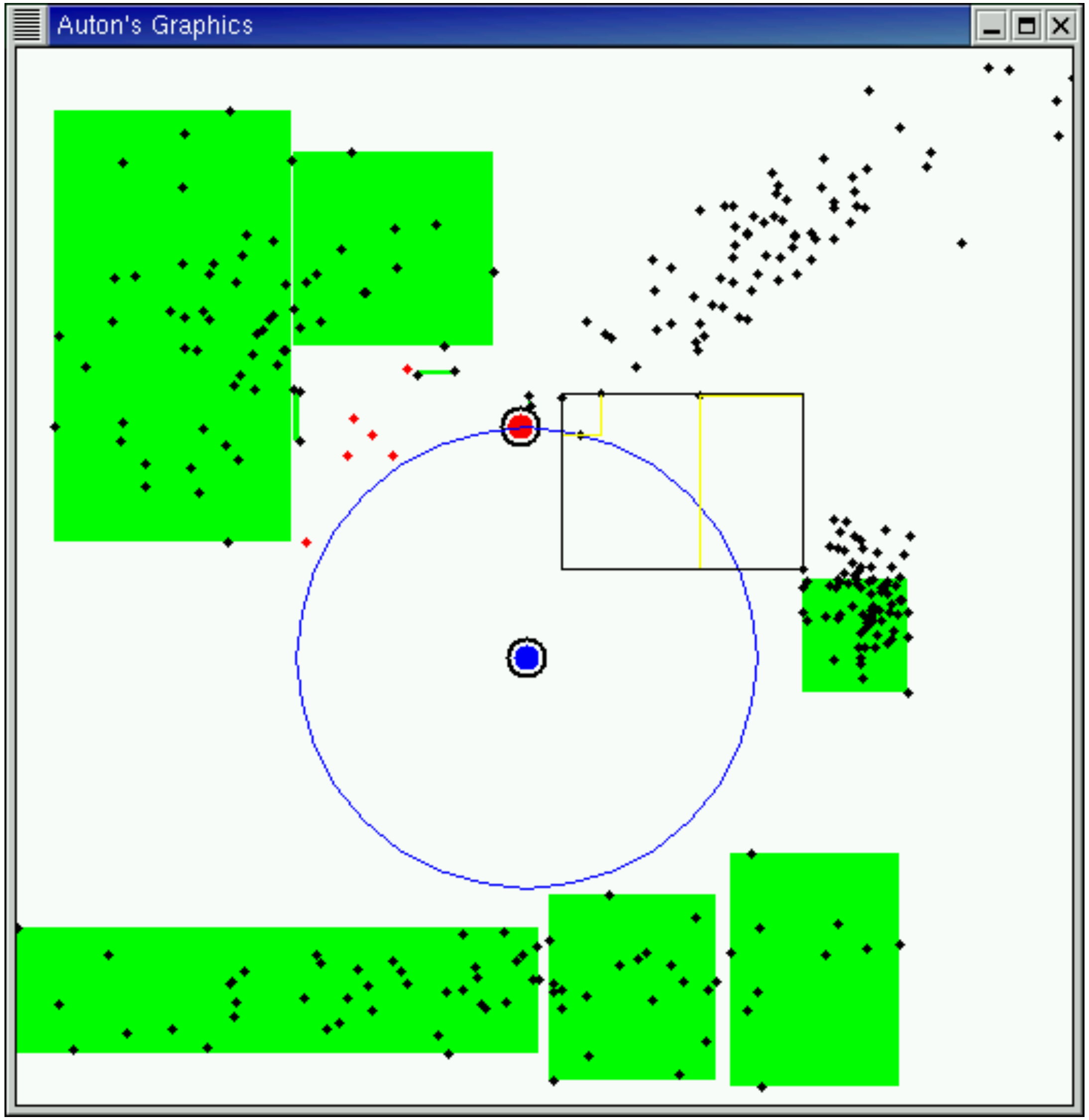

Auton's Graphics

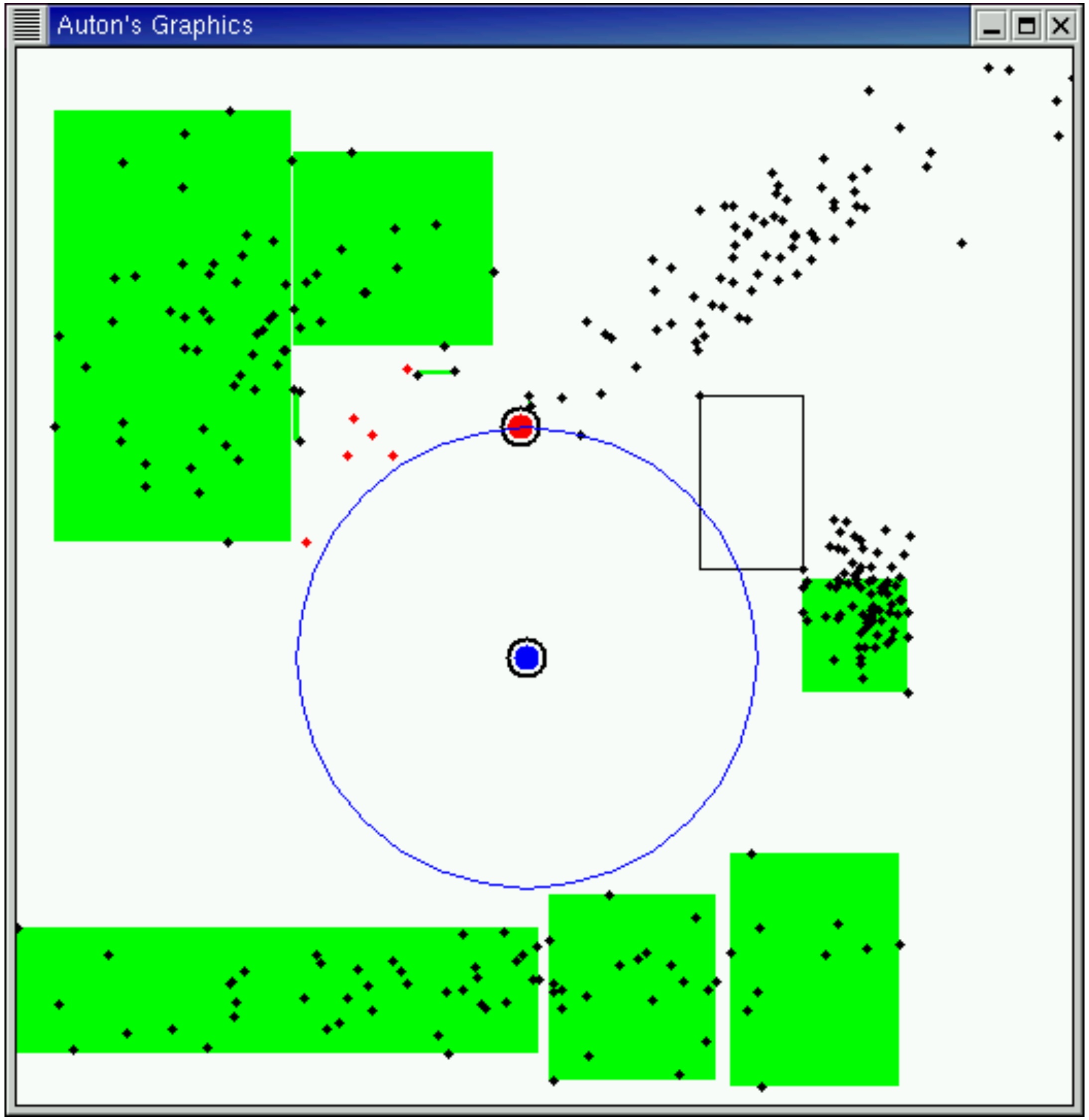

Auton's Graphics

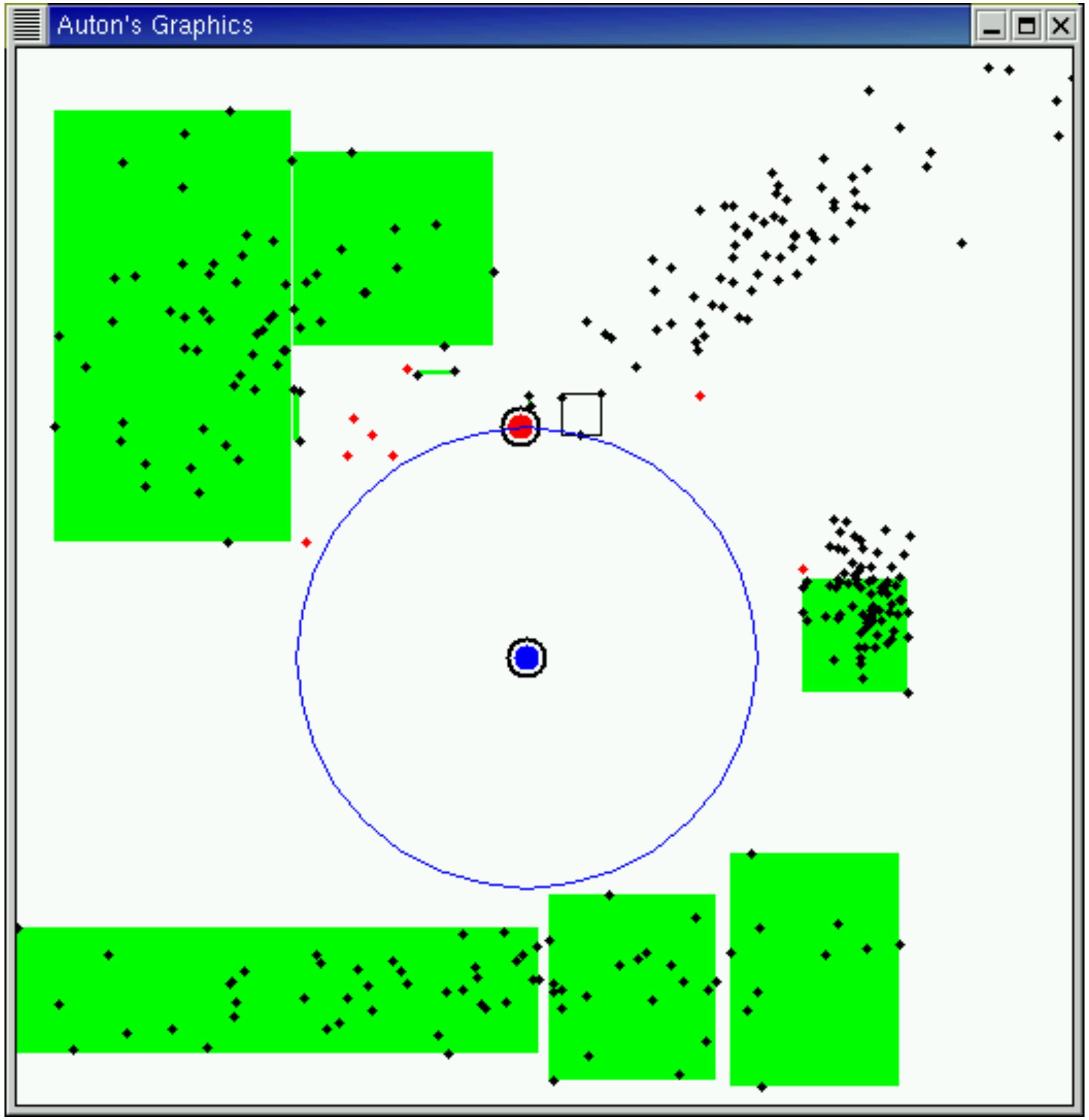

Auton's Graphics

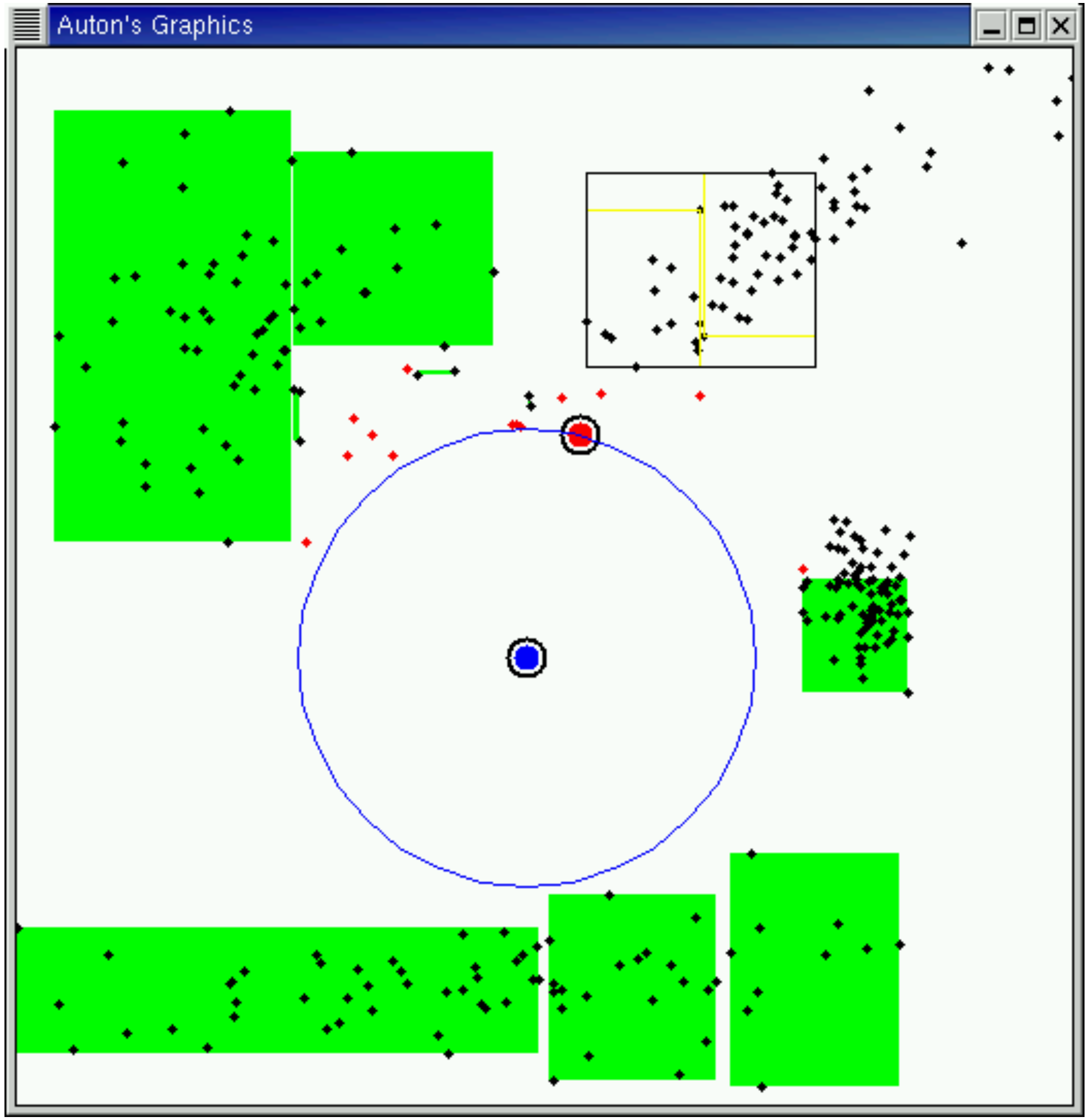

Auton's Graphics

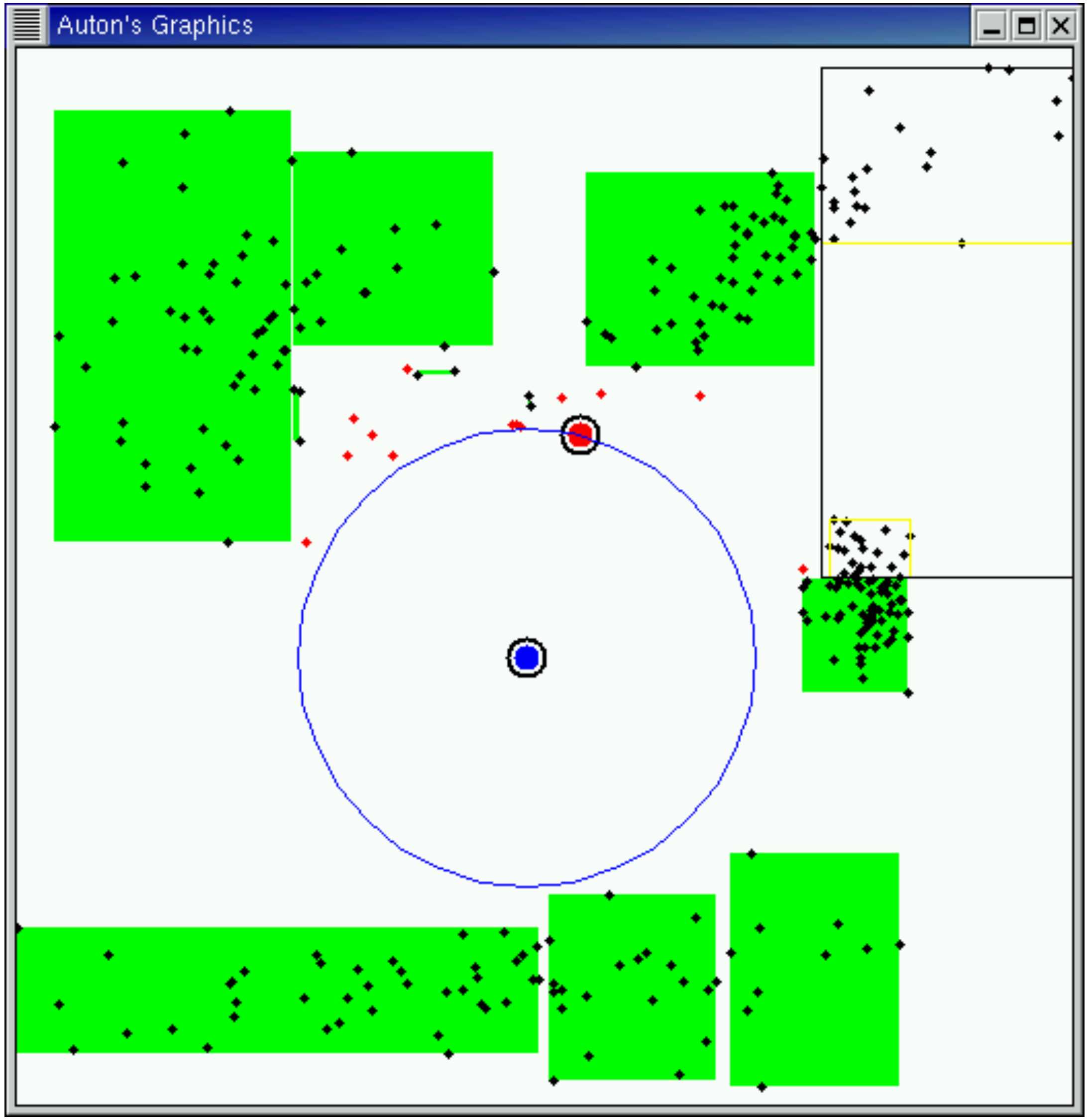

Auton's Graphics

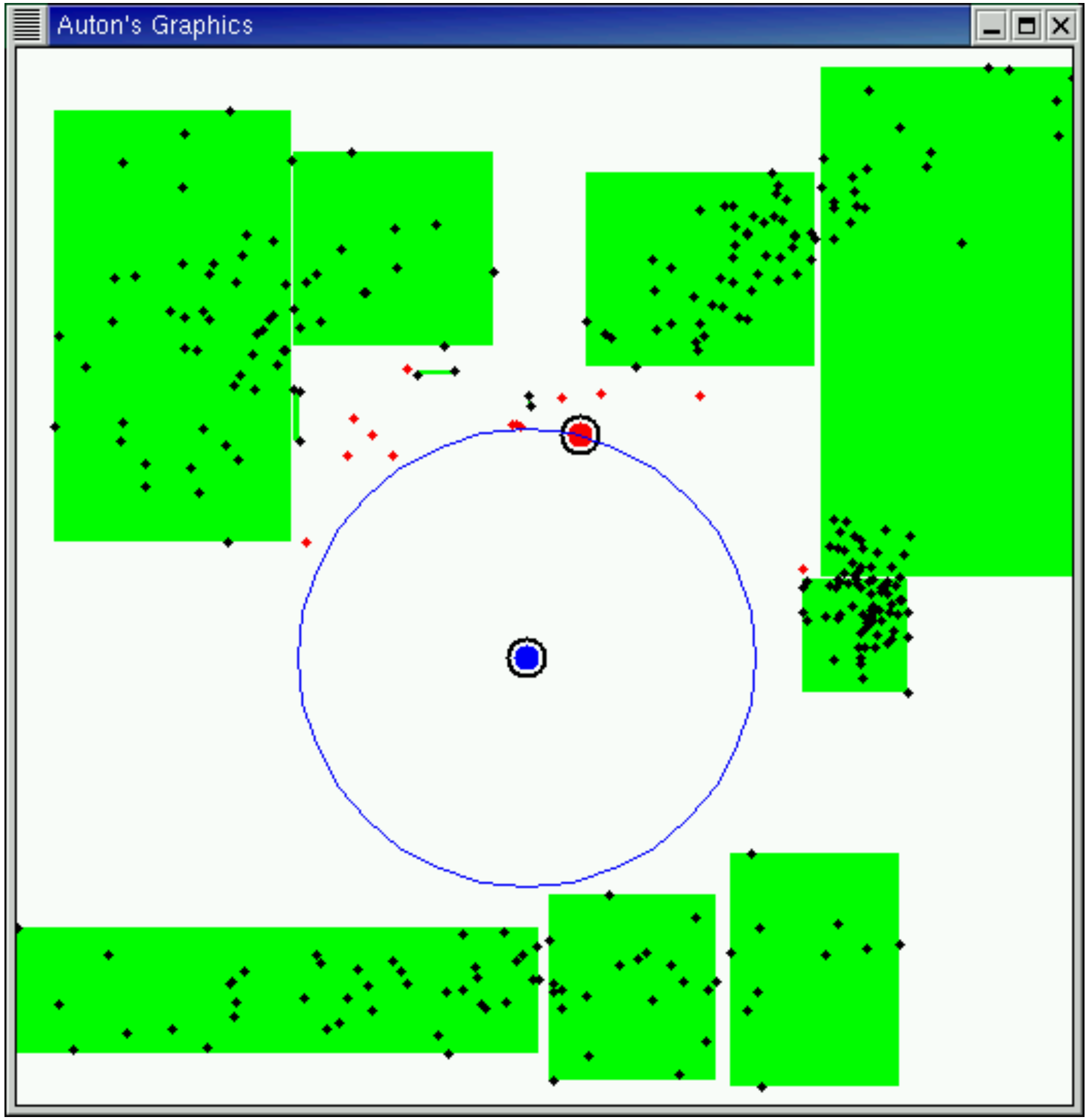

Auton's Graphics


Auton's Graphics

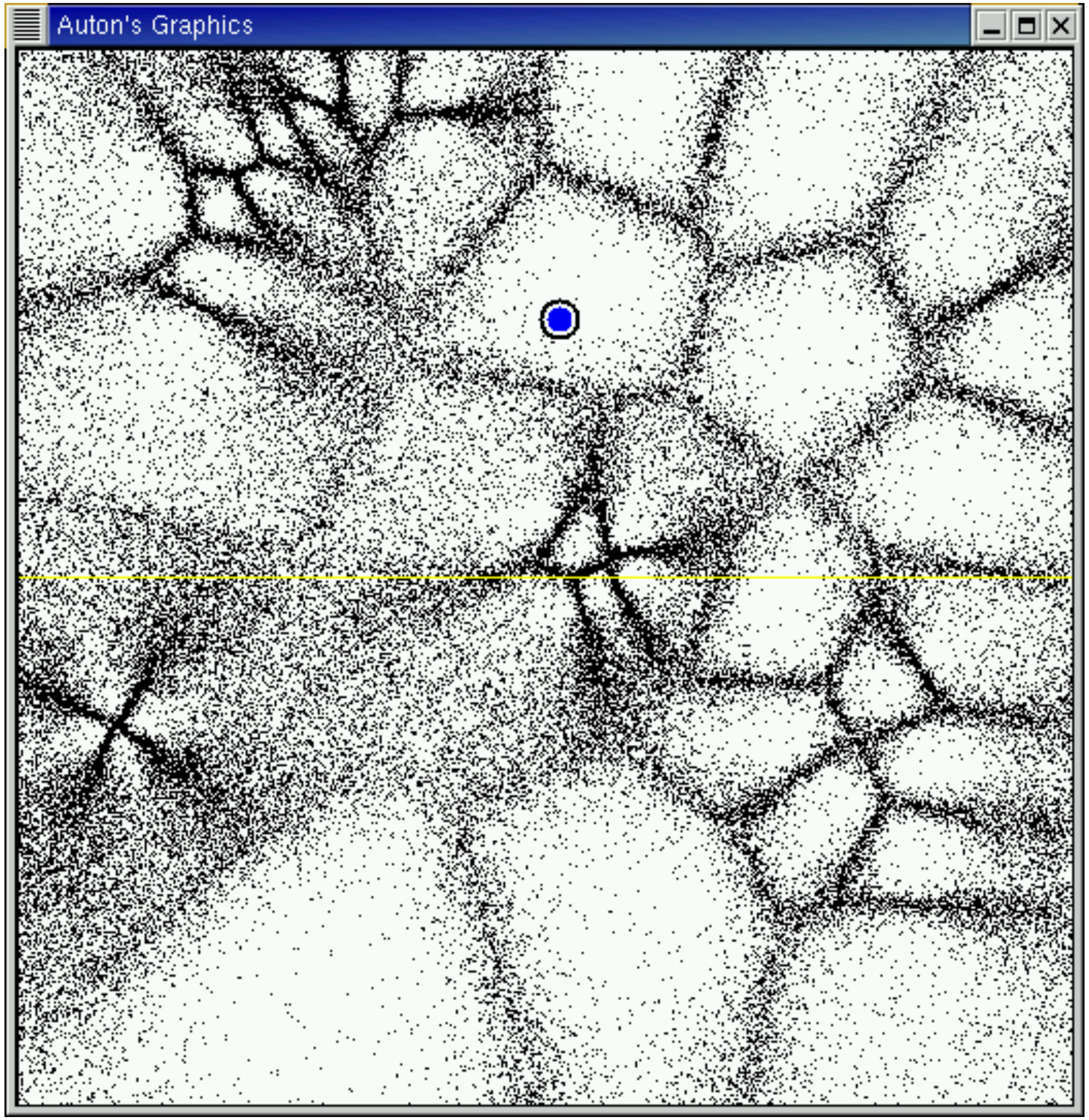

Auton's Graphics

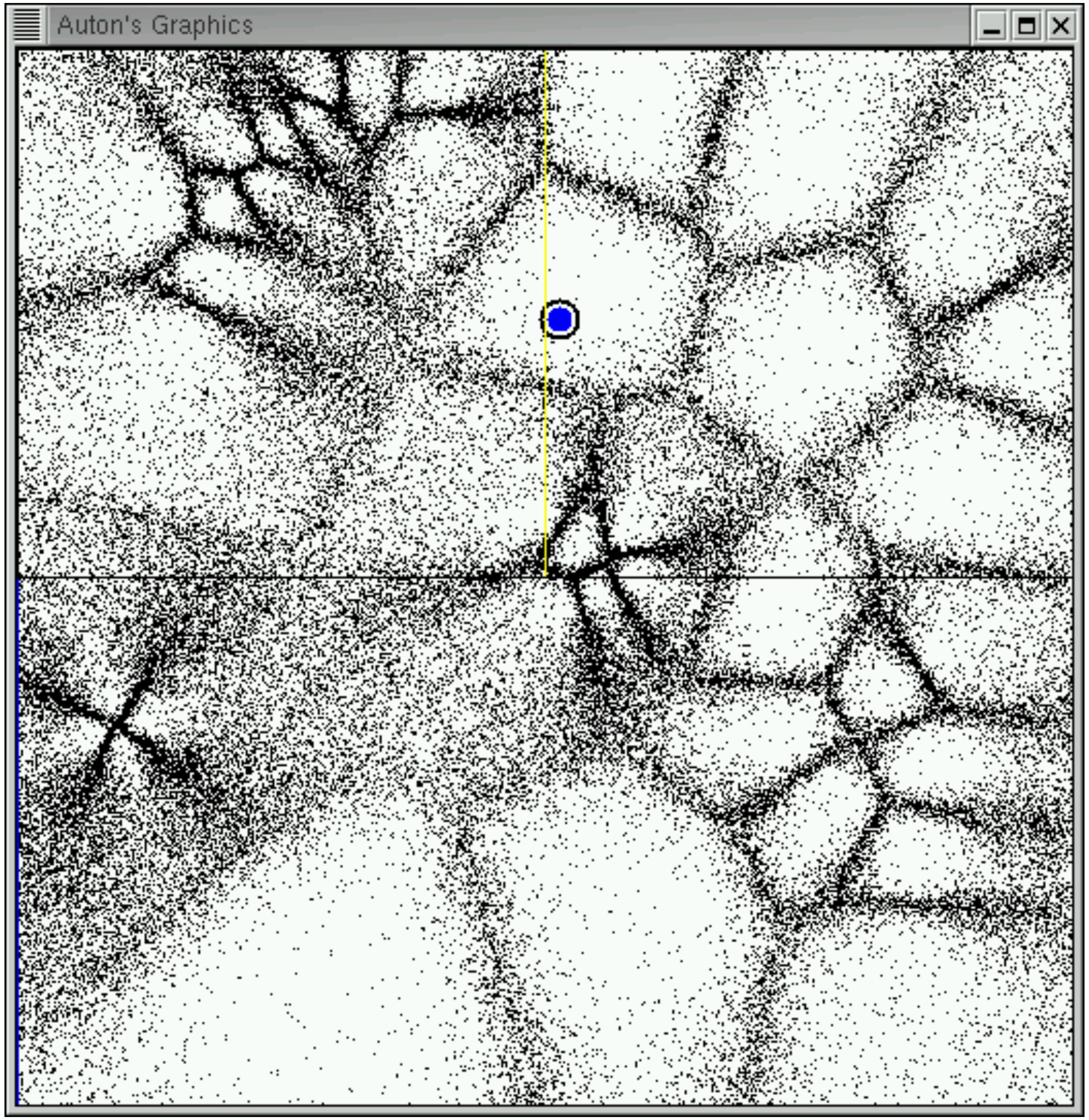

Auton's Graphics

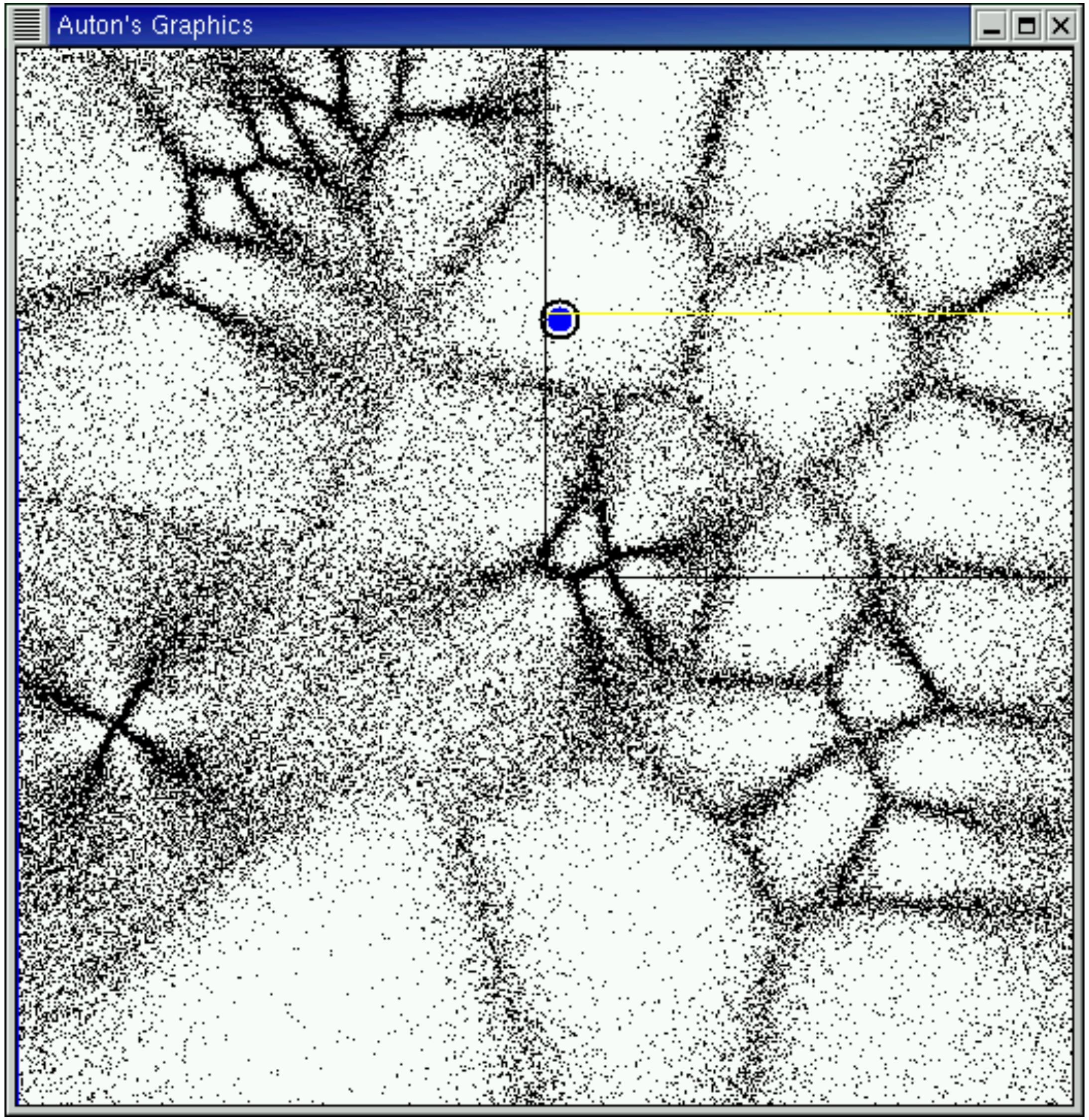

Auton's Graphics

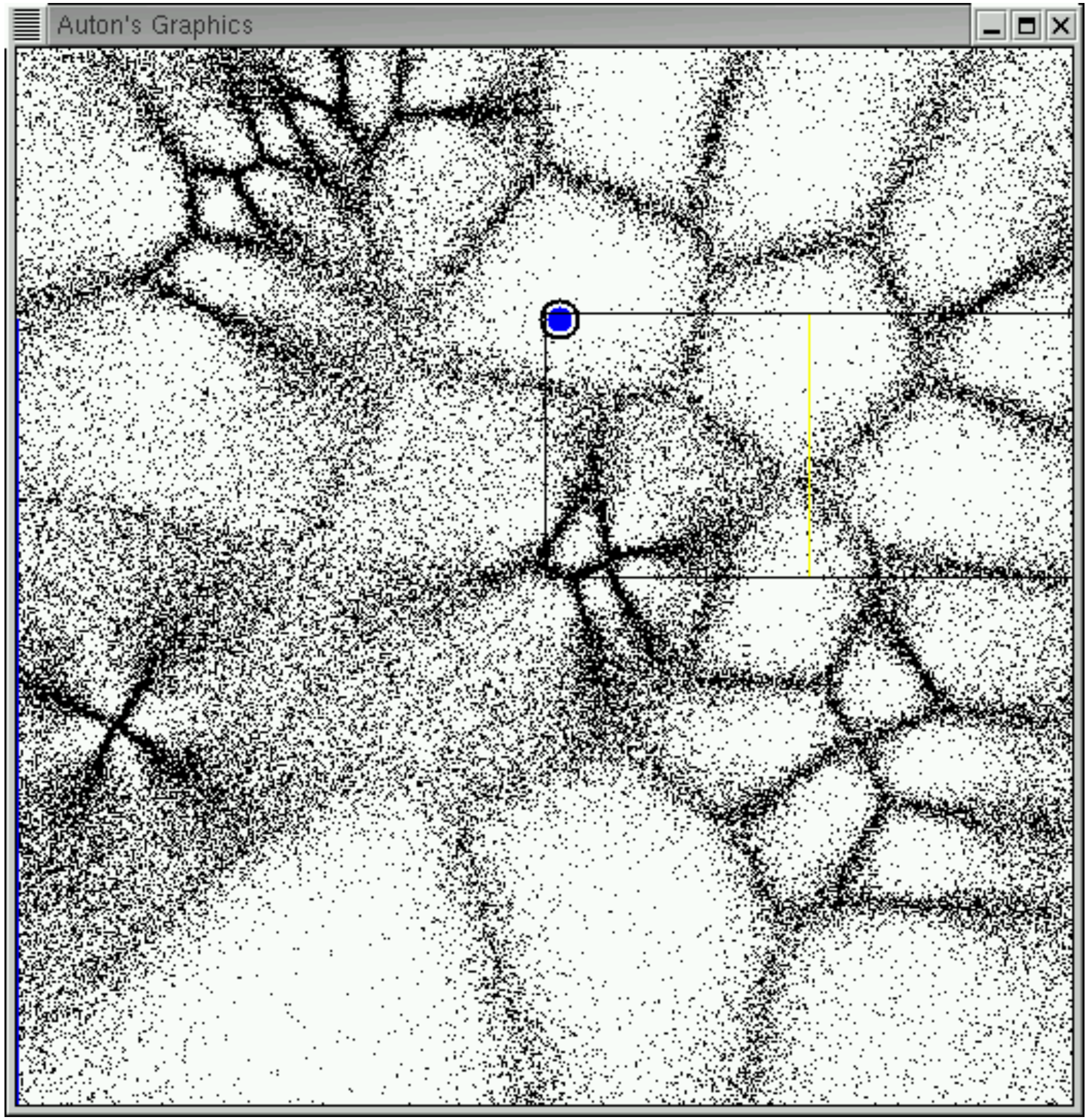

Auton's Graphics

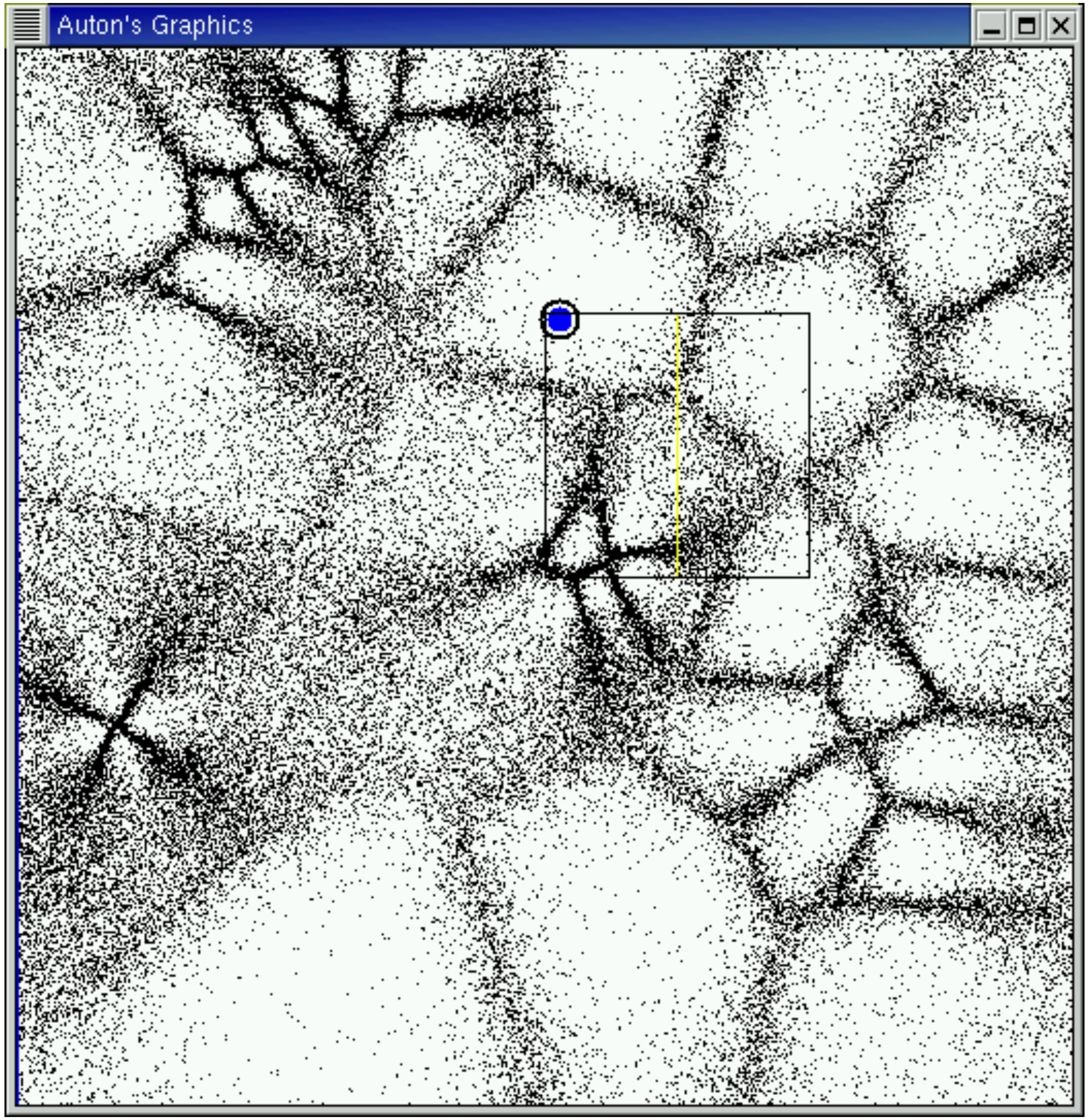
Animation of NN search with large data set

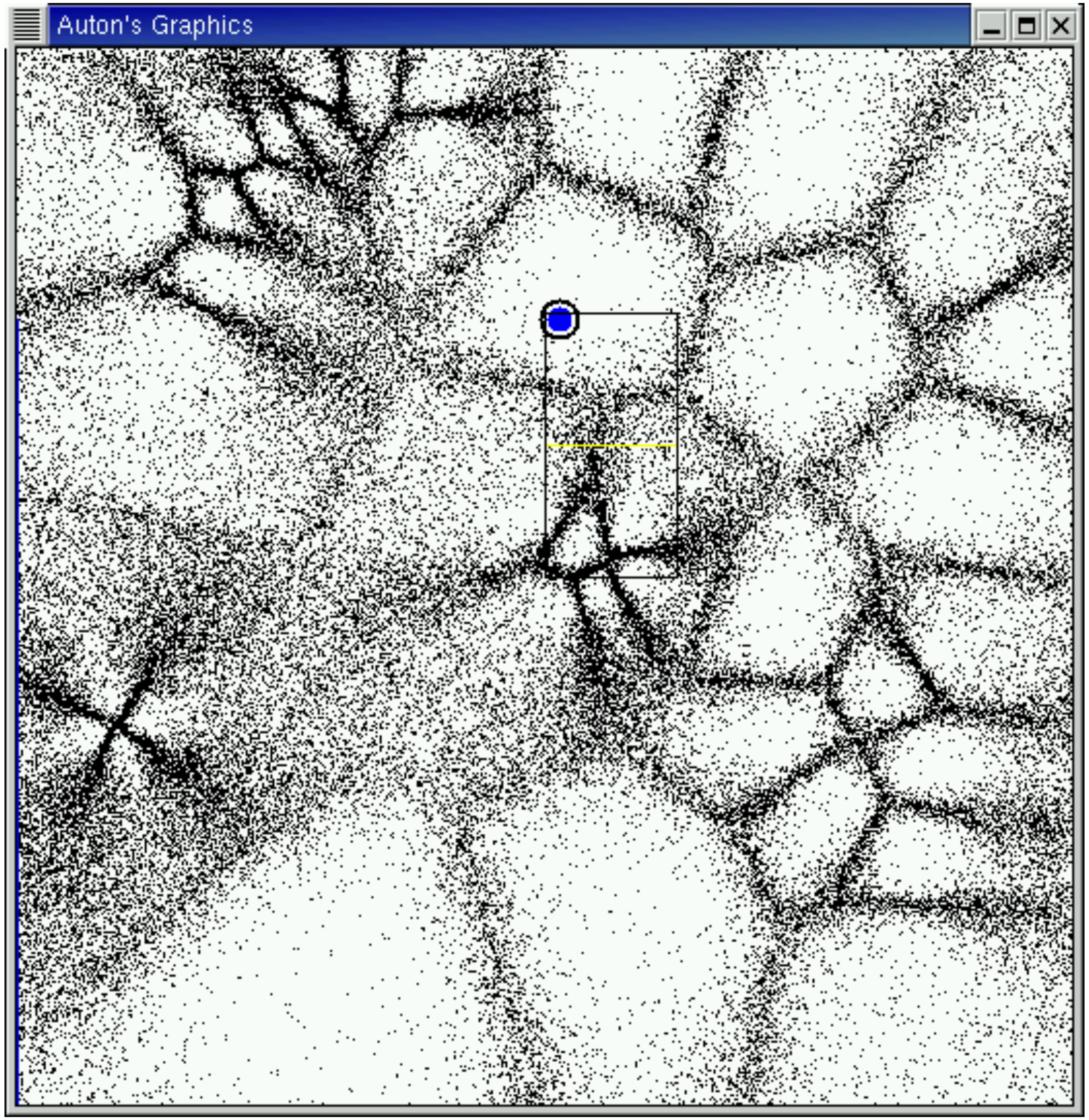


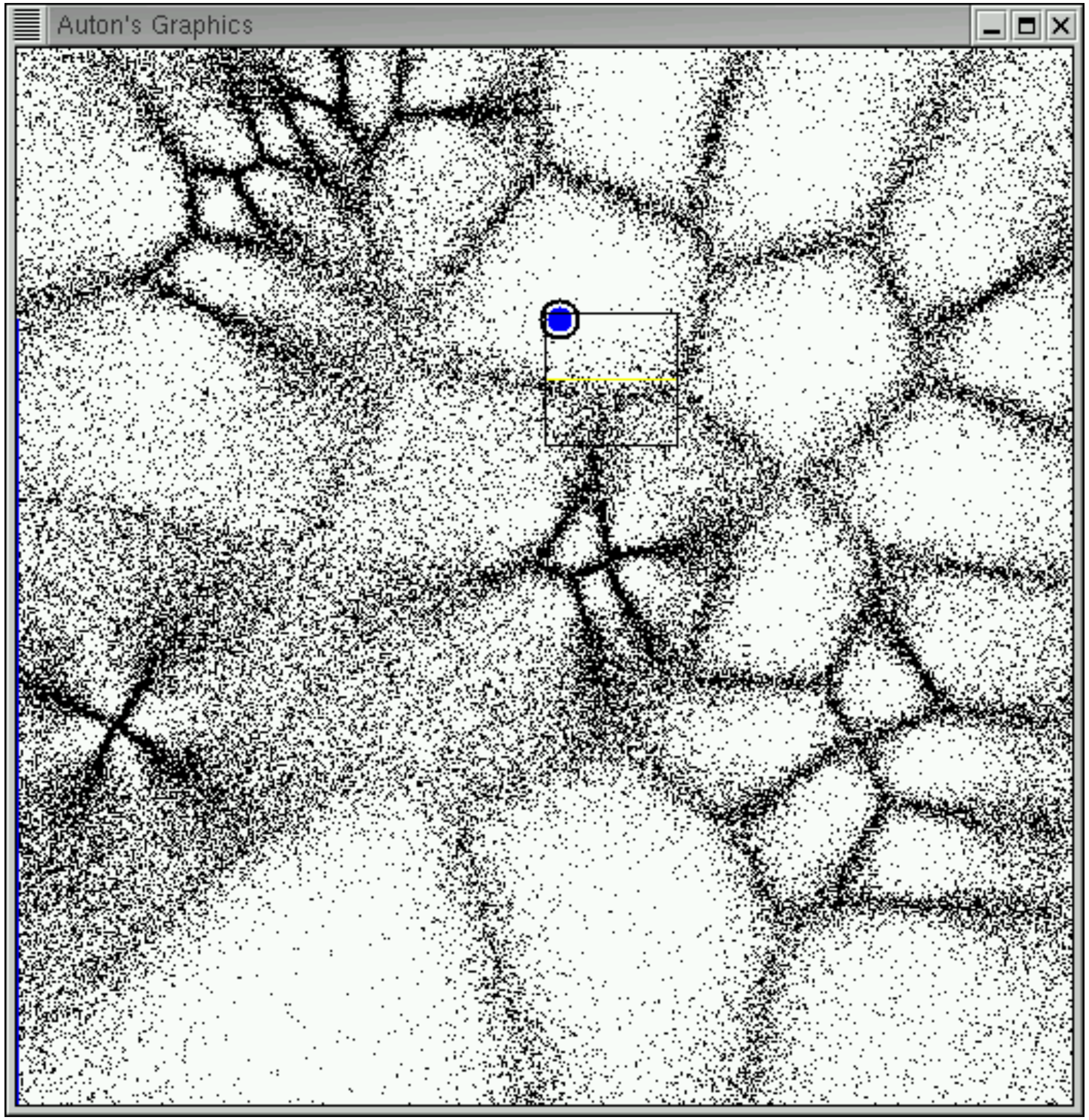

Auton's Graphics

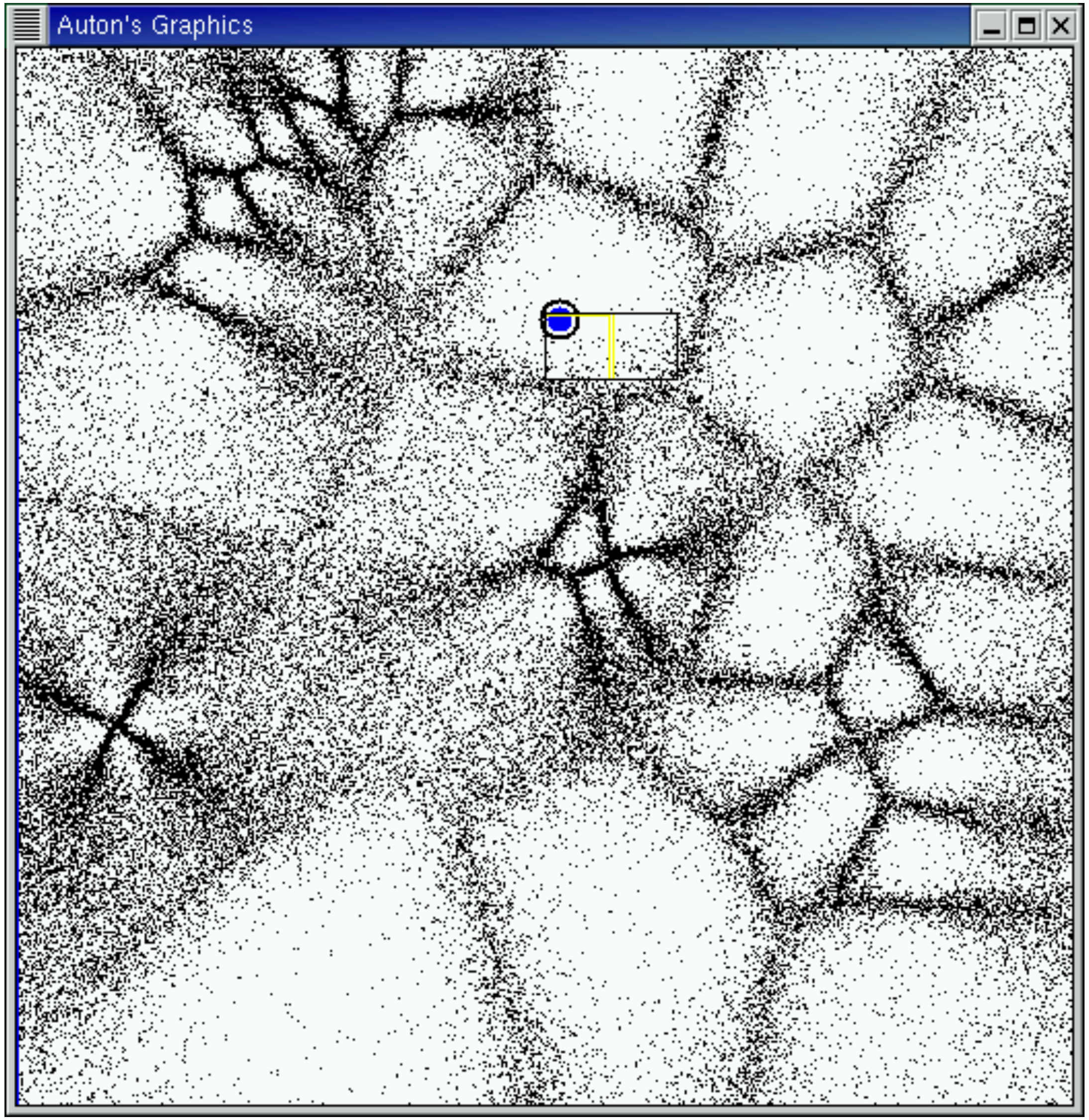

Auton's Graphics

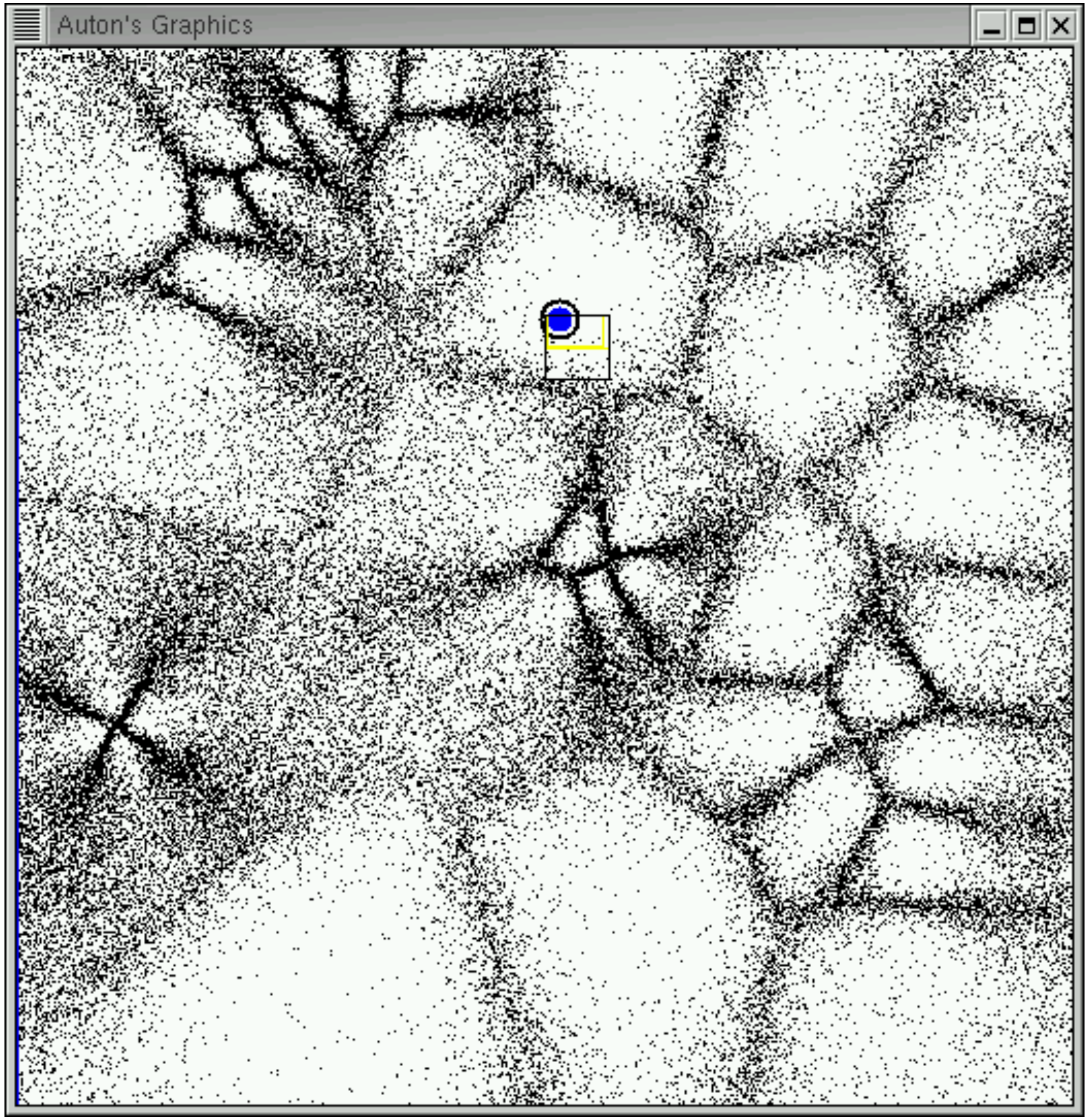

Auton's Graphics

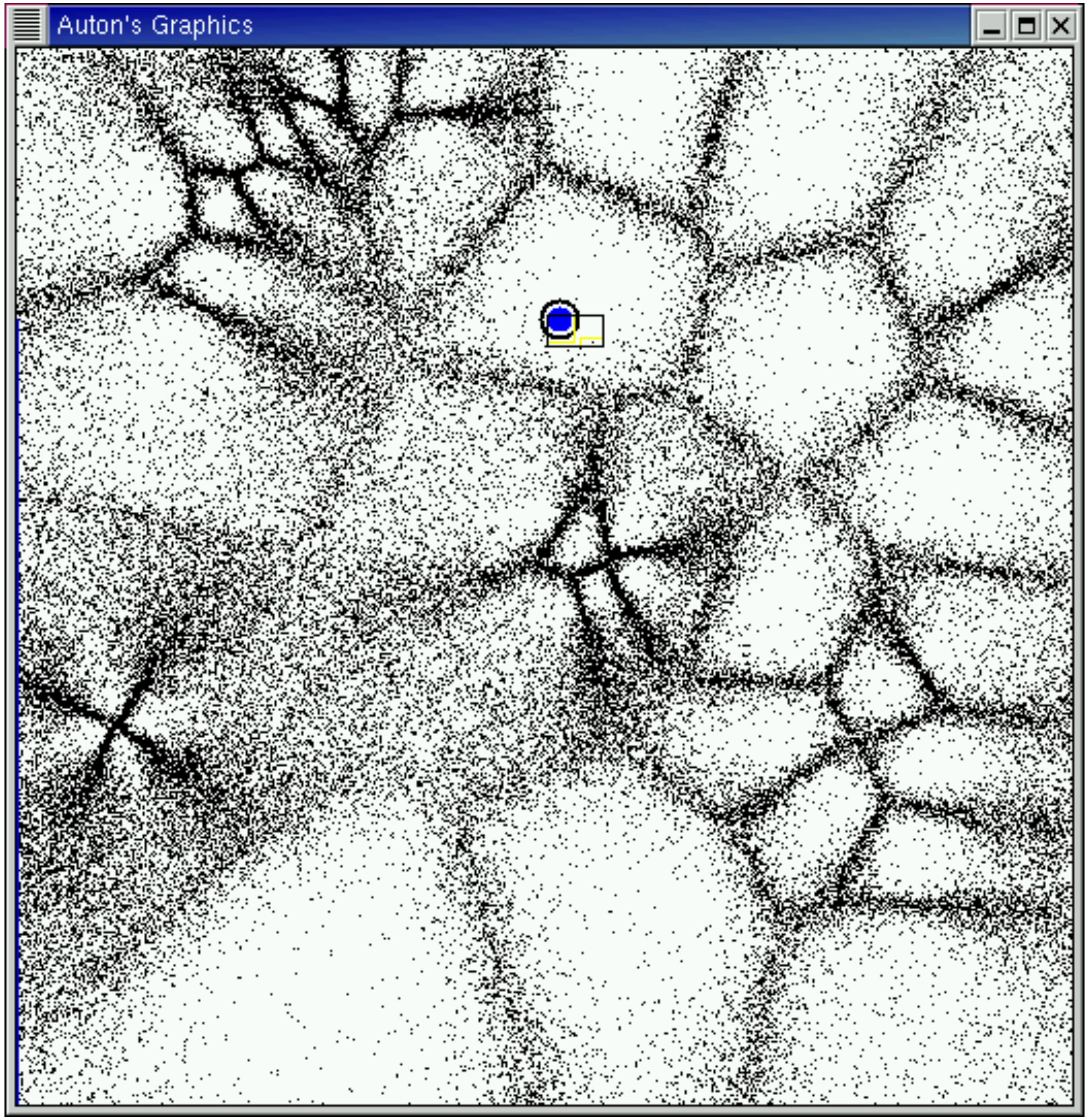

Auton's Graphics

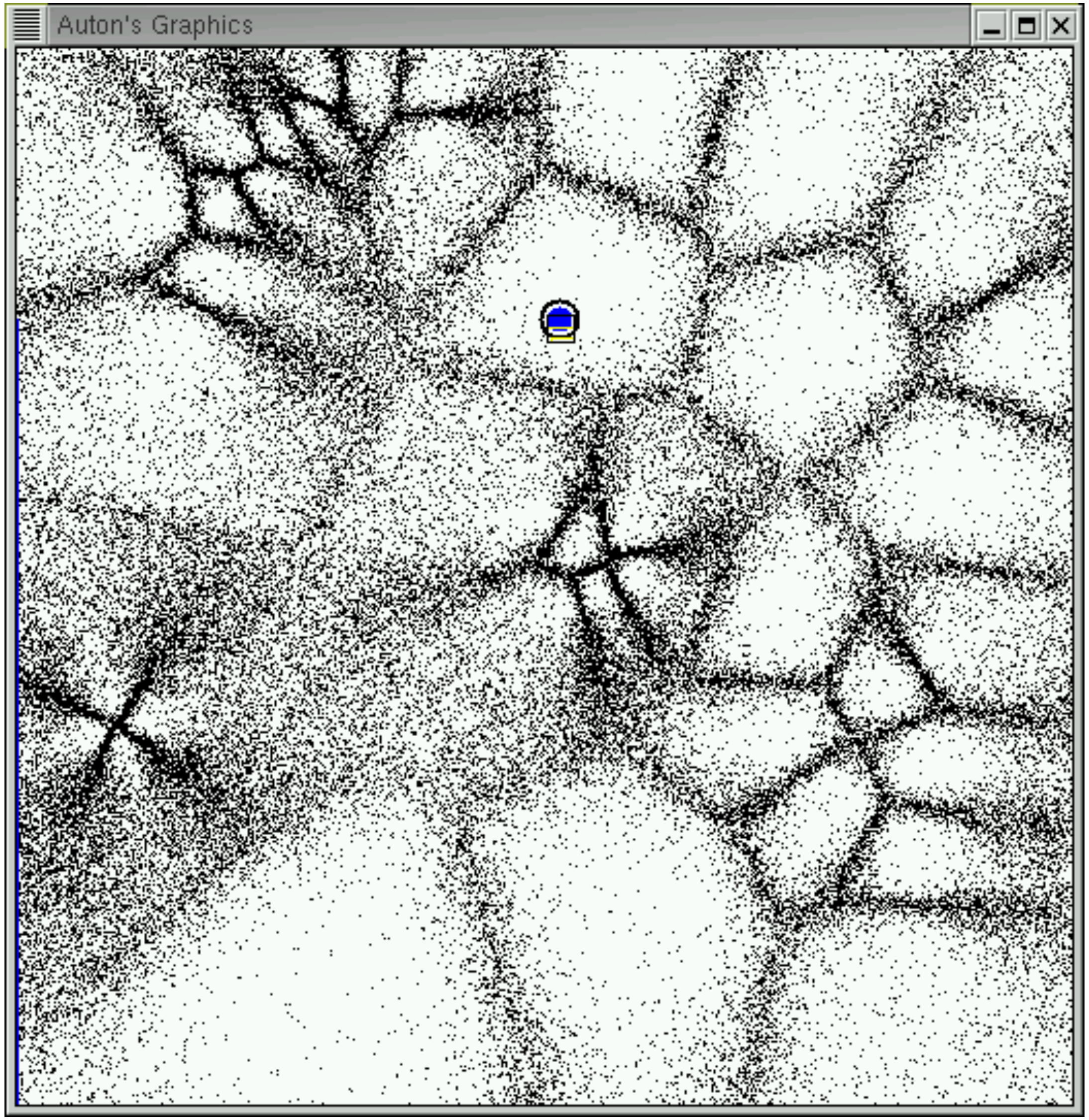

Auton's Graphics

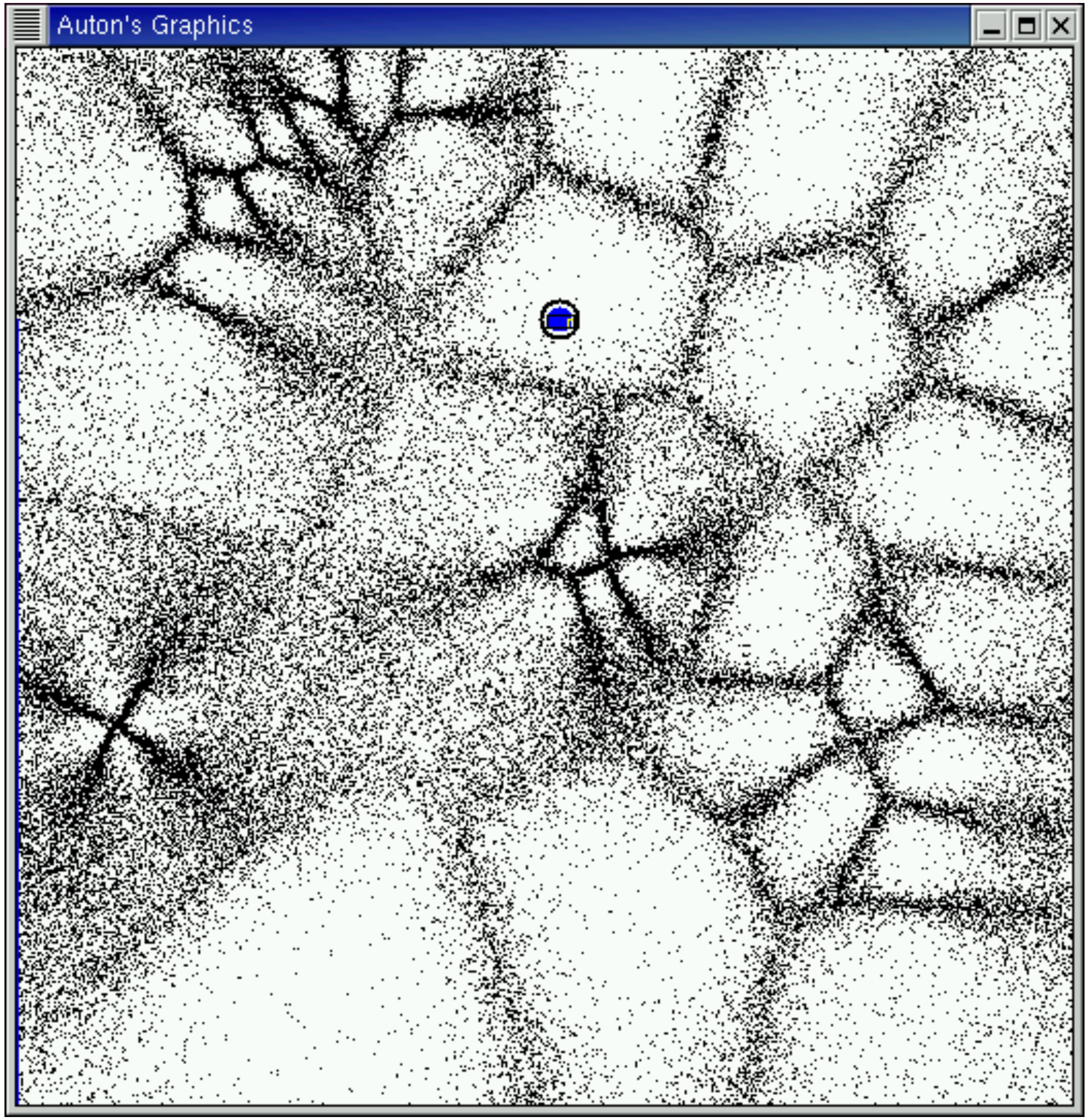

Auton's Graphics

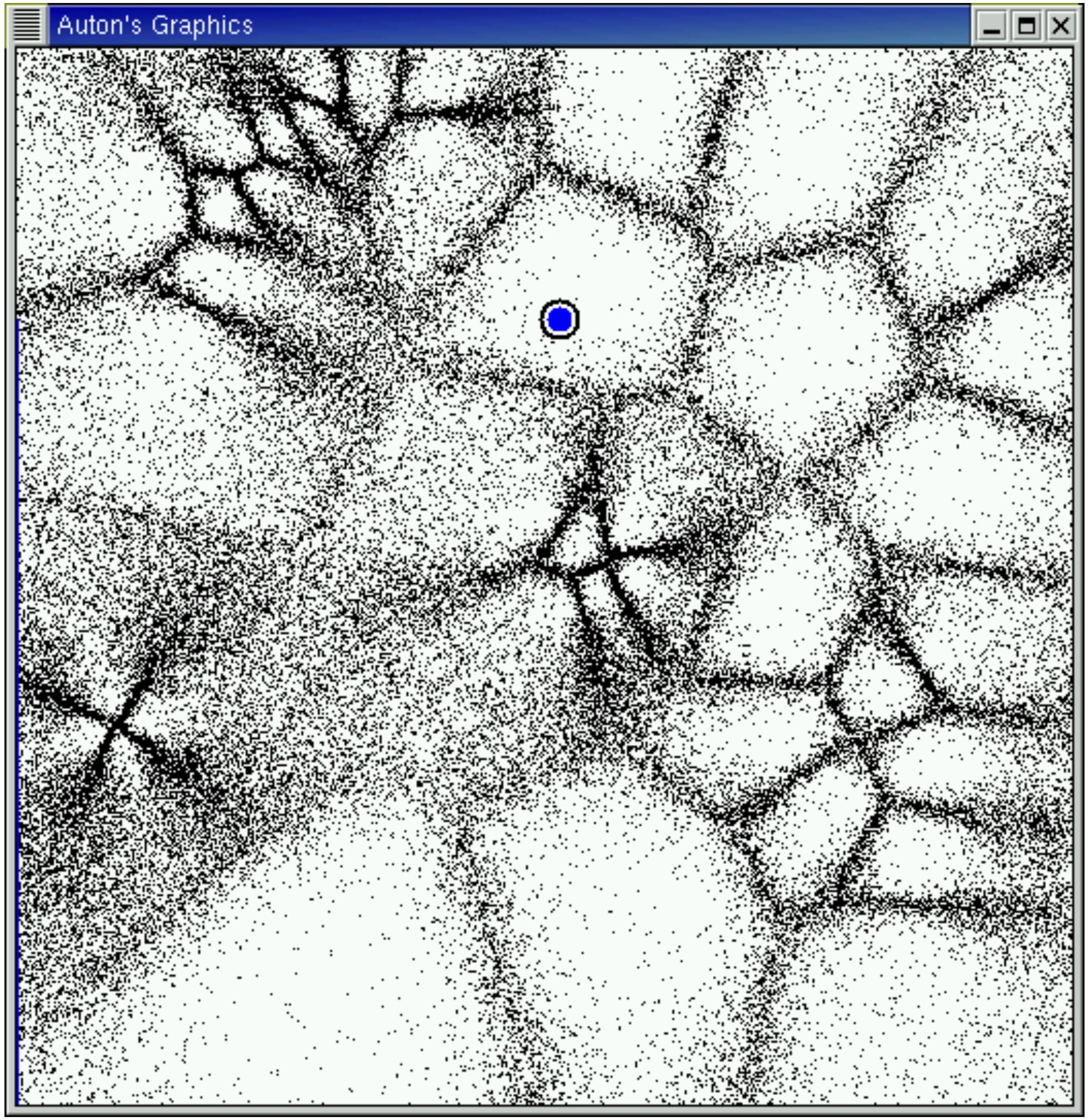

Auton's Graphics

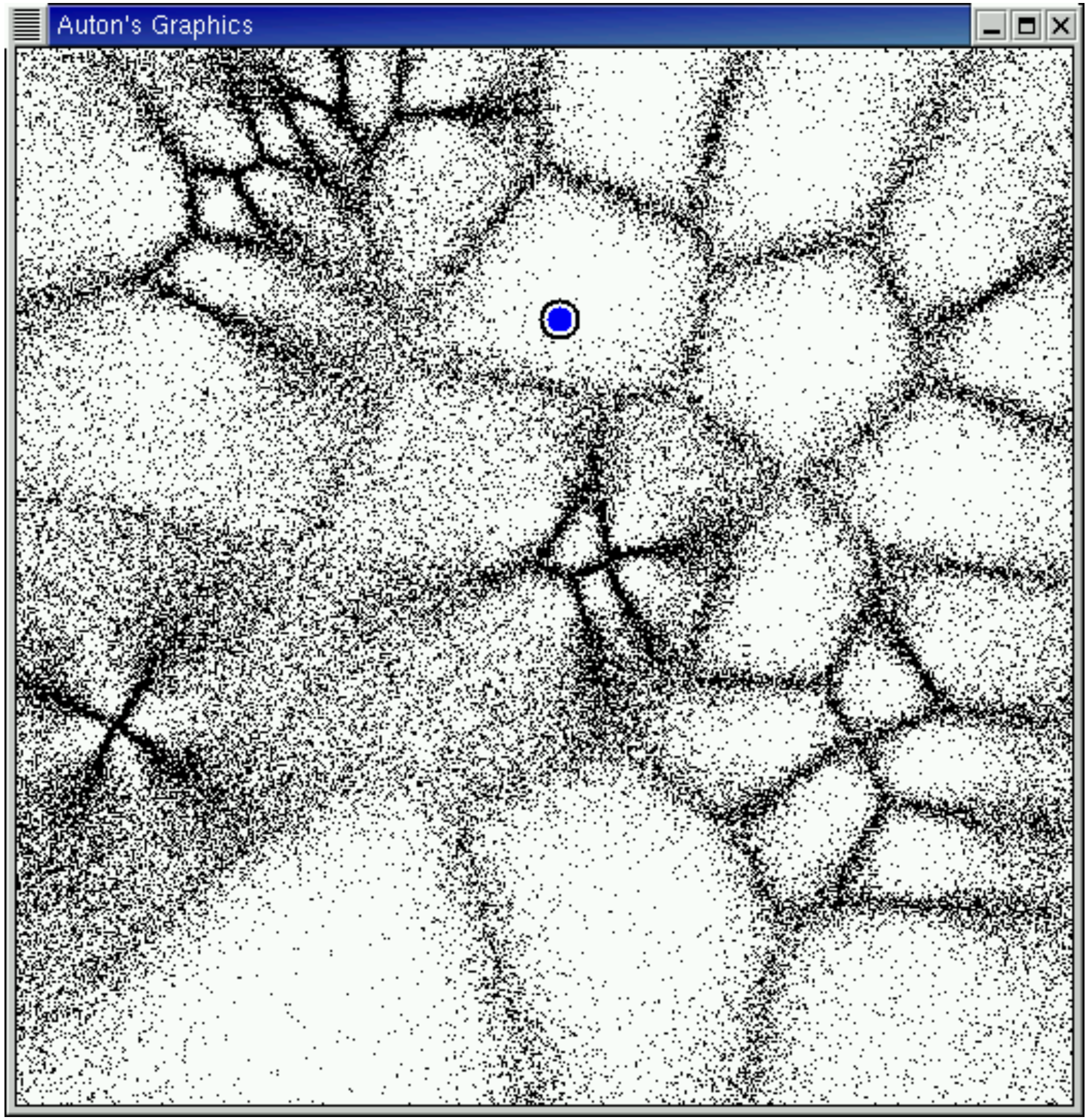

Auton's Graphics

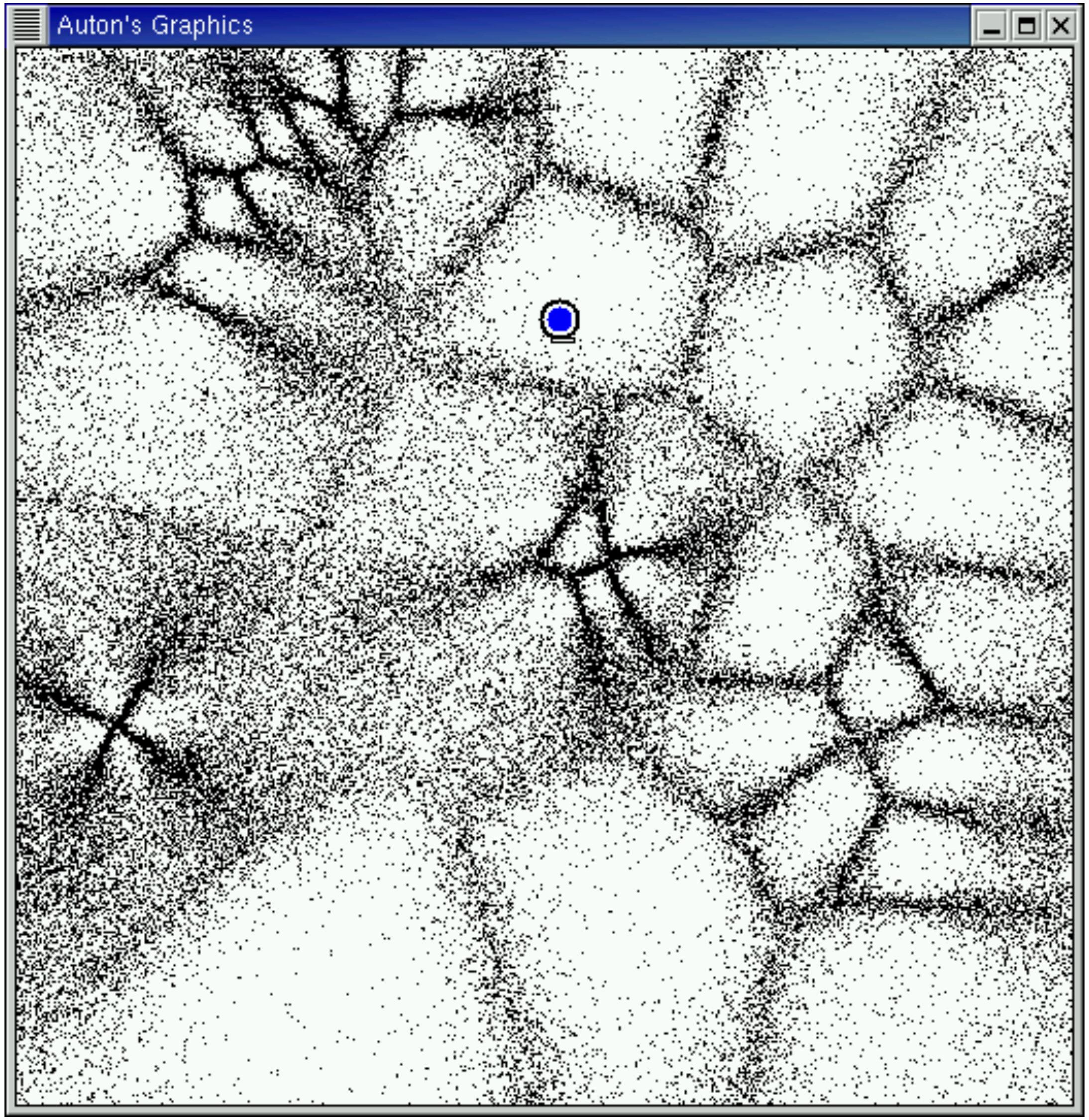

Auton's Graphics

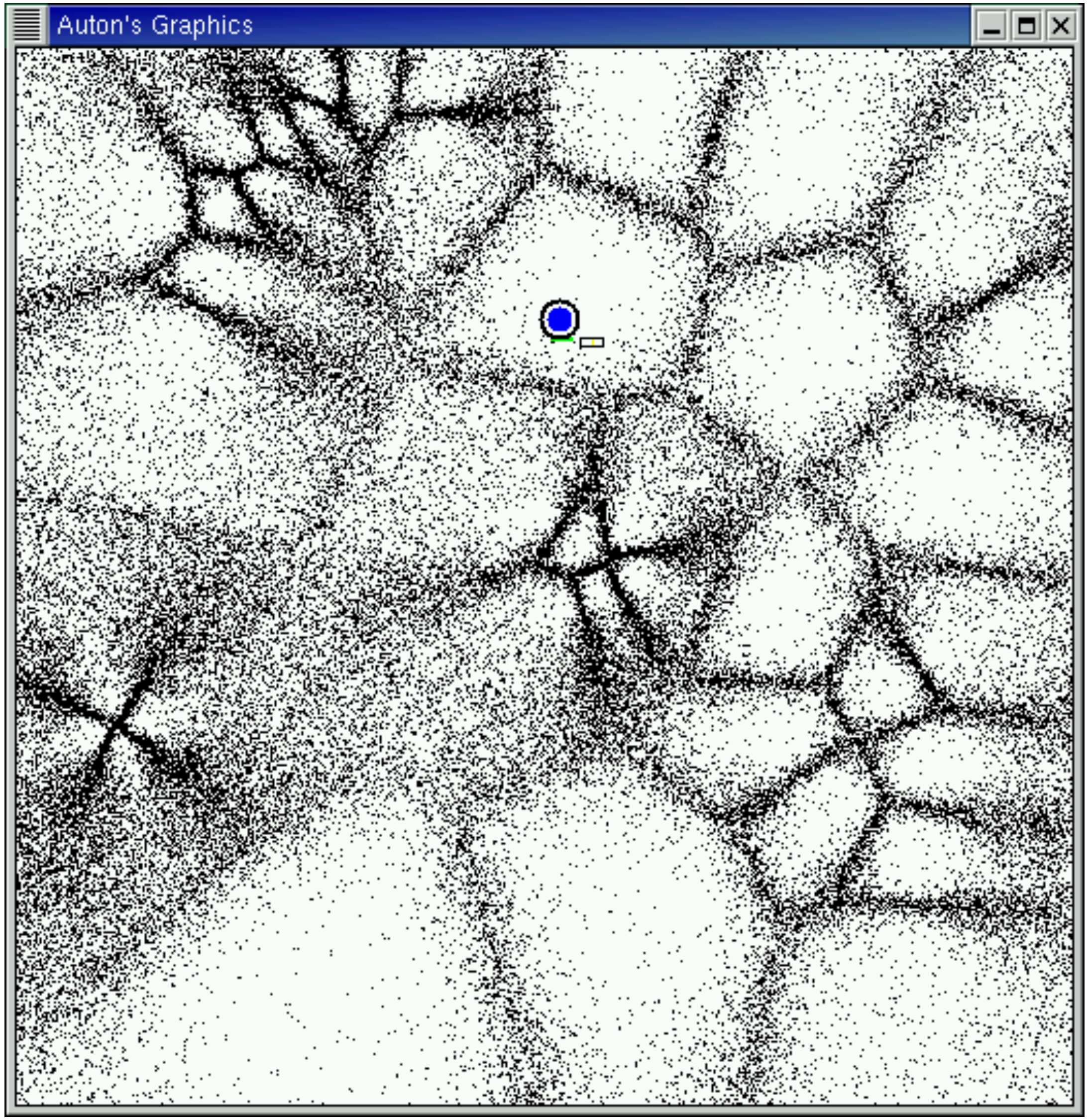

Auton's Graphics

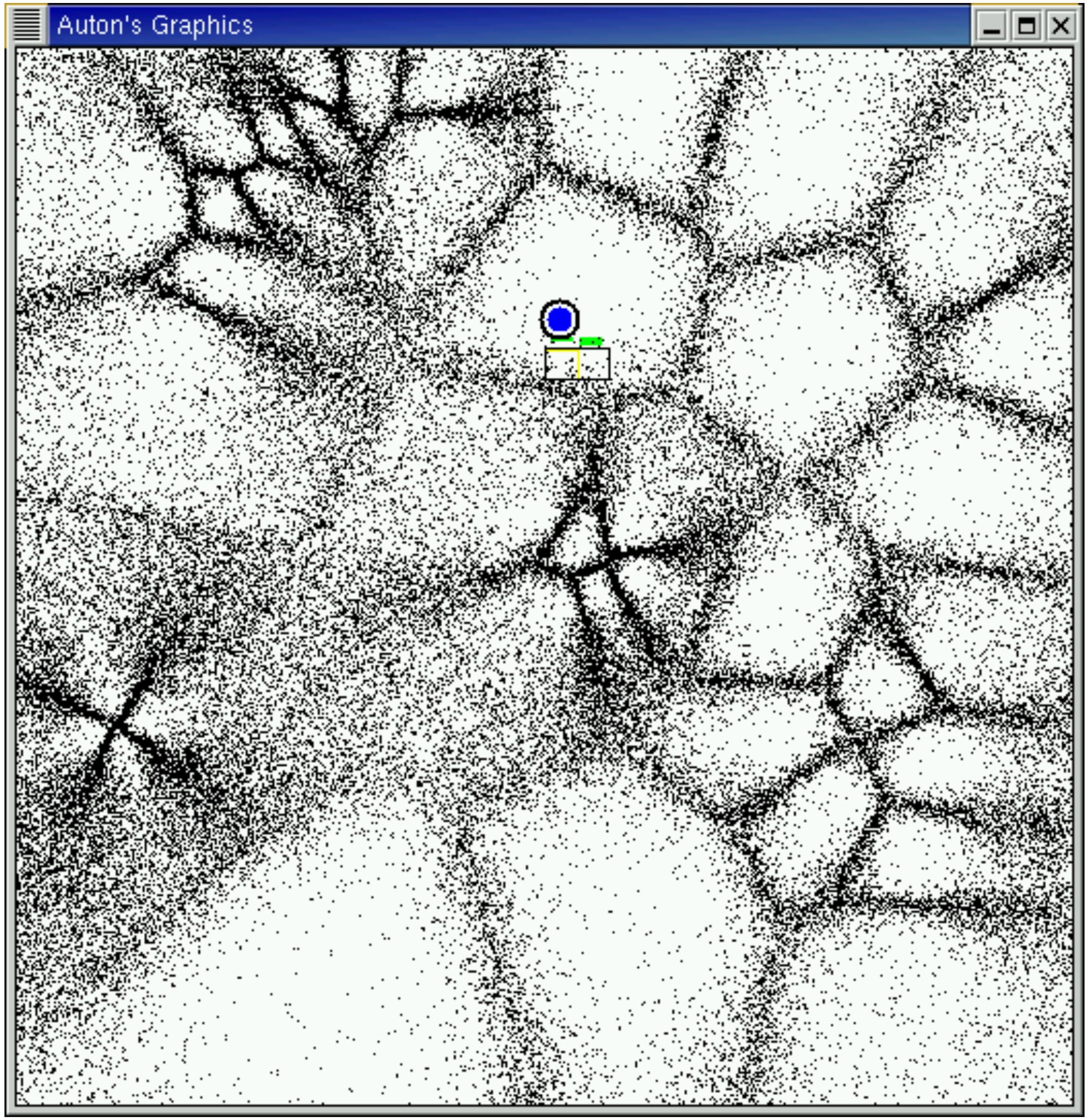

Auton's Graphics

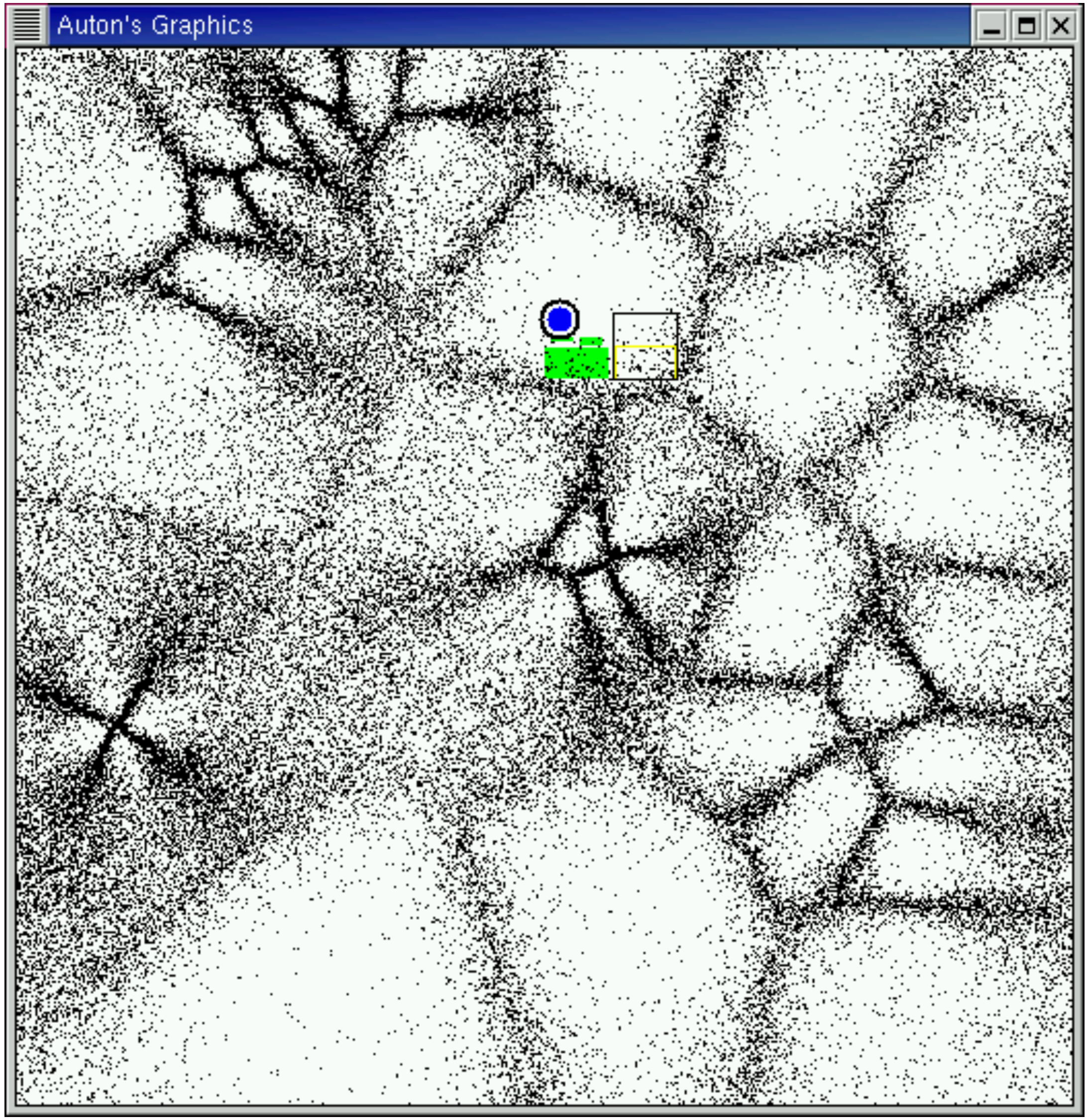

Auton's Graphics

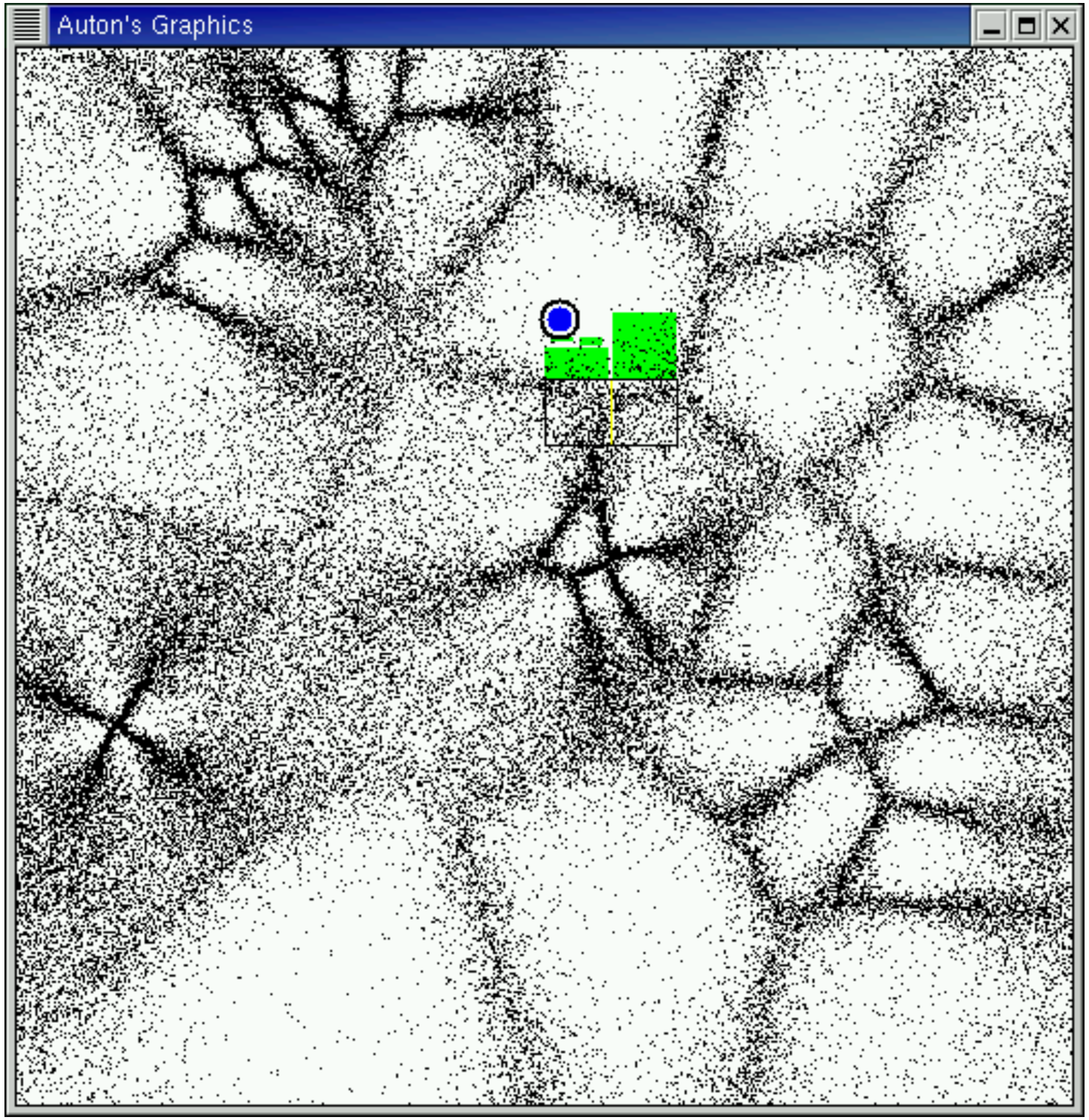

Auton's Graphics

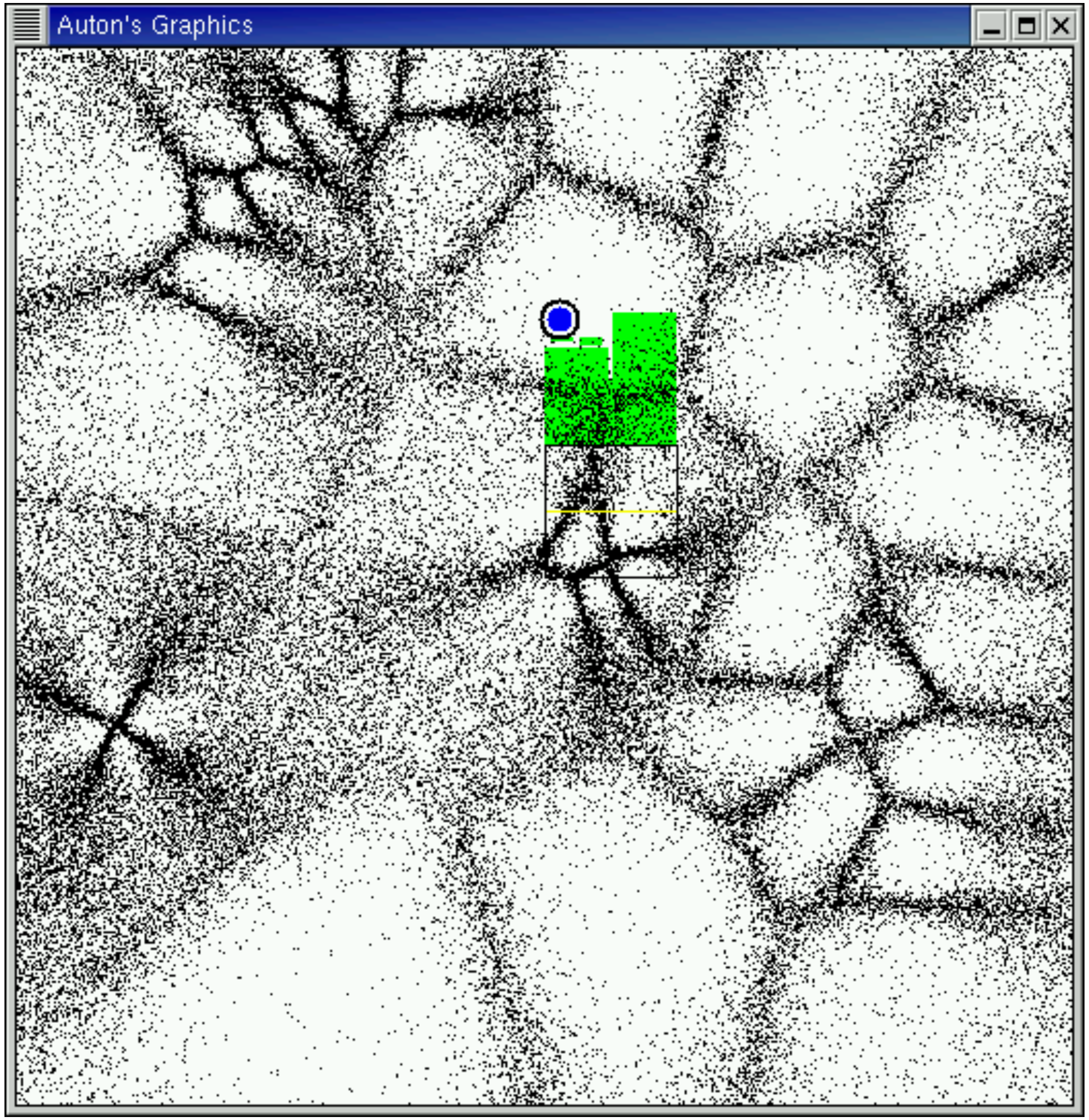

Auton's Graphics

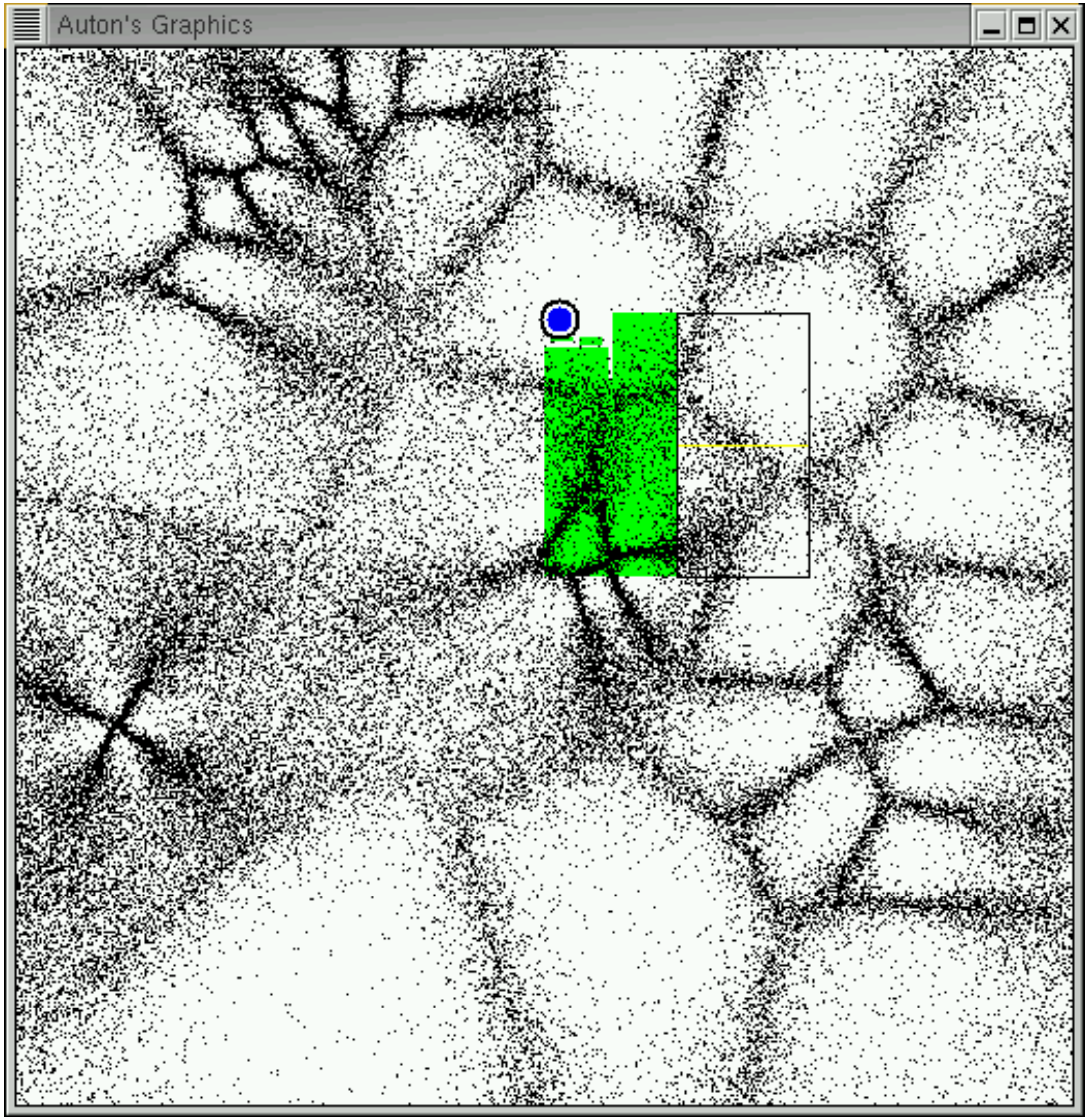

Auton's Graphics

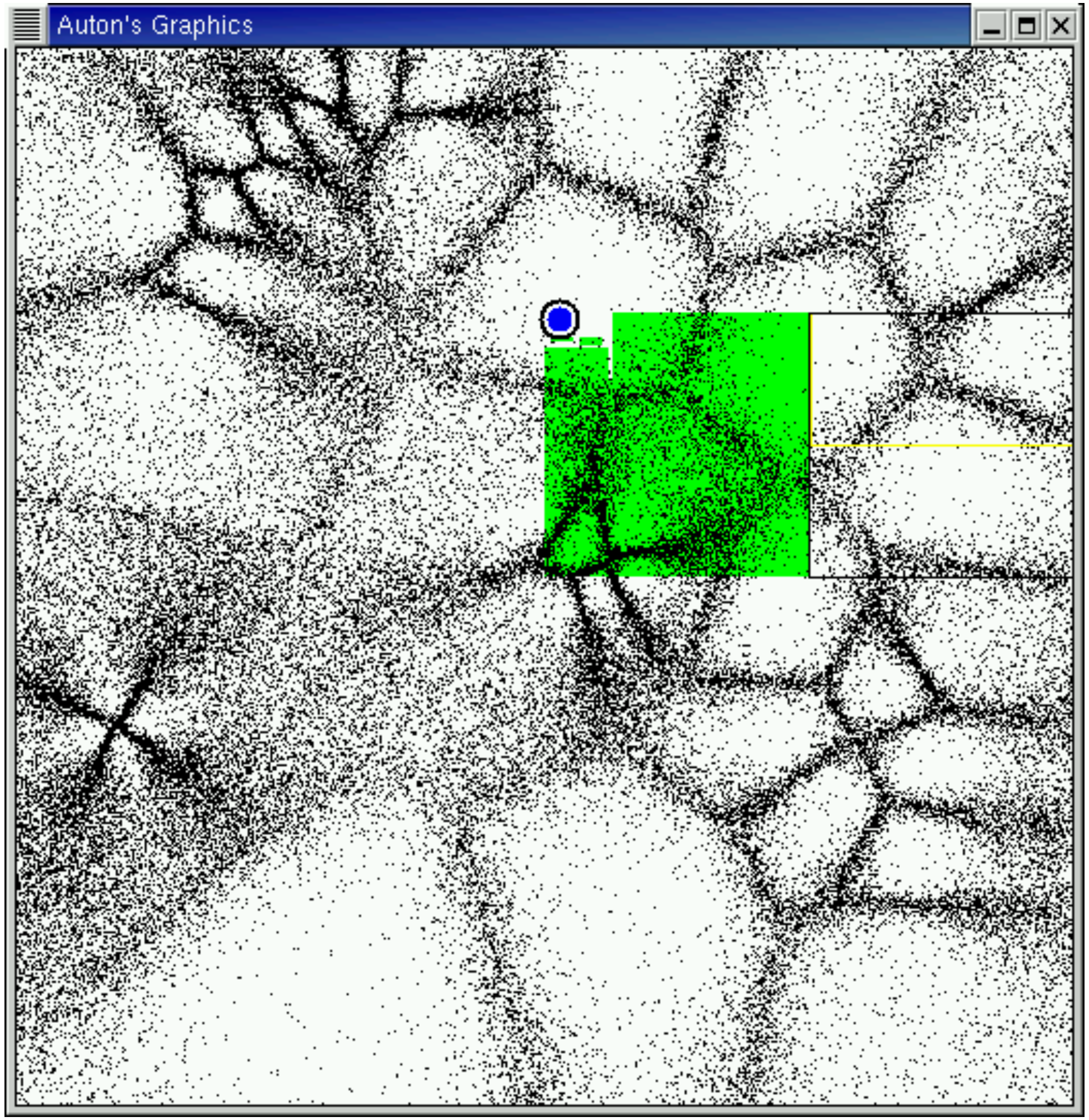

Auton's Graphics

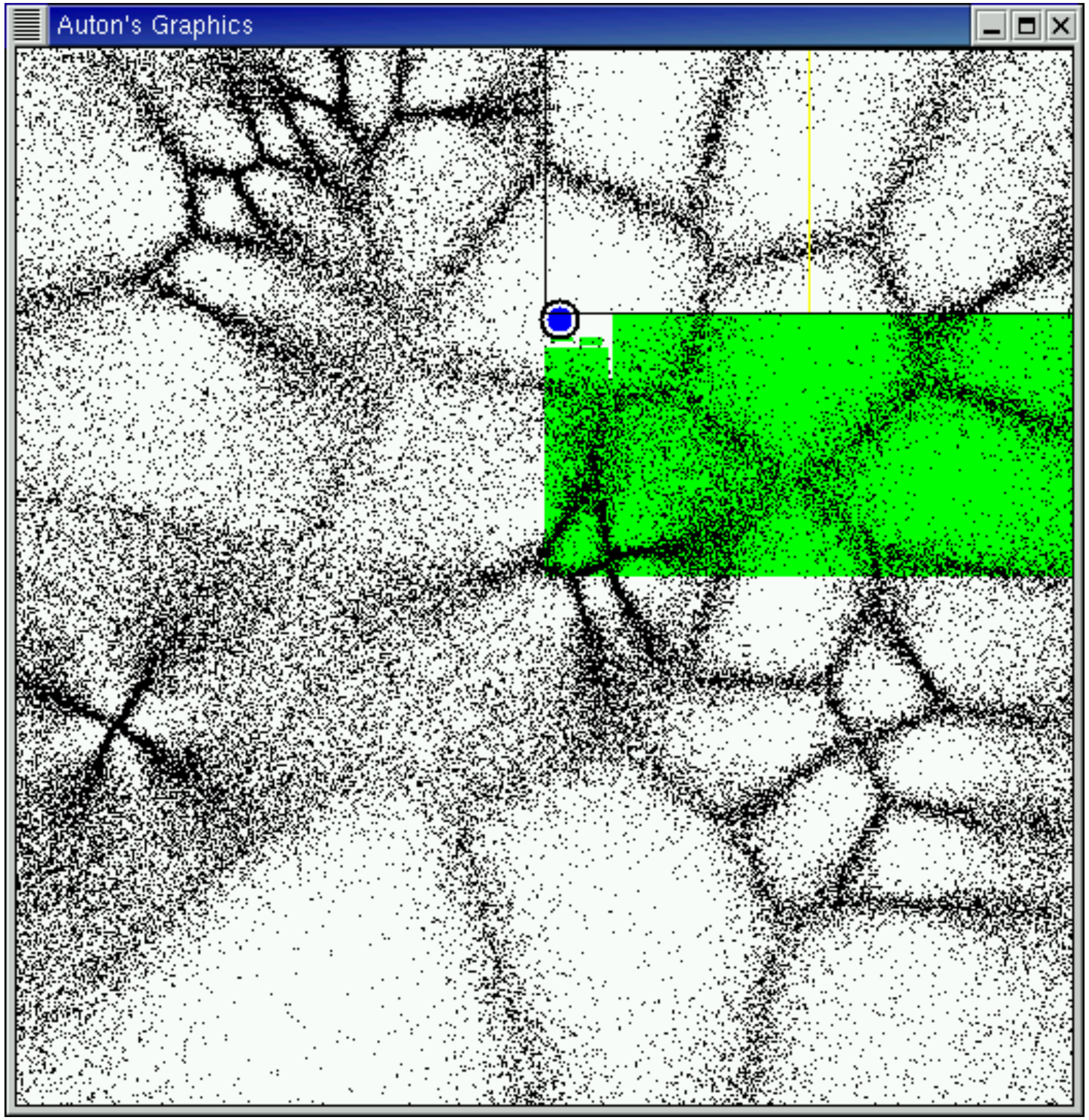

Auton's Graphics

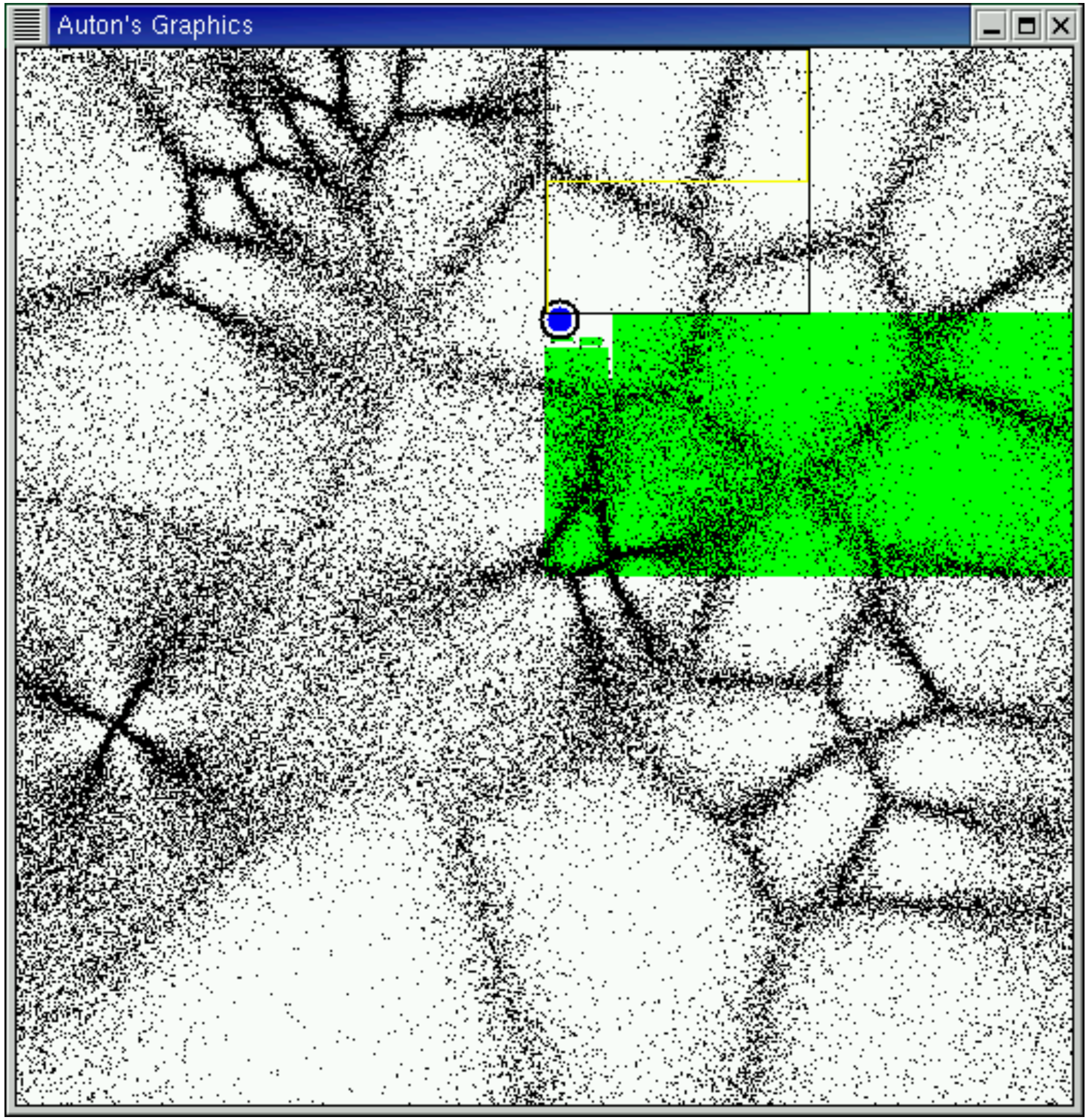

Auton's Graphics

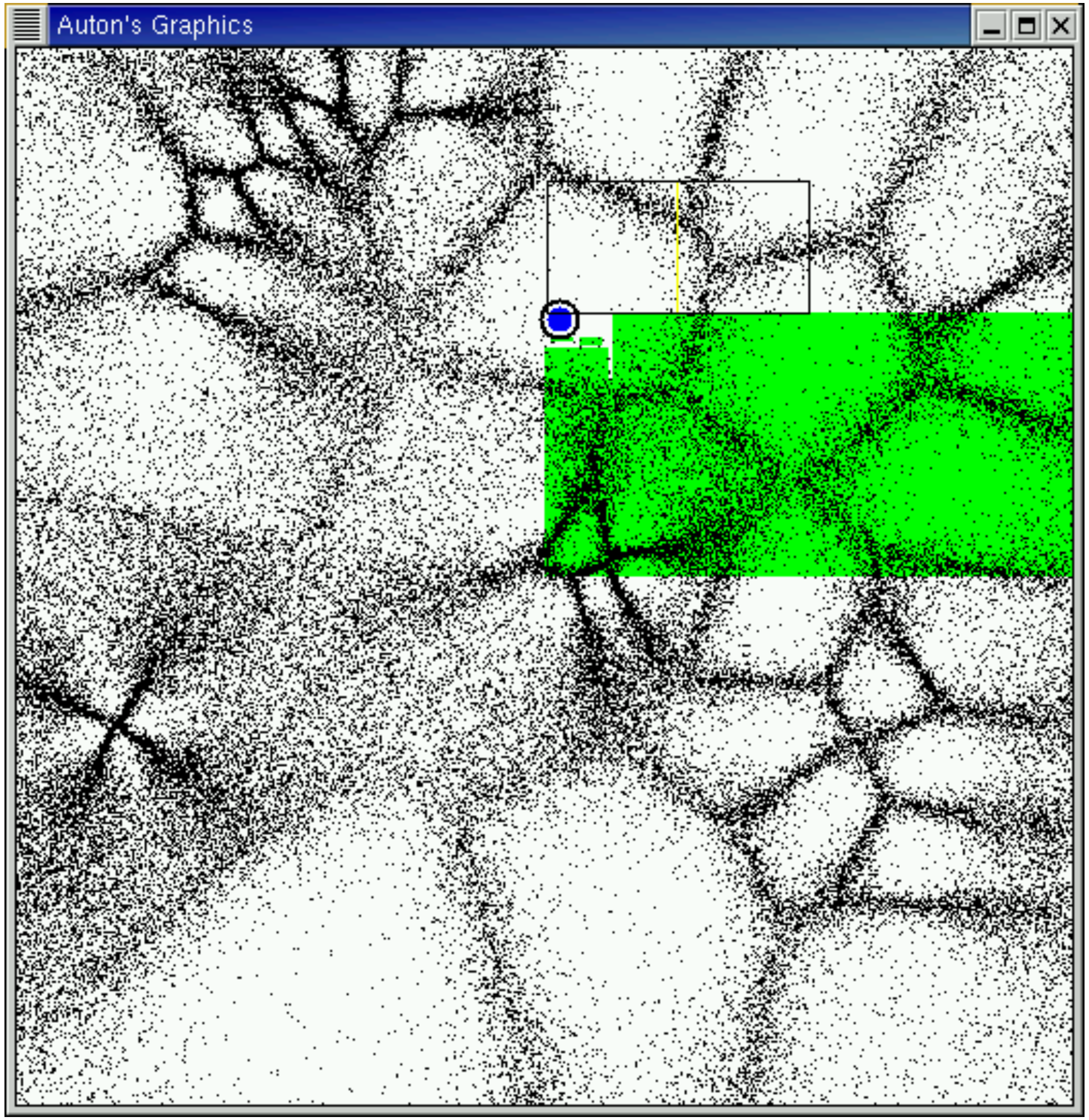

Auton's Graphics

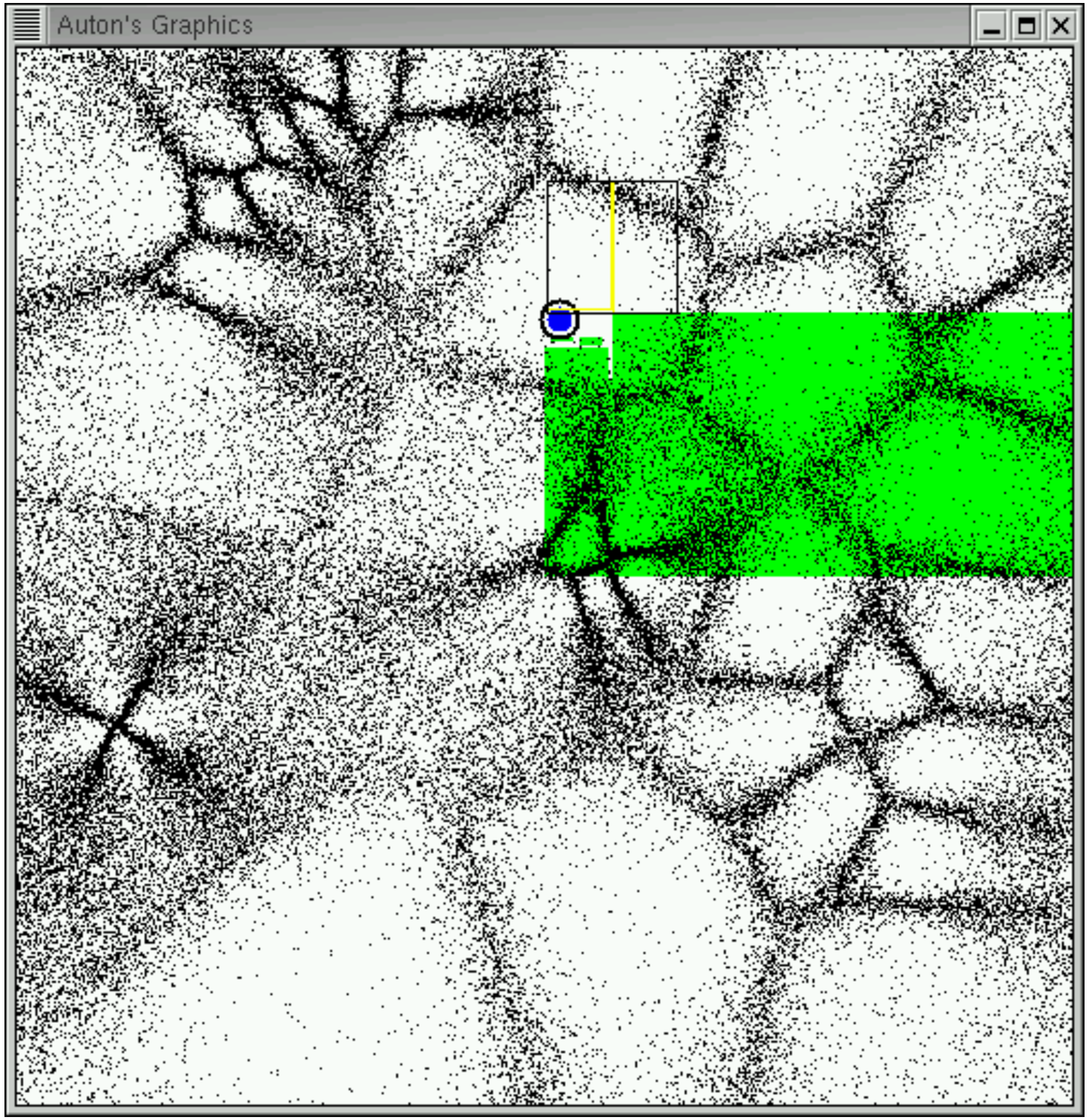

Auton's Graphics

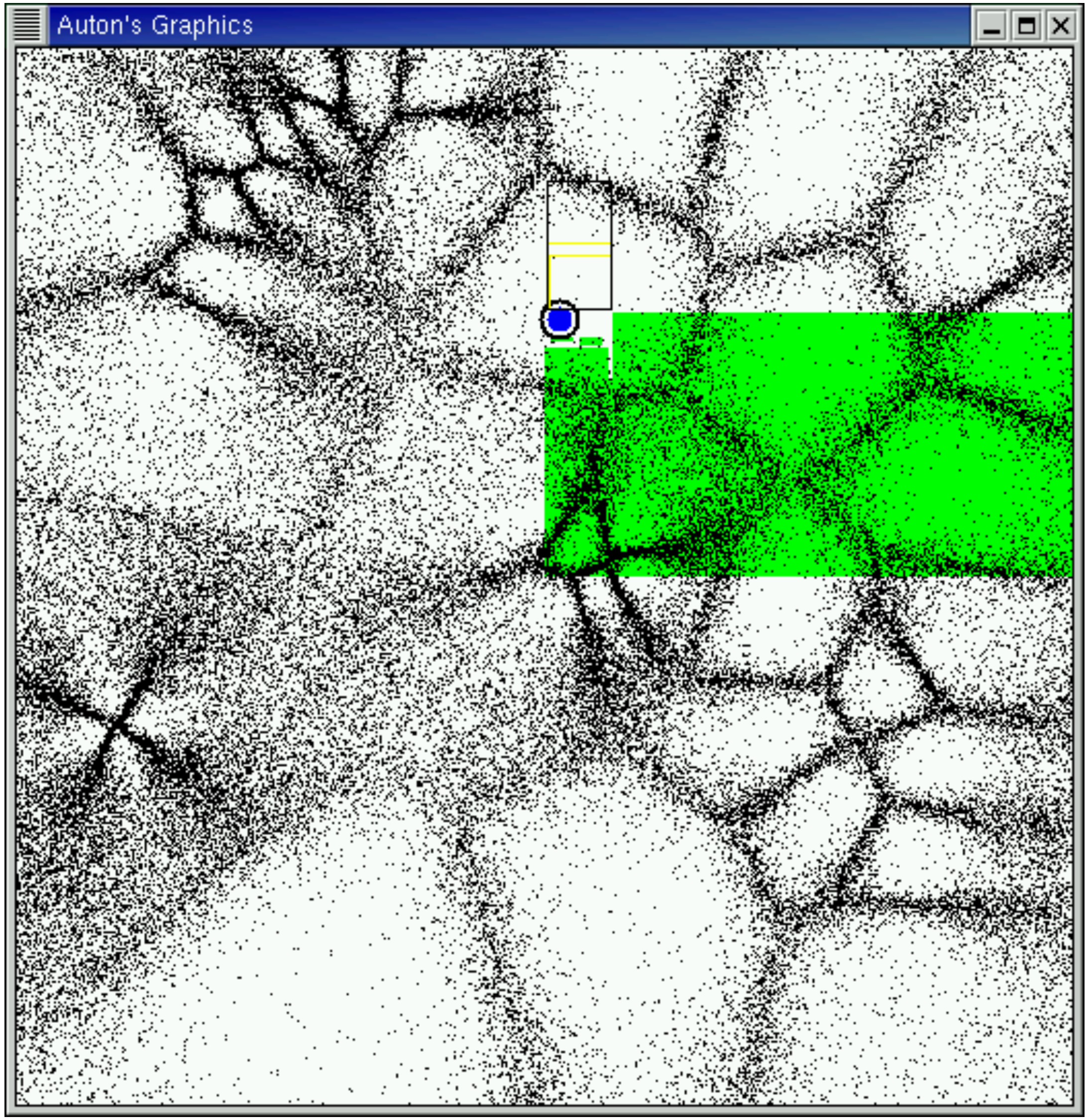

Auton's Graphics

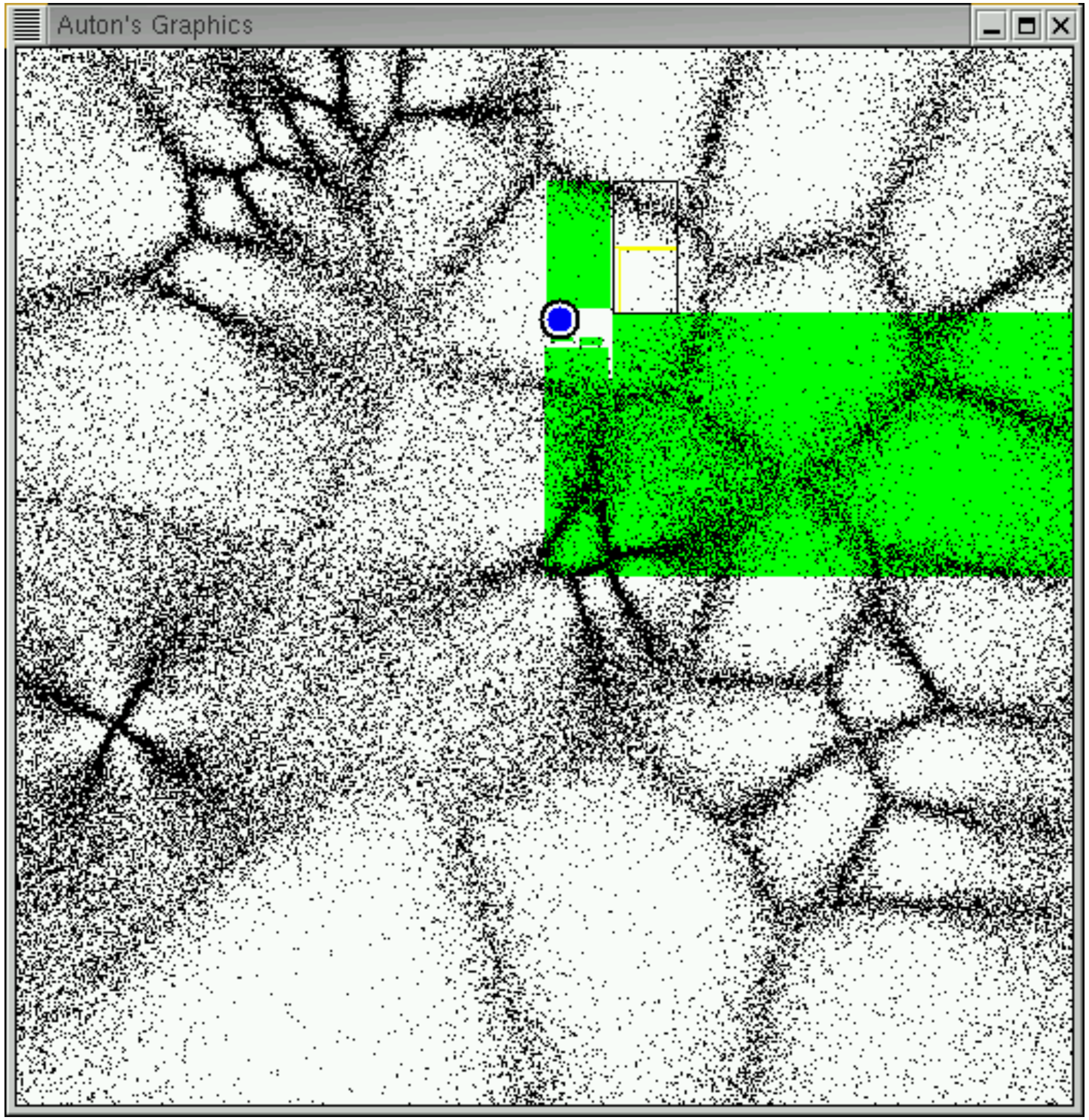

Auton's Graphics

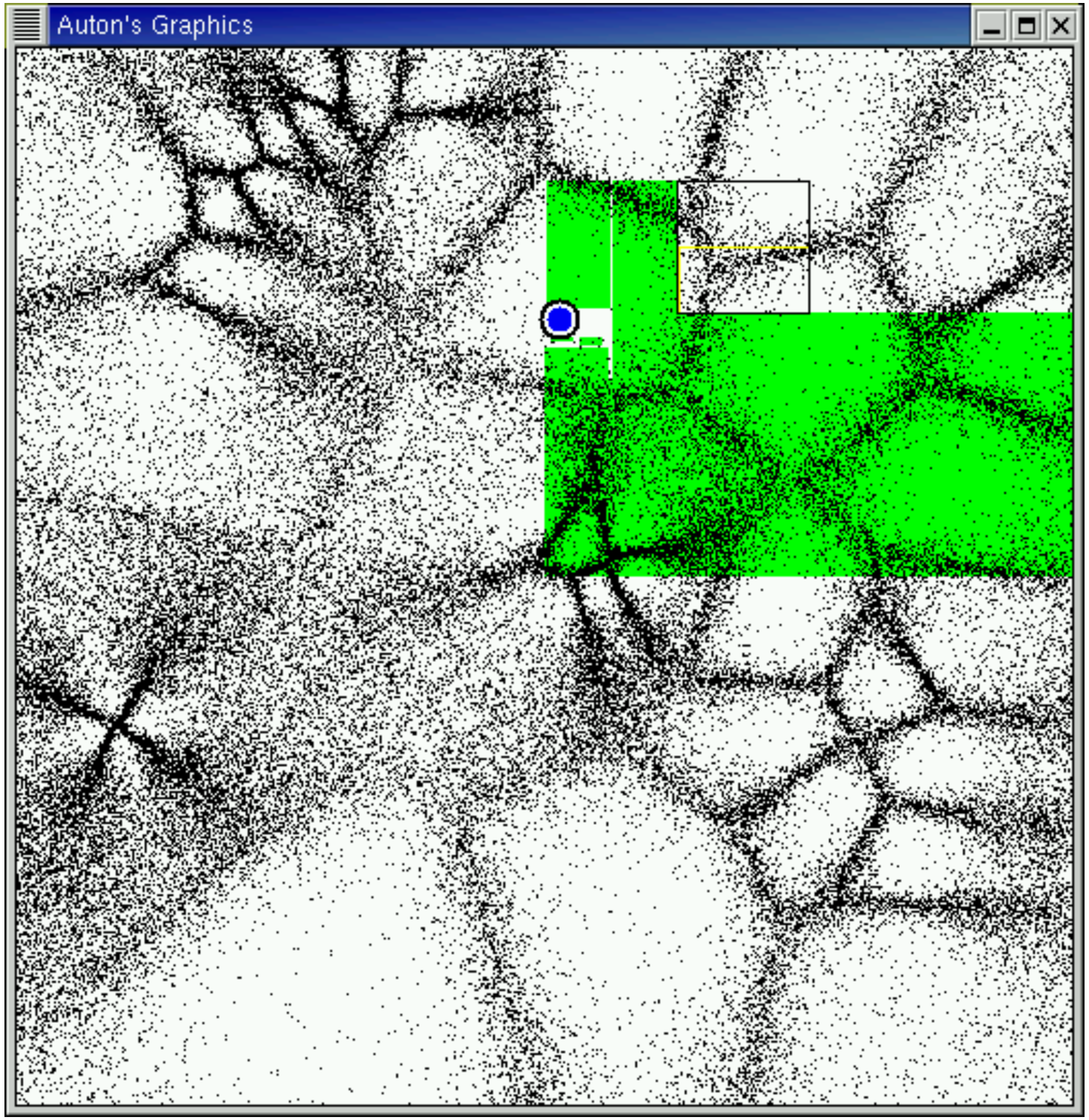

Auton's Graphics

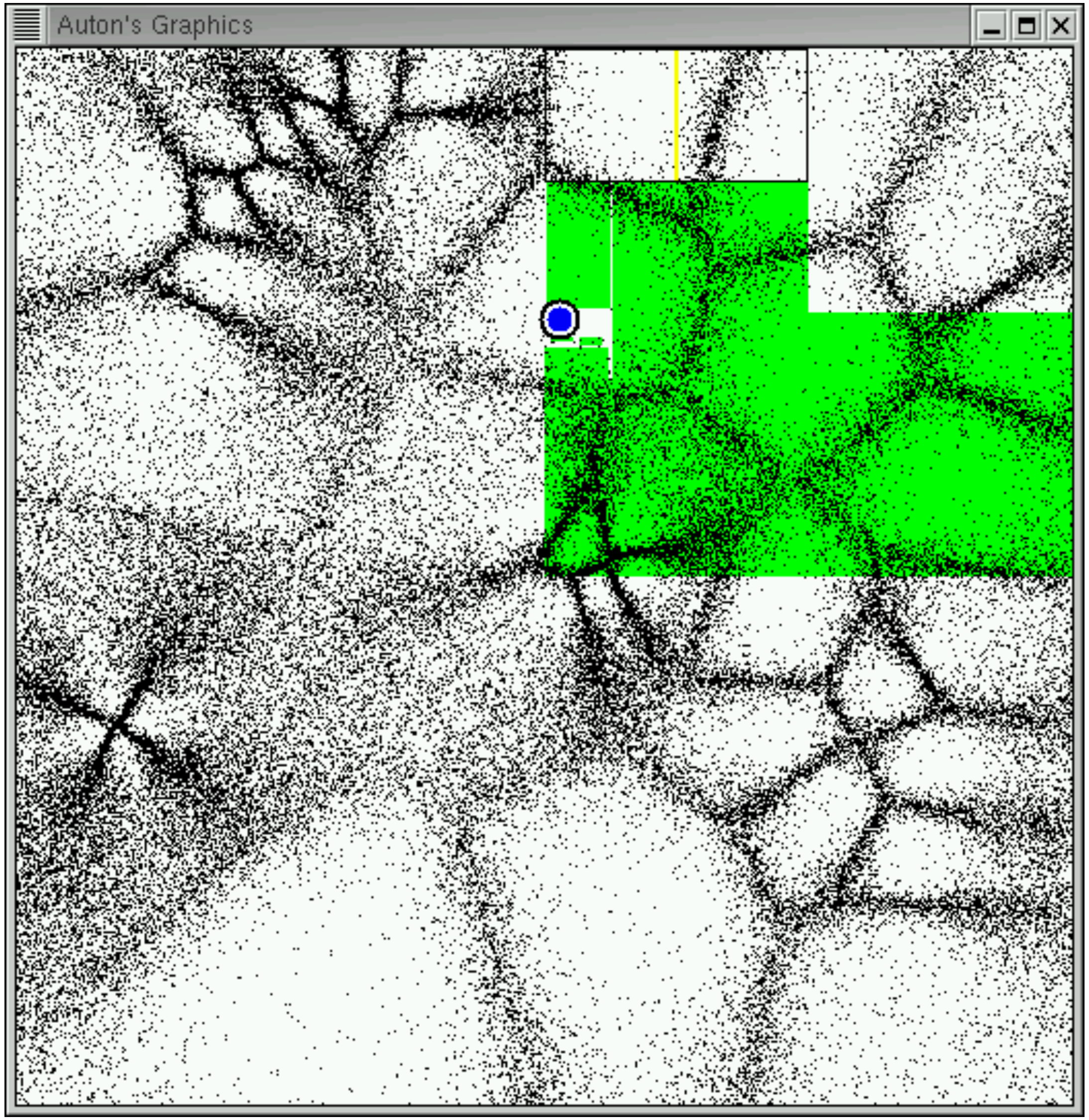

Auton's Graphics

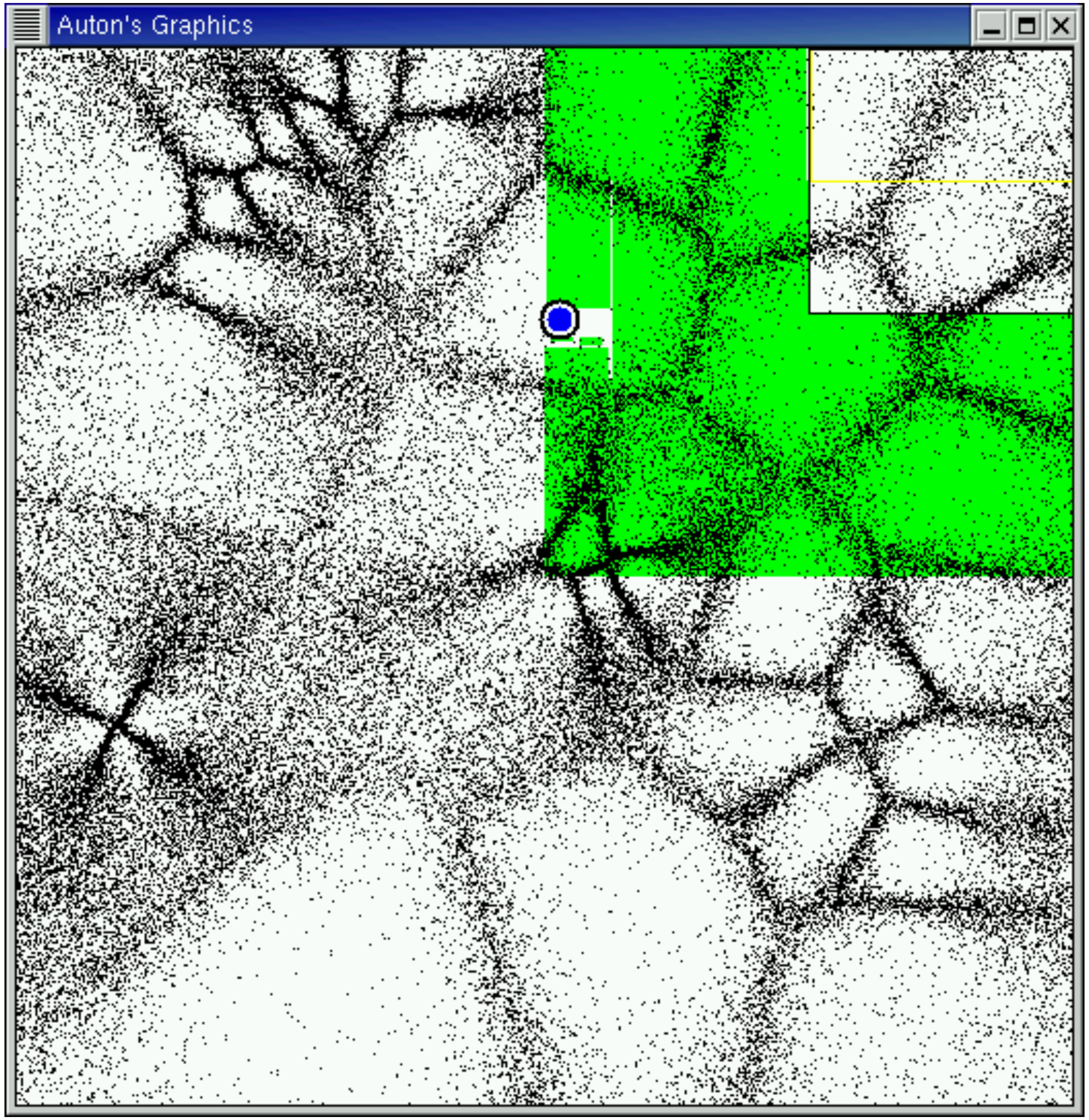

Auton's Graphics

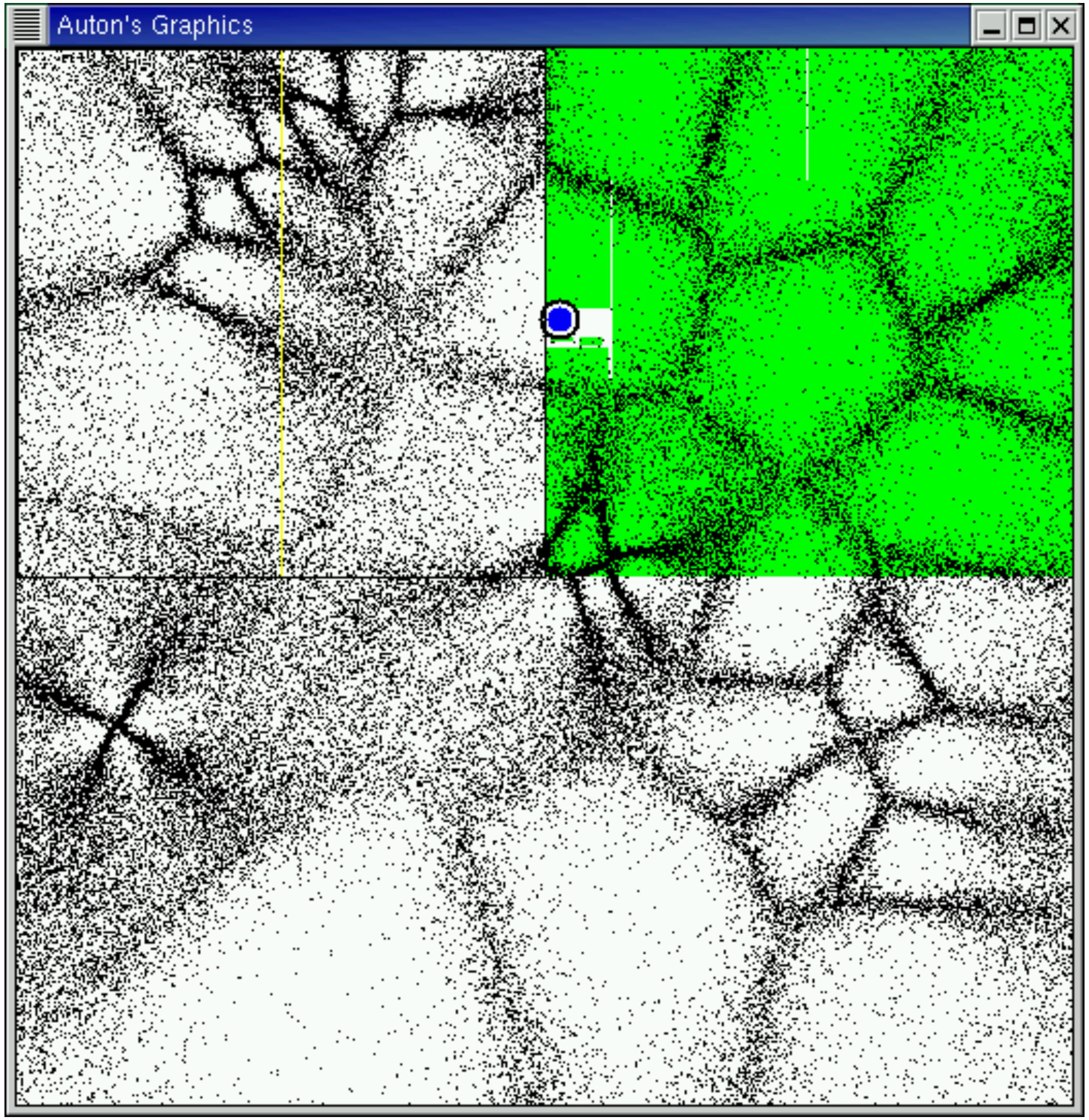

Auton's Graphics

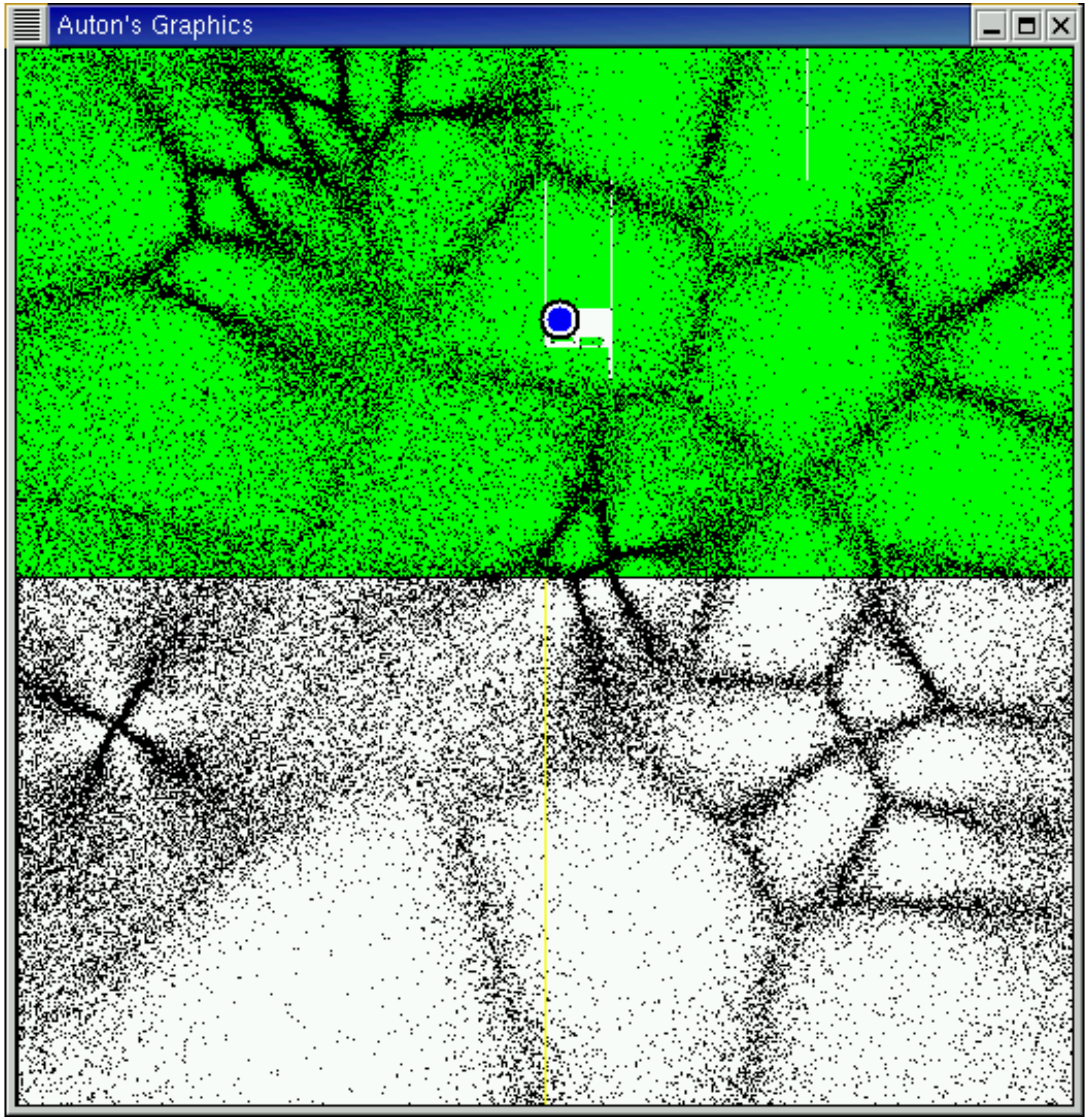

Auton's Graphics

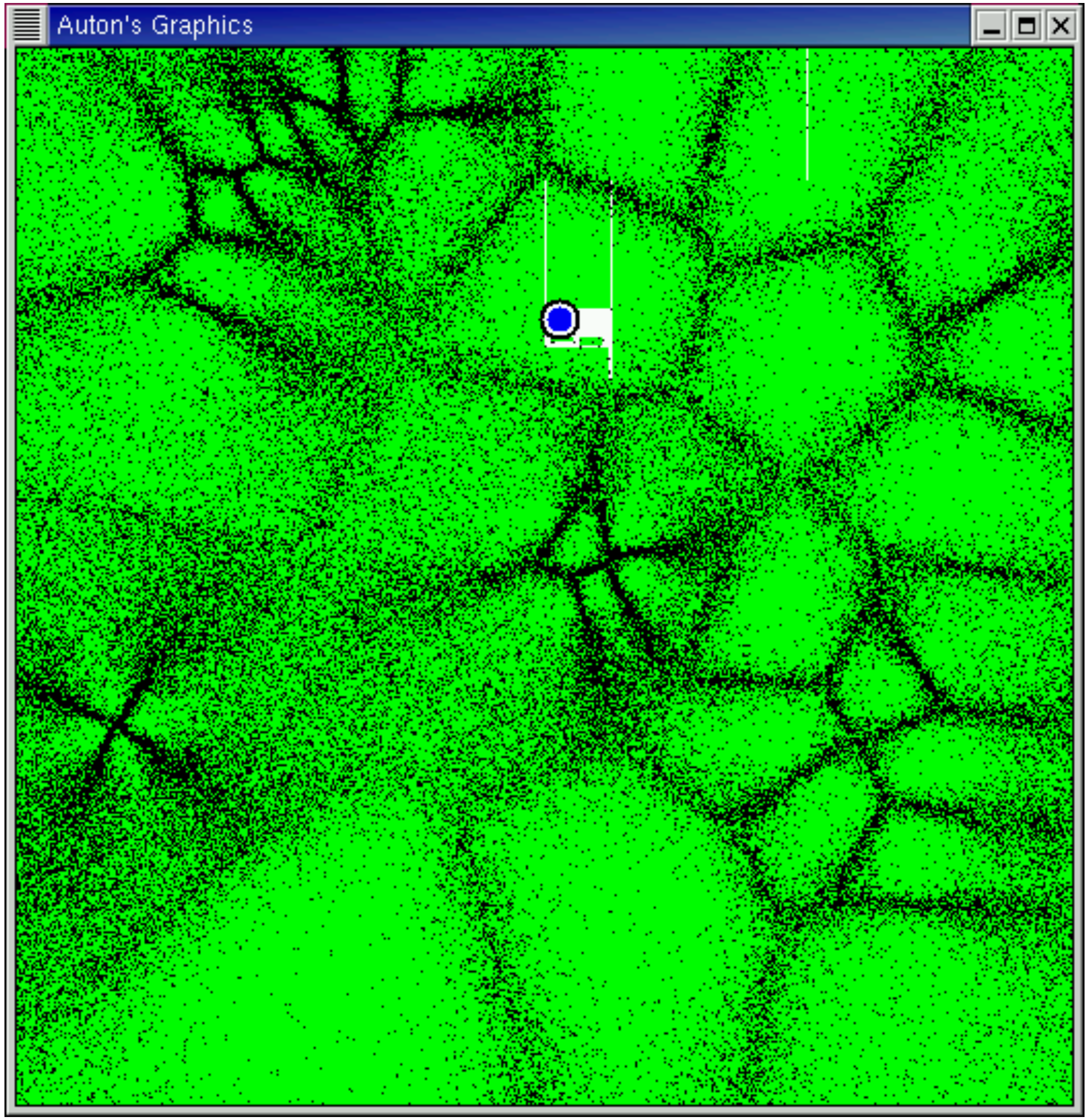

Auton's Graphics


Auton's Graphics


Auton's Graphics

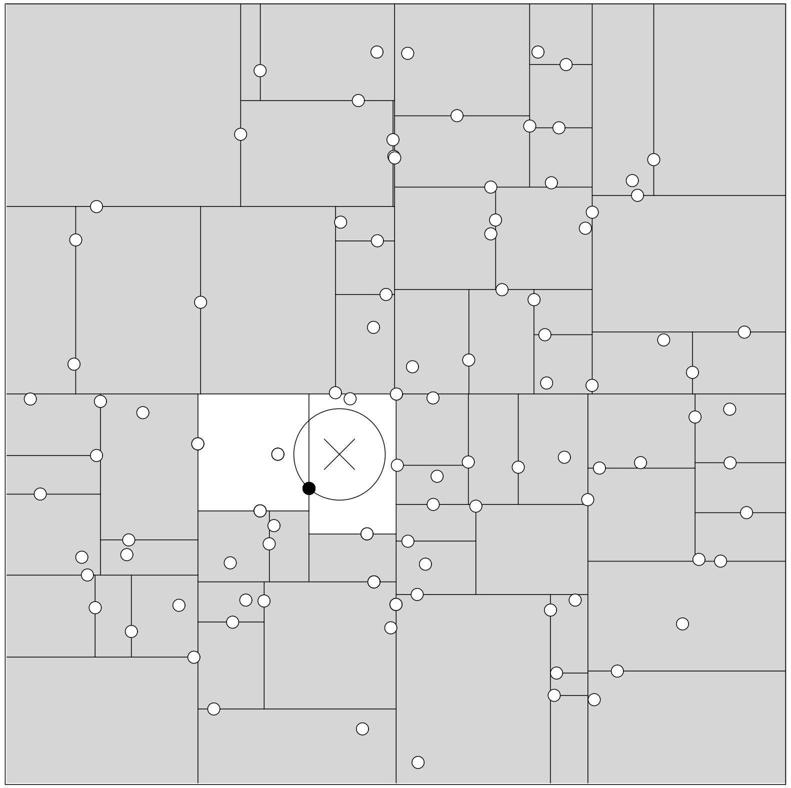

Auton's Graphics


Auton's Graphics

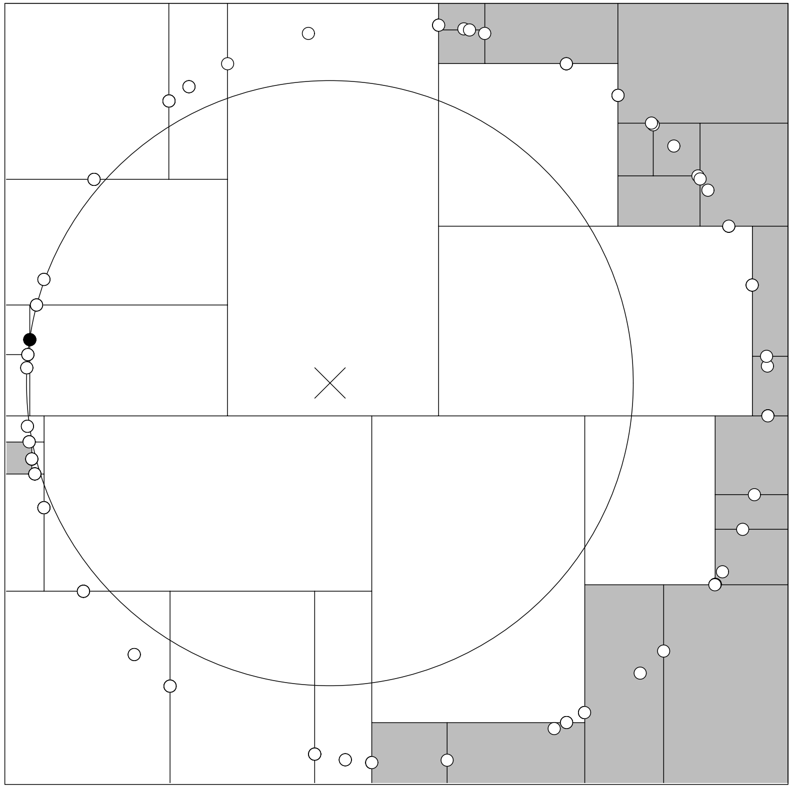

Auton's Graphics

A Worst-Case for NN-Search Using kd-Trees

Gutartiger Fall



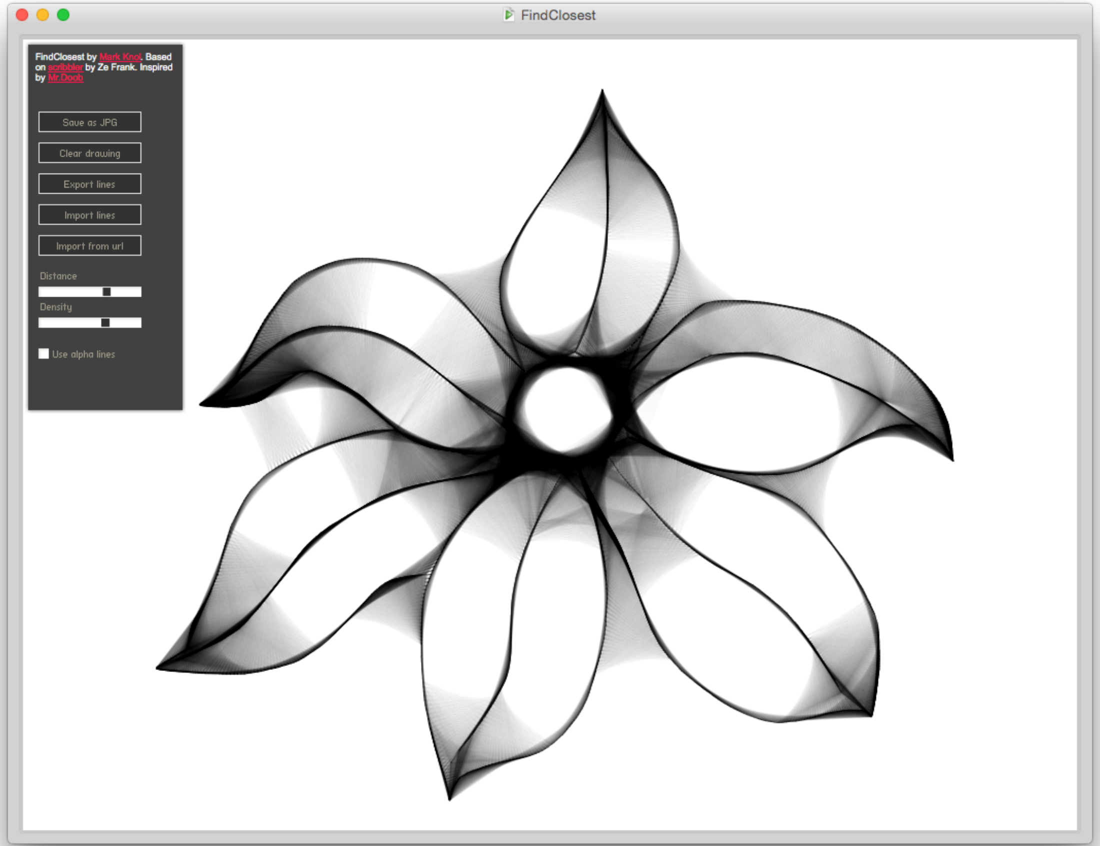
Bösartiger Fall



Alle weißen Blätter muß der NN-Algorithmus besuchen!

In a few moments, it will get worse ...

Artistic Application of k-NN Algorithm



Flatland (Edwin A. Abbott, presented by Carl Sagan)

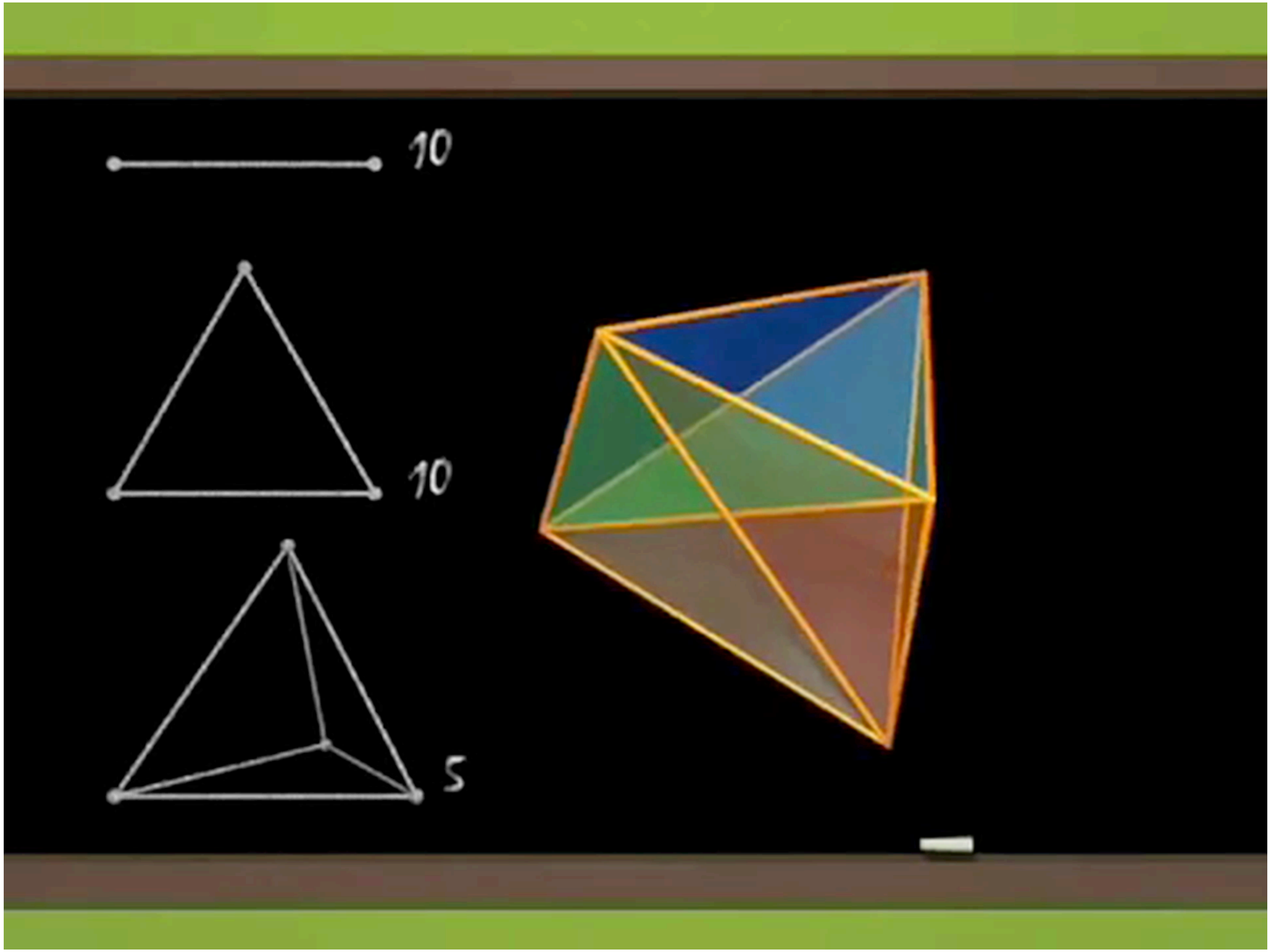
- Denksportaufgabe: wie sieht ein Würfel aus, der langsam duch Flatland hindurch "schwebt", beginnend mit einer Ecke?
- Was kann ein höher-dimensionales Wesen mit niedriger-dimensionalen Wesen machen:

4-Dimensional Tetrahedron by the Slicing Method

1-simplex

2-simplex

3-simplex



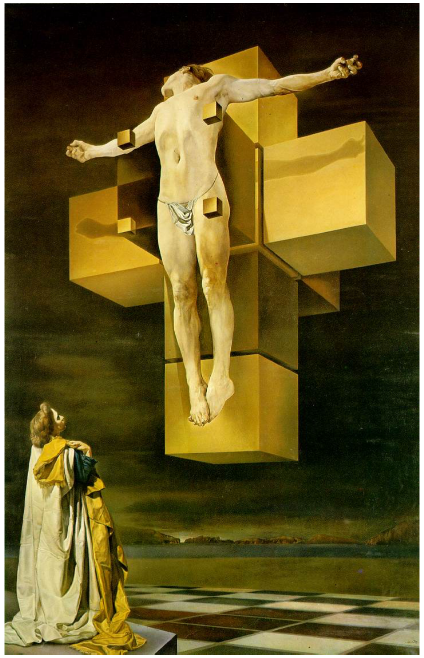
4-simplex

http://www.dimensions-math.org

The 4-Dimensional Hypercube (Tesseract)

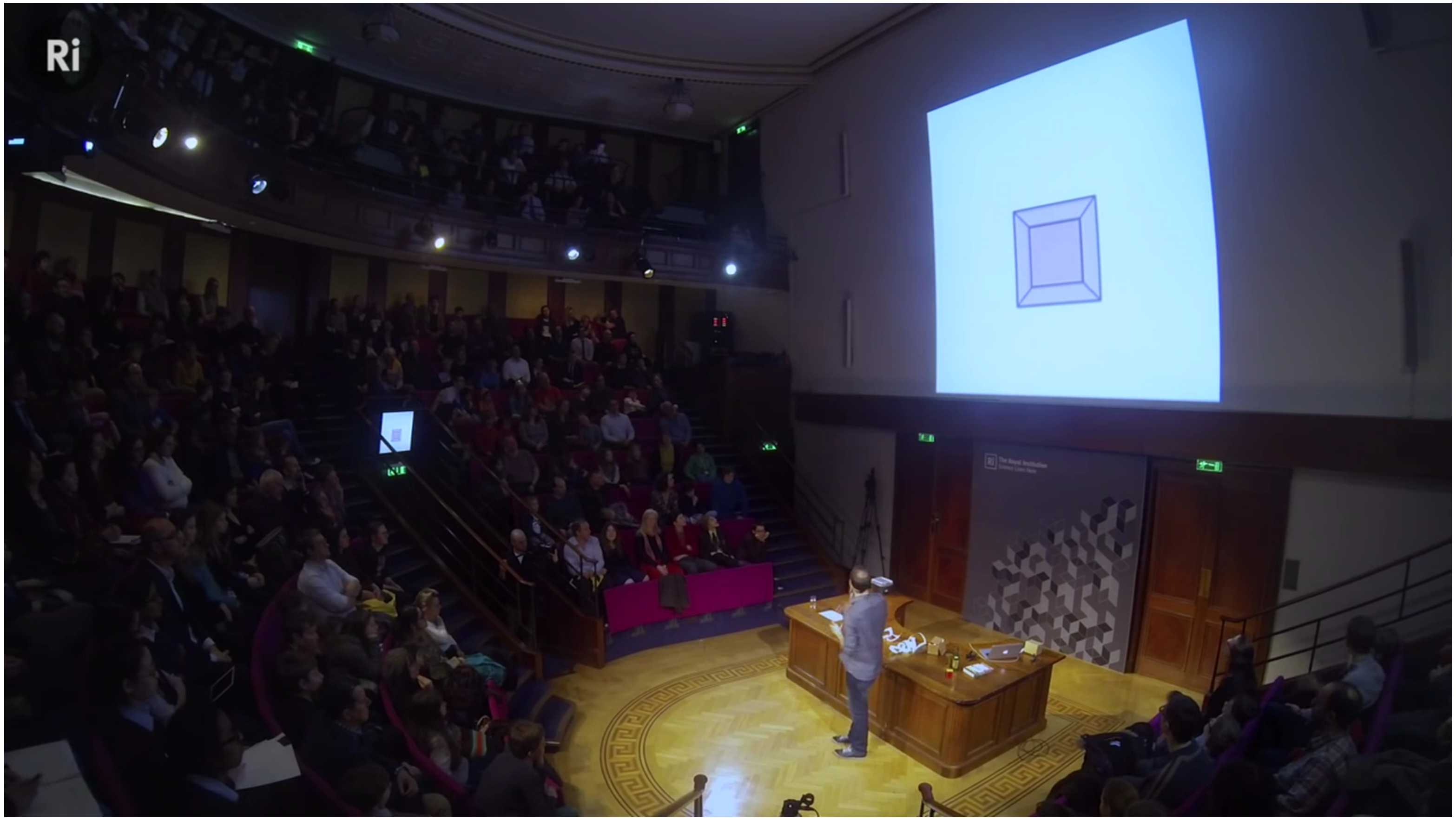
- Construction by analogy: number of points, edges, faces, cells
- Projection eines Tesseract nach 3D:

Unwrapping a Hypercube



Crucifixion (Corpus Hypercubus), 1954, Salvador Dali

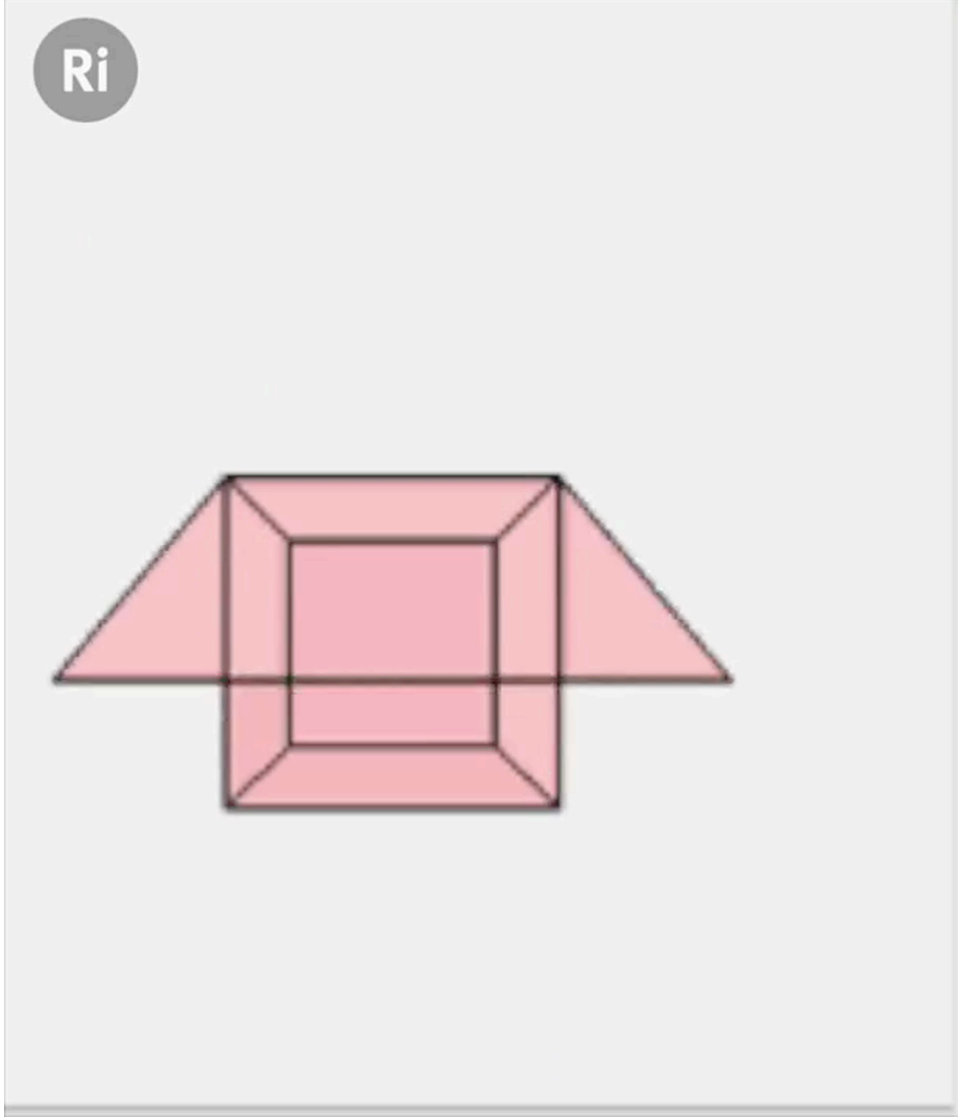
The unfolding method:
The projection of a 3D cube unfolding into its 2D net

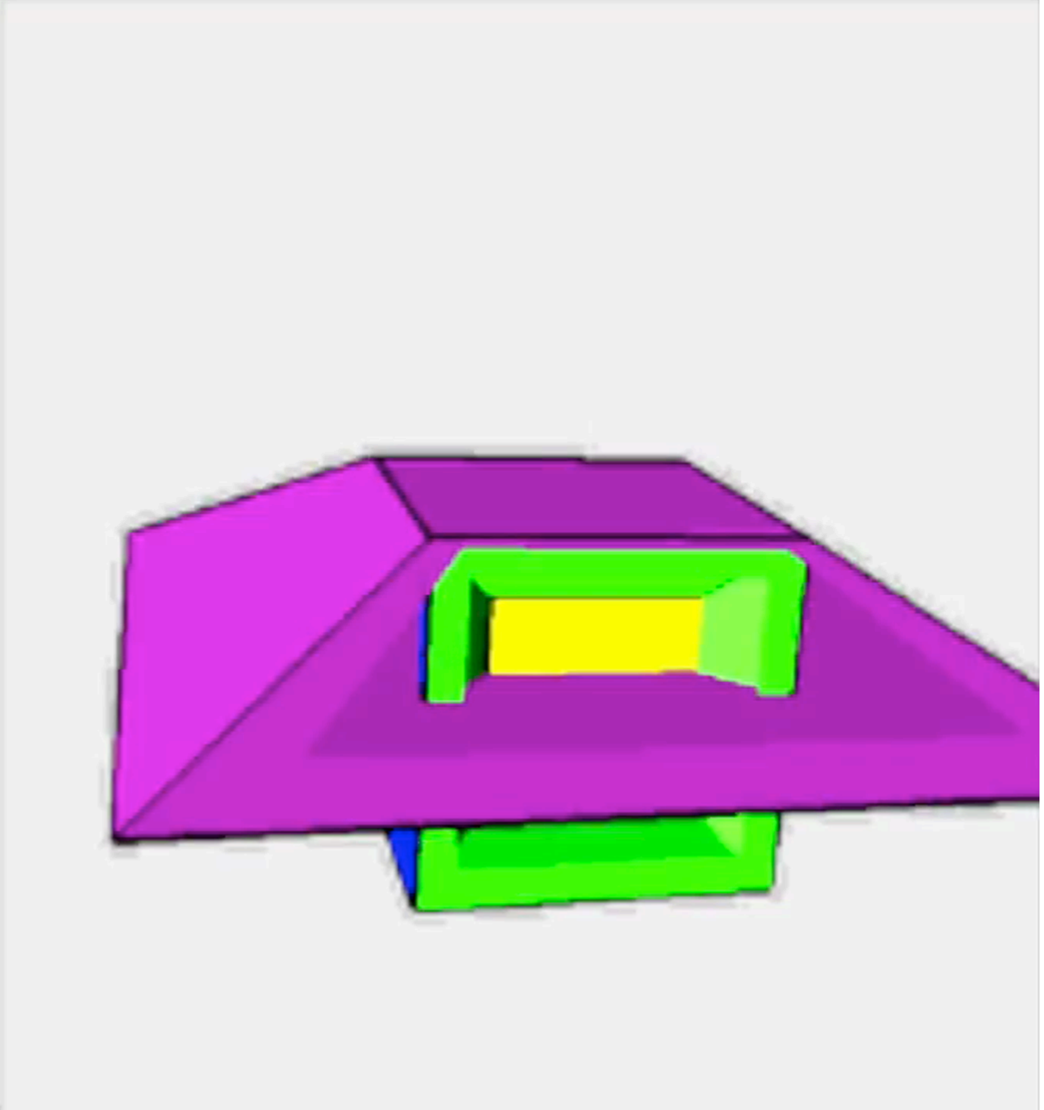


Matt Parker

Projection of 3D cube unfolding into its 2D net

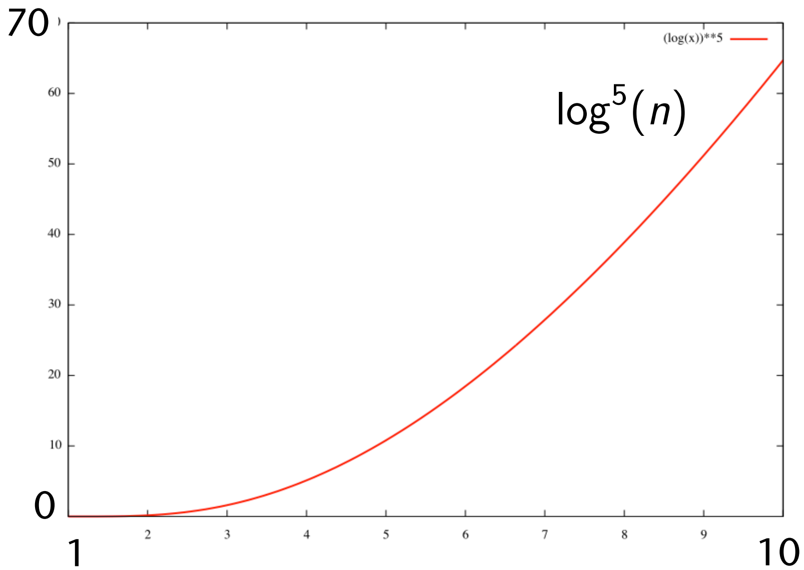
Projection of 4D cube unfolding into its 3D net

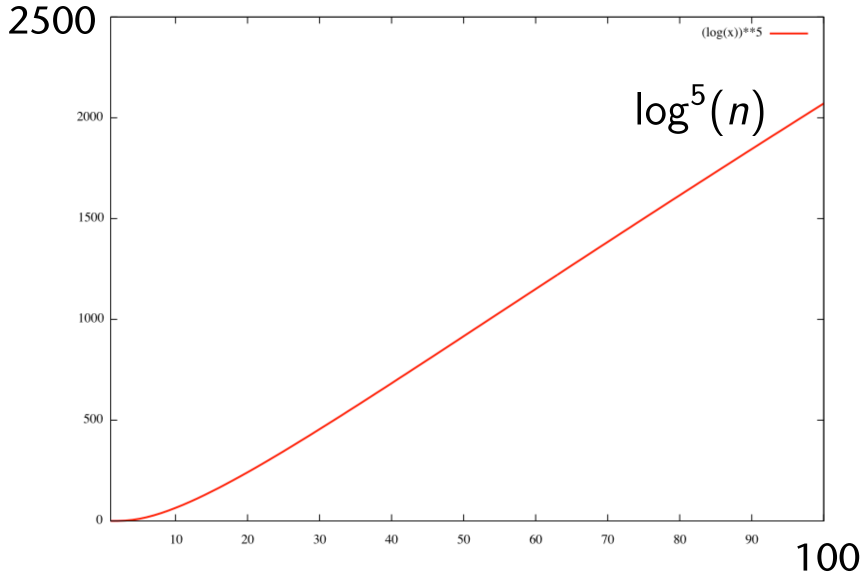


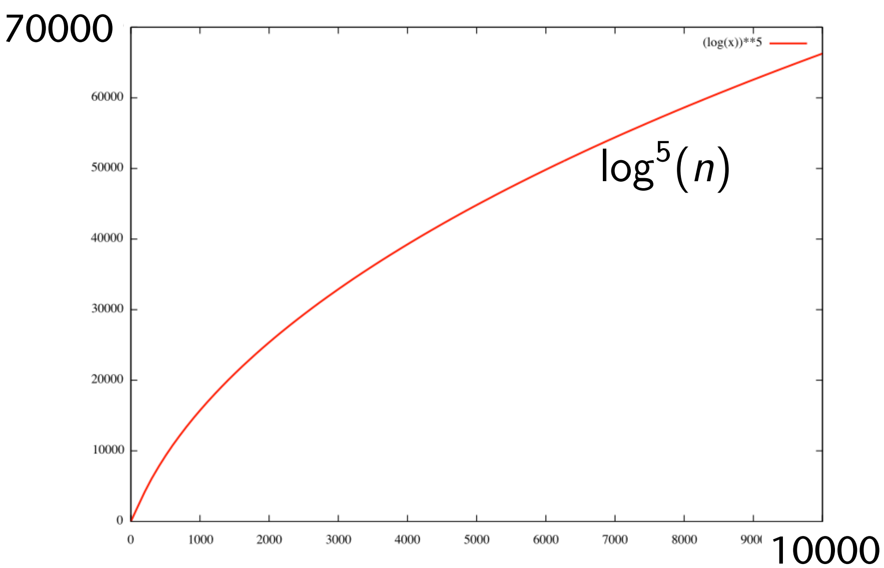


The "lid" of the 4D cube (what is it?) does not deform, of course; that is just an artefact of the projection into 3D, just like the lid of the 3D cube when projected into 2D.

Zum Verhalten von $\log^d(n)$

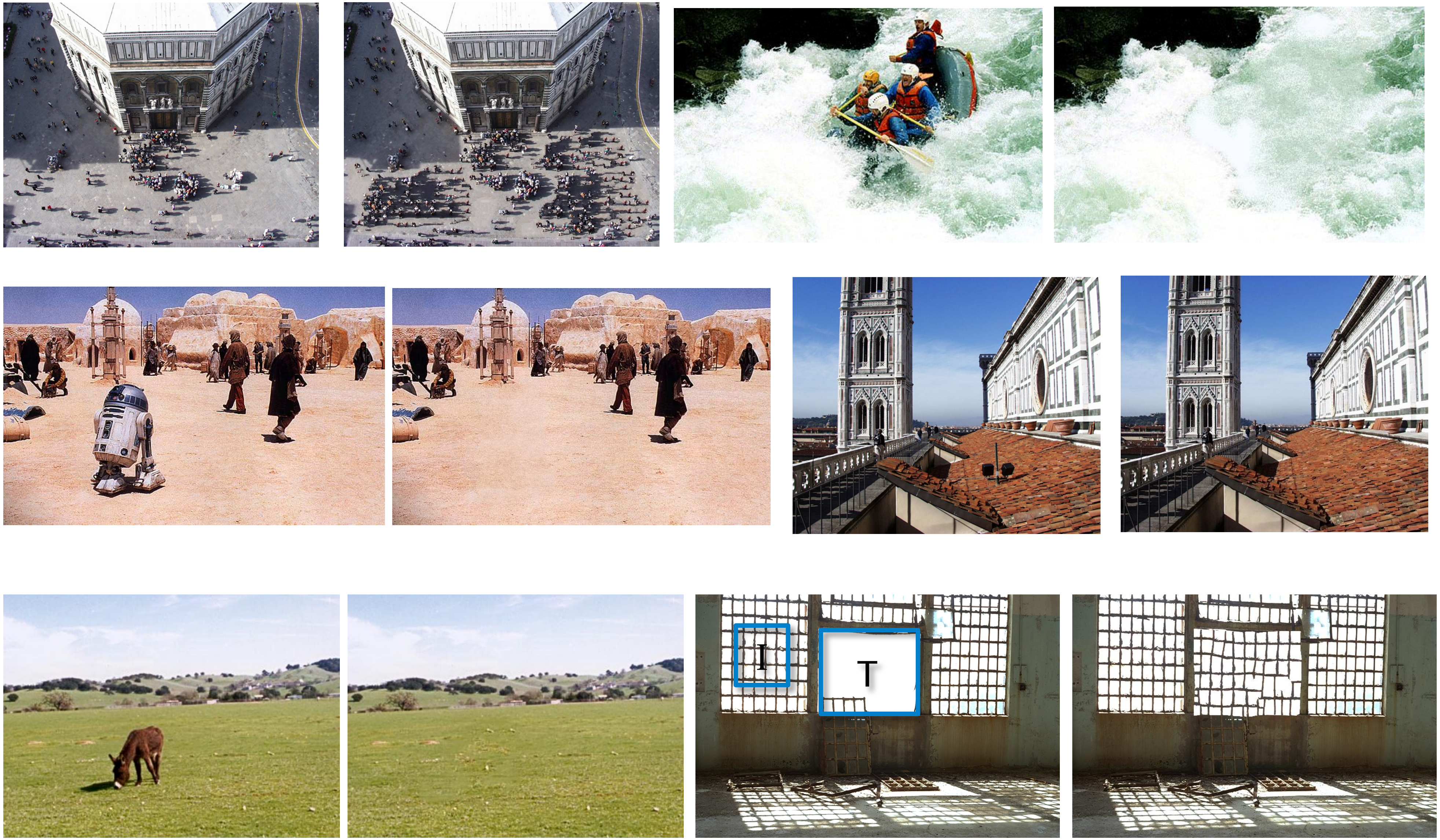






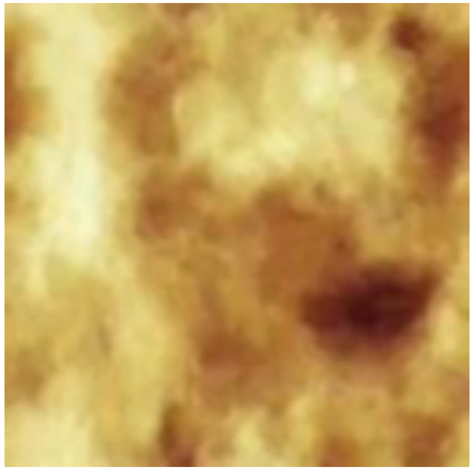
Der Algorithmus für die ANN-Suche ist also besser (asymptotisch) als brute-force-mäßig alle n Punkte zu besuchen und deren Abstand zum Query-Punkt q zu berechnen.

Texture Synthesis



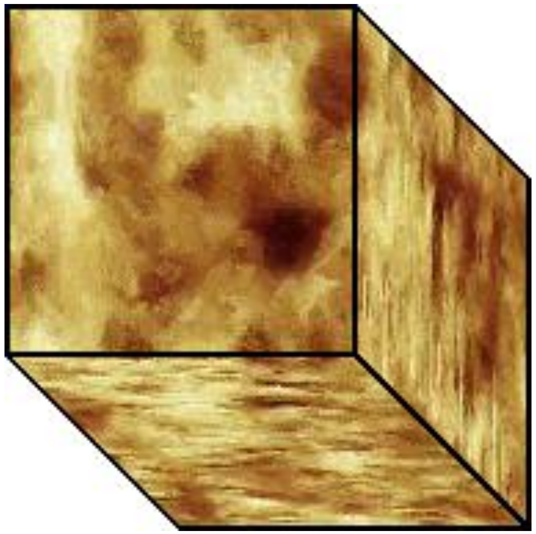
Wei & Levoy

Wei & Levoy



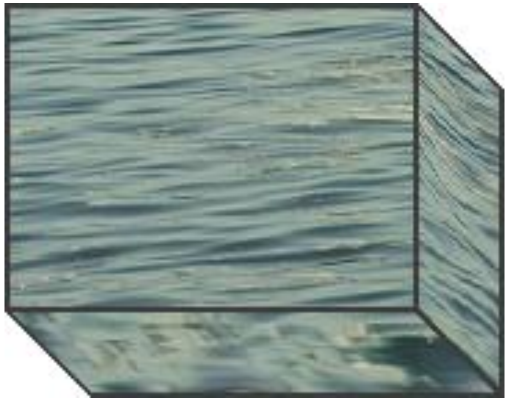
original

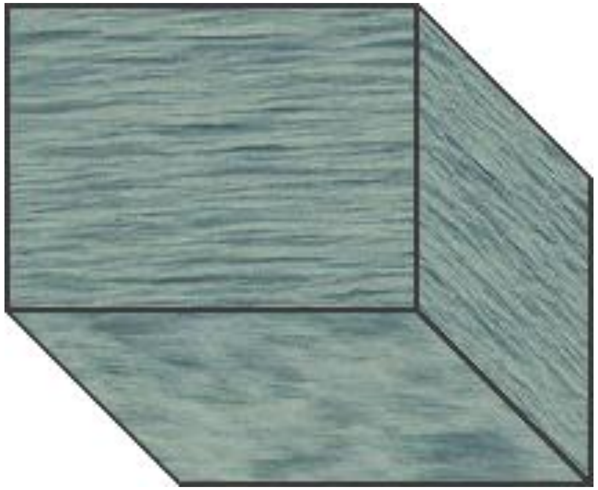
synthesized



original

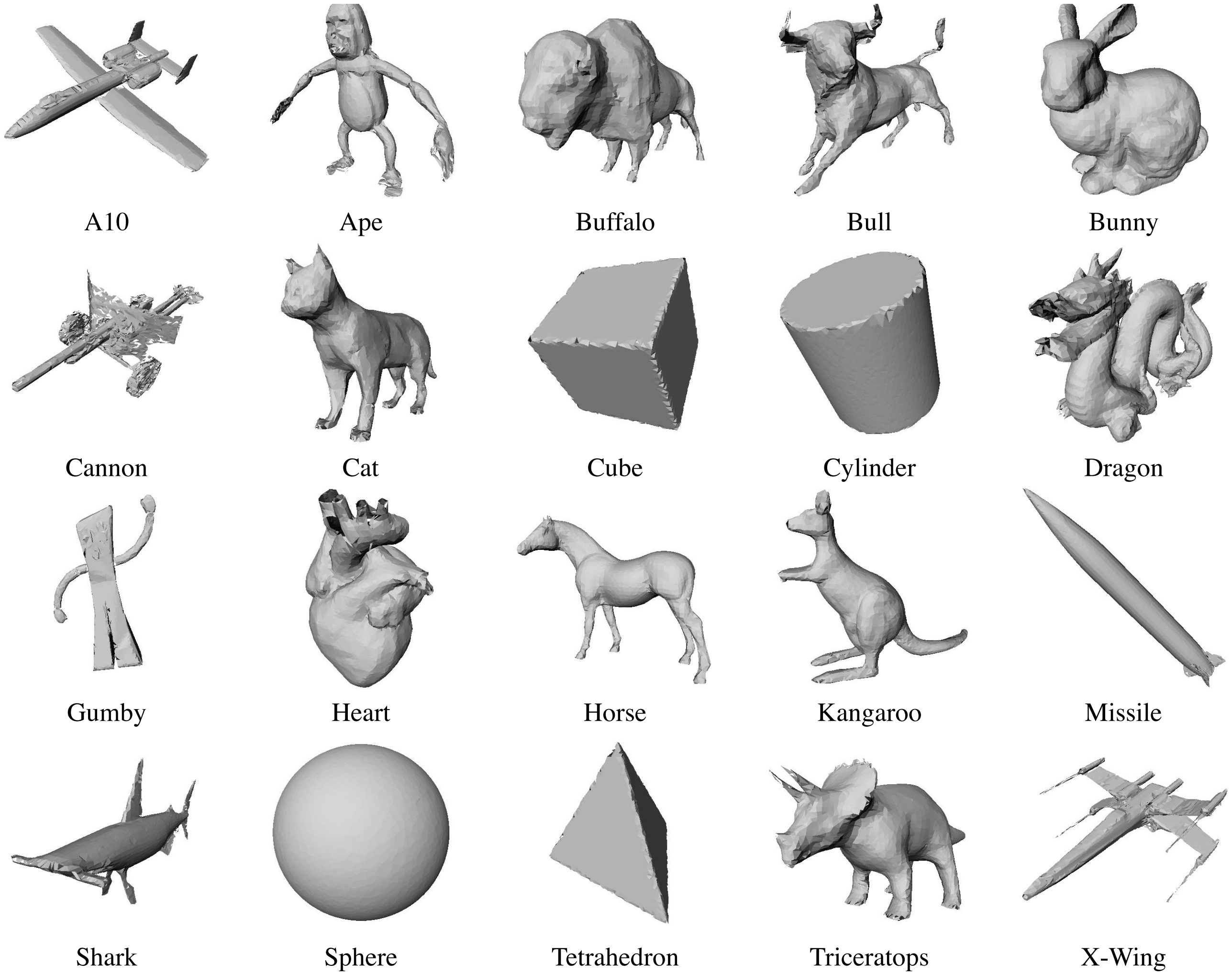
synthesized

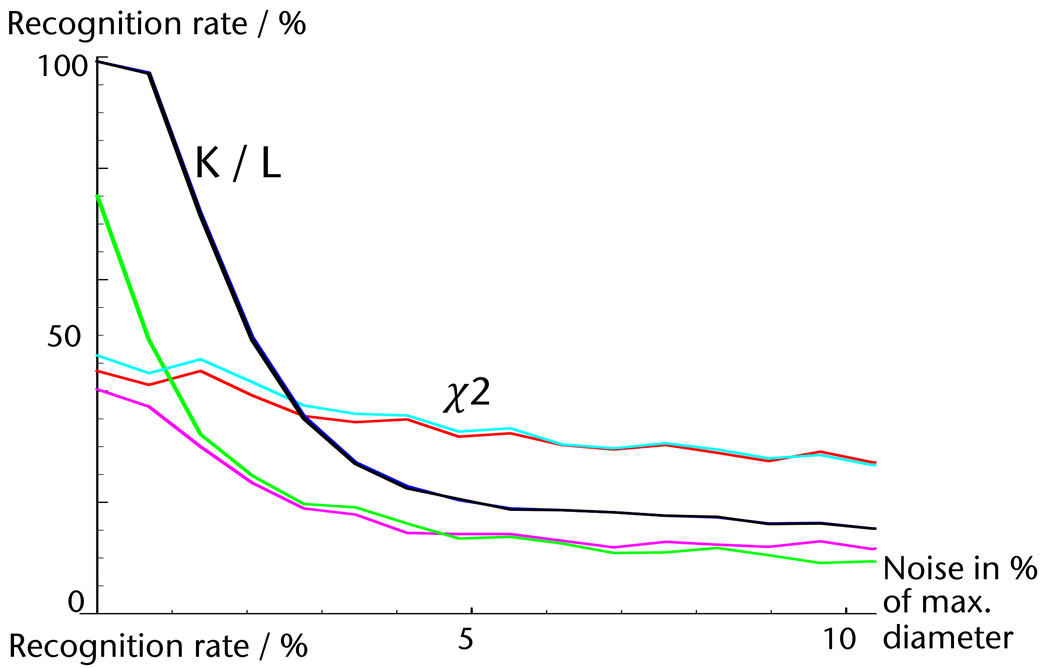


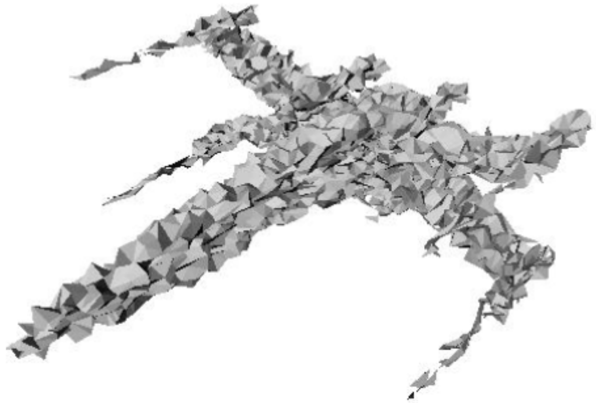


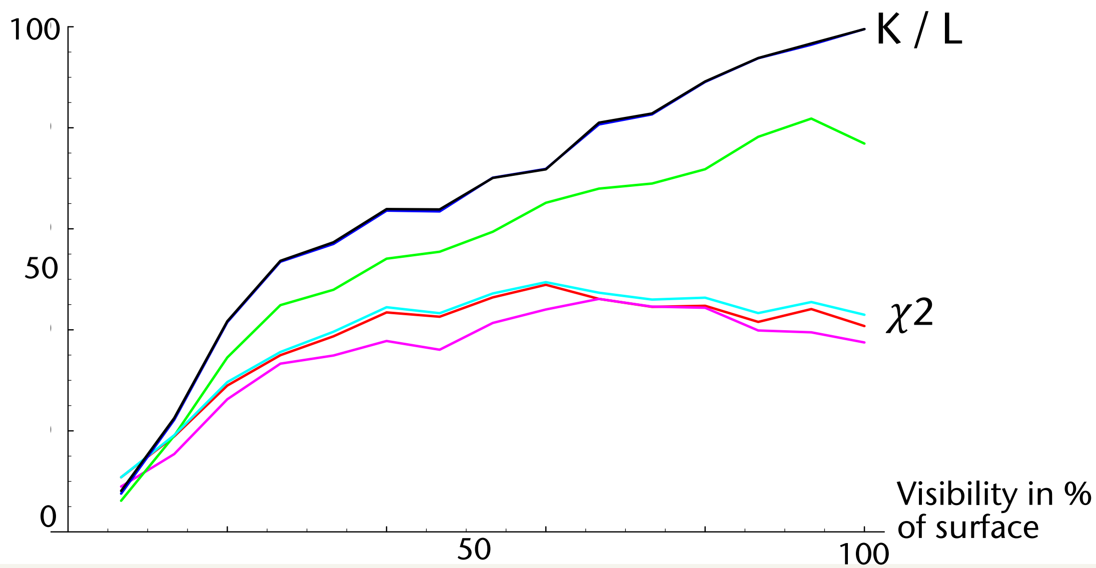
Experiments and Results Regarding Surflet-Pair Histograms

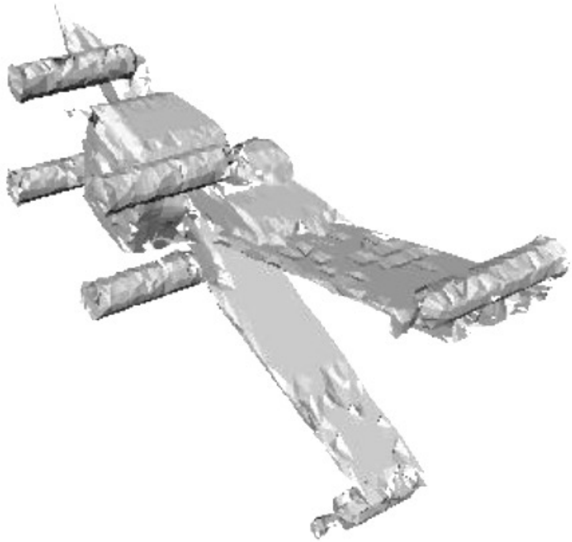
Test Objects




Recognition rate / %
100
50
0
K / L
$\chi 2$
5
10
Noise in % of max. diameter

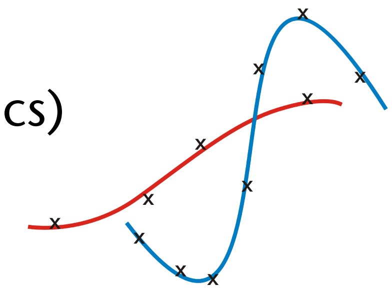



Recognition rate / %
100
50
0
K / L
$\chi 2$
50
100
Visibility in % of surface



Point Cloud Surfaces

- Increasingly popular geometry representation
- Lots of sources of point clouds (laser scanners, Kinect et al., ...)
- Goal: surface definition that is ..
  - Quick to evaluate
  - Robust against noise
  - Smooth
- Applications:
  - Ray tracing (rendering)
  - Collision detection (physics)

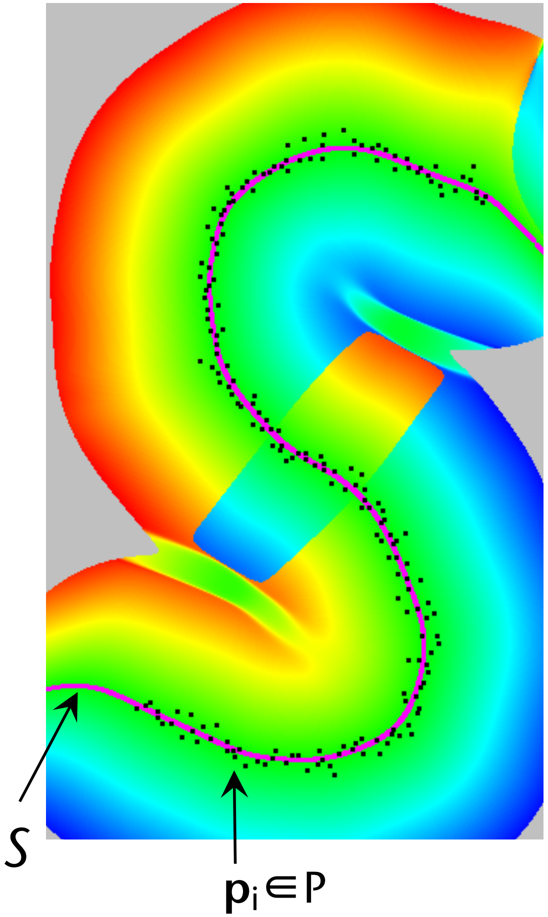


Approach

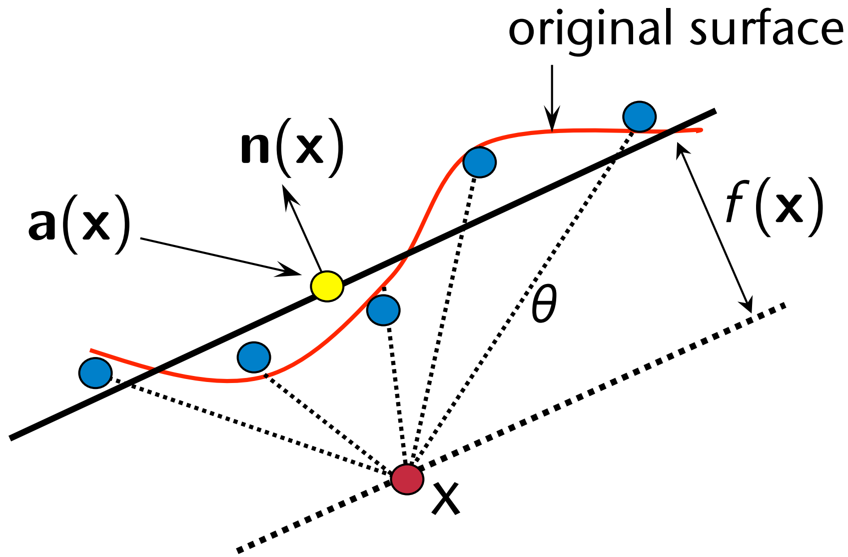
- Consider a point cloud *P* as *noisy* sampling of a smooth surface
  - Consequence: surface should *not interpolate* the points
- Define the surface as an implicit surface over a smooth distance function *f*, determined by the point cloud *P*:

$$S = \{\mathbf{x} | f(\mathbf{x}) = 0\}$$

where *f* is the distance to the yet unknown surface *S*



- Define f using weighted moving least squares



- The surface is approximated *locally* by a plane with

$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \theta(\|\mathbf{x}-\mathbf{p}_i\|)\mathbf{p}_i}{\sum_{i=1}^{N} \theta(\|\mathbf{x}-\mathbf{p}_i\|)}$$

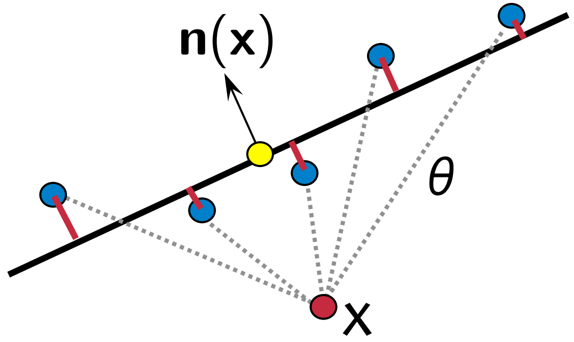
where θ is an *appropriate* weight function based on "distance"

- Overall:

$$f(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{x})$$

- Choose **n** as

$$\min_{\mathbf{n},\|\mathbf{n}\|=1} \sum_{i=1}^{N} \left(\mathbf{n}\cdot(\mathbf{a}(\mathbf{x}) - \mathbf{p}_i)\right)^2 \theta(\|\mathbf{x} - \mathbf{p}_i\|)$$



- Reminder: **n** happens to be the *smallest eigenvector* of the weighted covariance matrix B = (b_{ij}) with

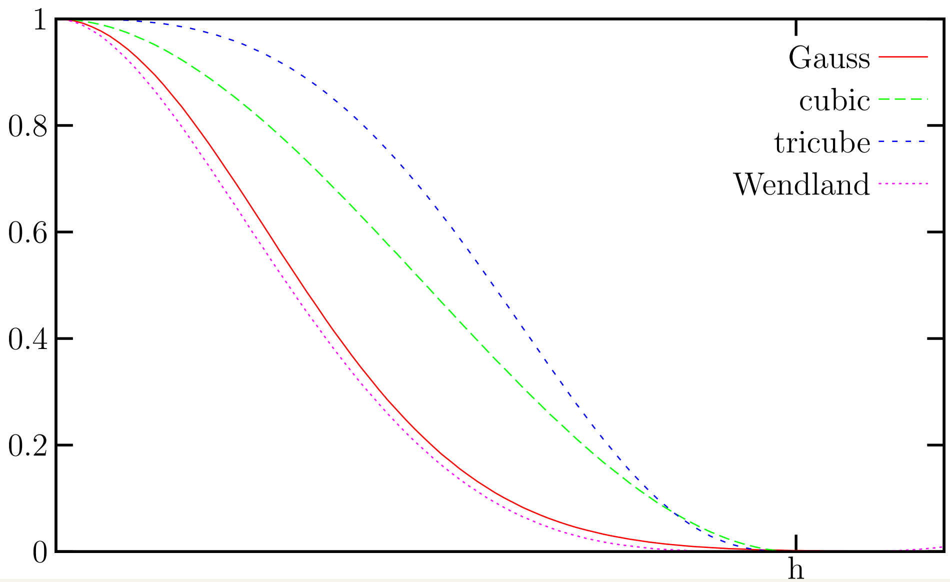
$$b_{ij} = \sum_{k=1}^{N} \theta(\|\mathbf{x} - \mathbf{p}_k\|)(p_{k,i} - a_i)(p_{k,j} - a_j)$$

- For the weight function , use (for now) a Gaussian kernel

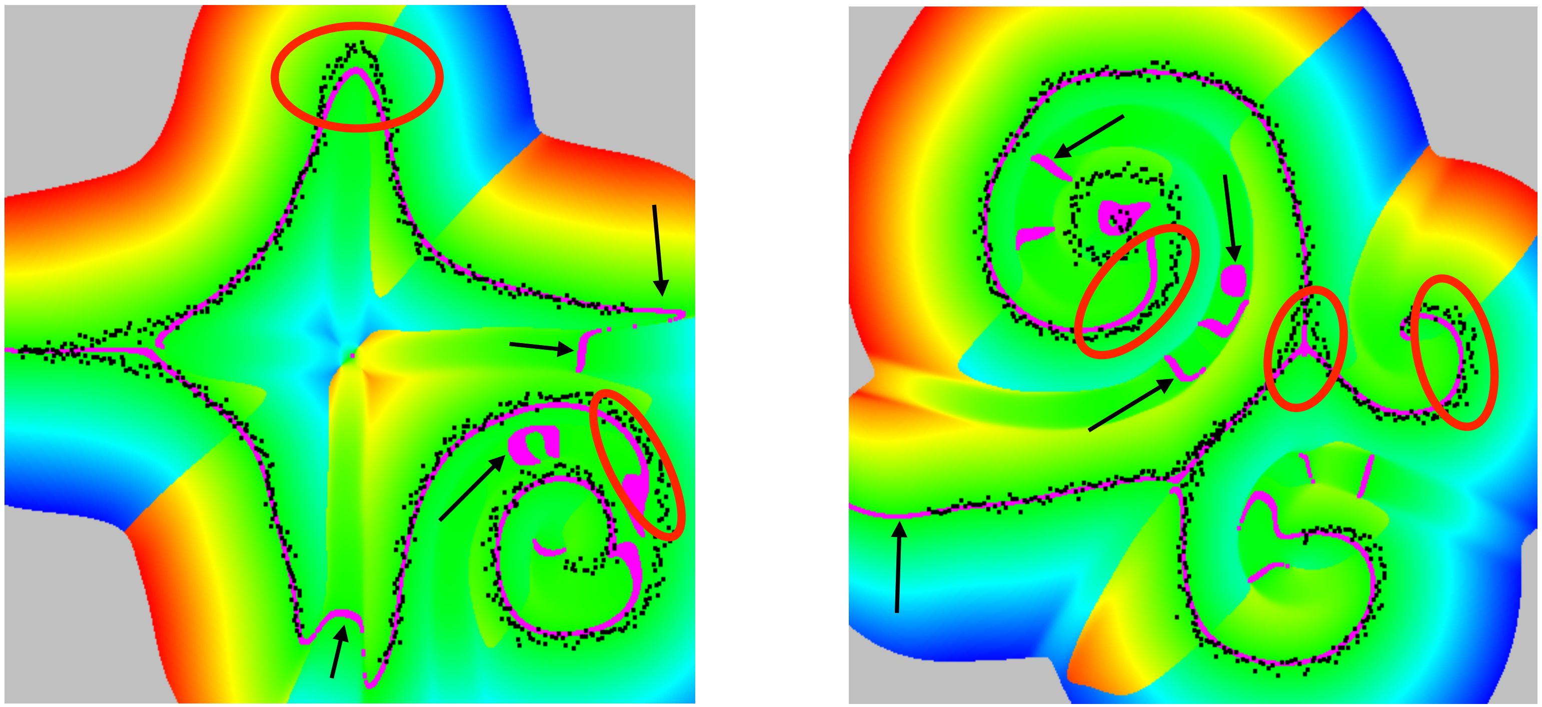
$$\theta(d) = e^{-d^2/h^2}, \quad d = \|\mathbf{x} - \mathbf{p}\|$$

with Euclidean distance (for now), where h is called bandwidth

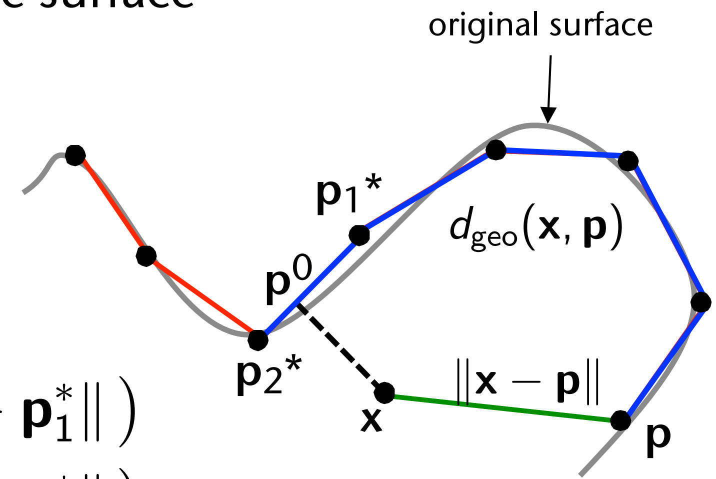
- Possible weight functions (kernels):
  - Gauß kernel
  - The cubic polynomial $\theta(d) = 2\left(\frac{d}{h}\right)^3 - 3\left(\frac{d}{h}\right)^2 + 1$
  - The tricube function $\theta(d) = \left(1 - \left|\frac{d}{h}\right|^3\right)^3$
  - The Wendland function $\theta(d) = \left(1 - \frac{d}{h}\right)^4 \left(4\frac{d}{h} + 1\right)$



- Whatever kernel you use, it is fine to consider only "close neighbors" around **x** in the computation of **a**(**x**) and **n**(**x**)
  $\longrightarrow$ need lots of k-NN searches in P
- More important: what distance measure to use in $\theta(\|\mathbf{x} - \mathbf{p}_i\|)$?
- Euclidean distance produces artefacts like this:



- Solution: use topology-based distance measure
  - Try to mimic the the geodesic distance on the surface
  - Except without knowing the surface yet
- Use proximity graph over point cloud
- Define



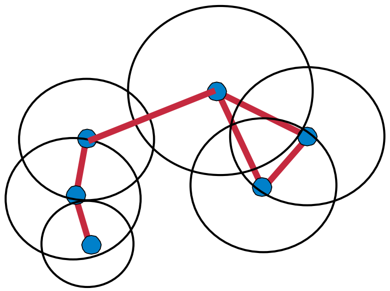
$$d_{\text{geo}}(\mathbf{x}, \mathbf{p}) = (1-a)\cdot\left(d(\mathbf{p}_1^*, \mathbf{p}) + \|\mathbf{p}^0 - \mathbf{p}_1^*\| \right) + a\cdot\left(d(\mathbf{p}_2^*, \mathbf{p}) + \|\mathbf{p}^0 - \mathbf{p}_2^*\| \right)$$

with $a = \|\mathbf{p}^0 - \mathbf{p}_1^*\|$
and $d(\mathbf{p}_i^*, \mathbf{p})$ = length of shortest path through proximity graph

- Note: don't add $\|\mathbf{p}^0 - \mathbf{x}\|$

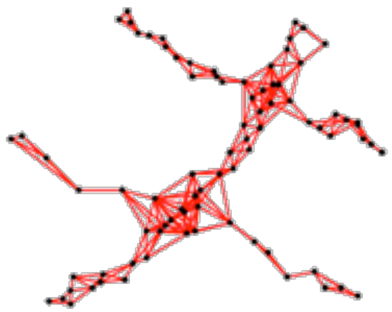
Which Proximity Graph to Use

- Many kinds of proximity graphs
  - Delaunay graph (explained later)
    - Needs kind of a "pruning" because of "long" edges; still has problems
  - Most other proximity graphs are subgraphs of the Delaunay graph
  - Sphere-of-Influence graph (SIG; is not a subgraph of the DG)
- Definition of the SIG:
  - For each point $\mathbf{p}_i \in P$ define $r_i = \|\mathbf{p}_i - \mathrm{NN}(\mathbf{p}_i)\|$
  - Connect $\mathbf{p}_i$ and $\mathbf{p}_j$ by an edge iff $\|\mathbf{p}_i - \mathbf{p}_j\| \leq r_i + r_j$
- Extension: k-SIG
  - Define $r_i = \|\mathbf{p}_i - k\mathrm{NN}(\mathbf{p}_i)\|$

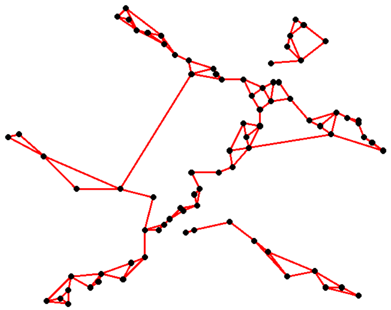


Results

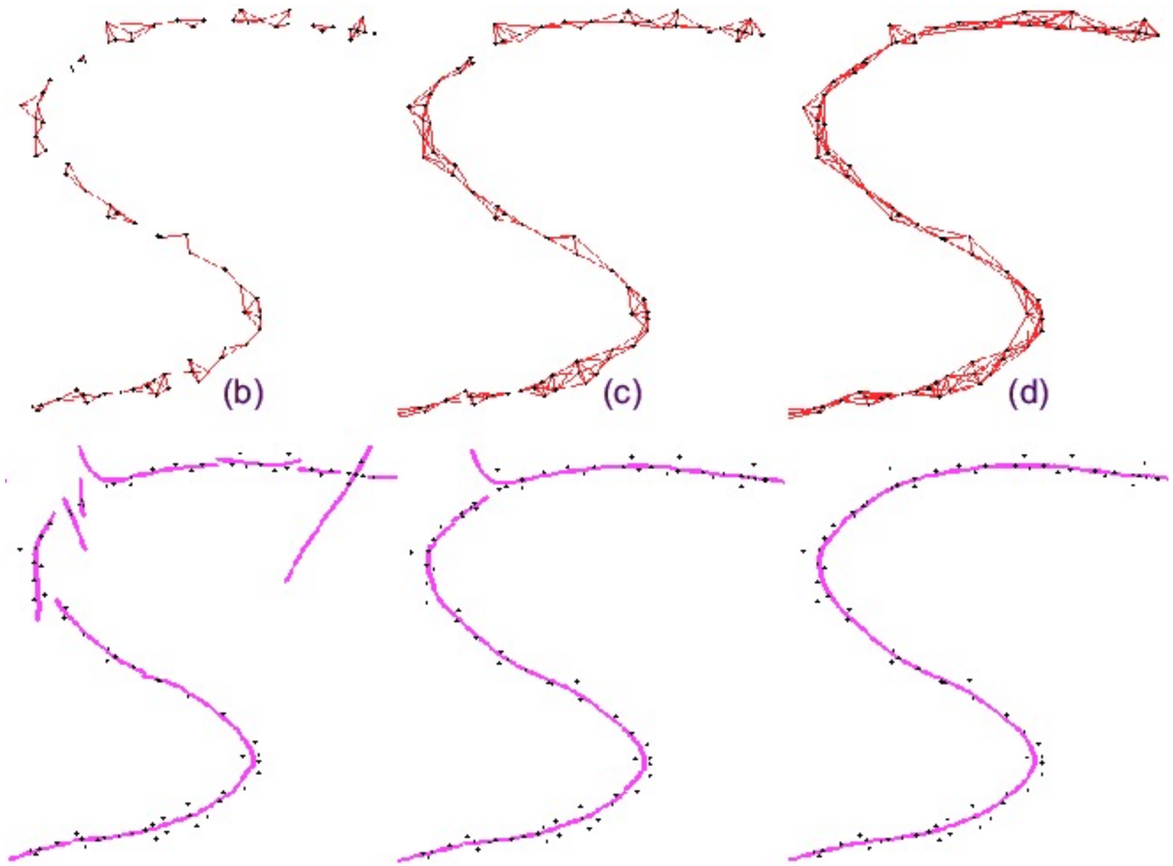
Example sphere-of-influence graph



DG w/ pruning



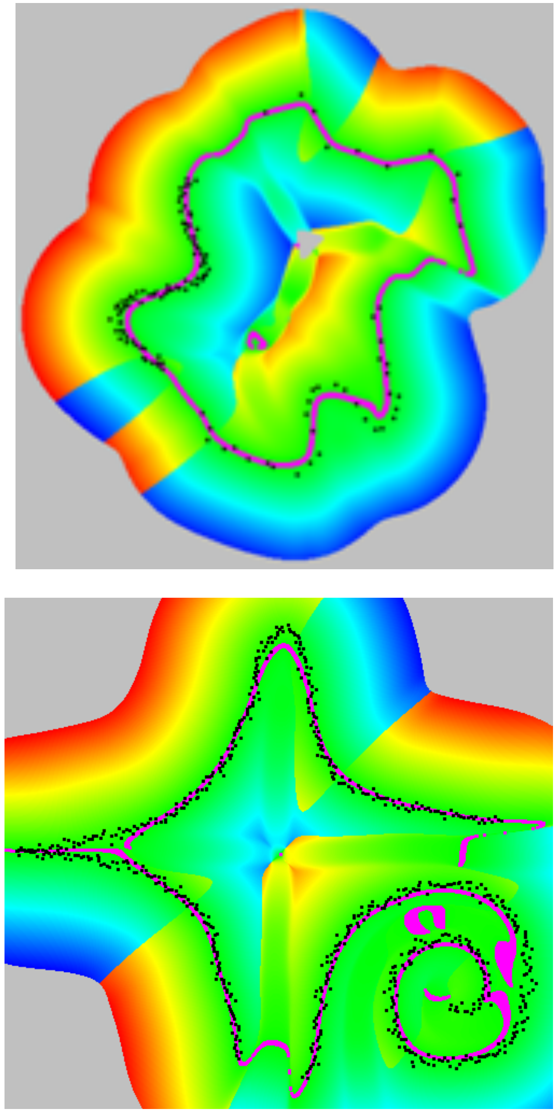
Weighted MLS surfaces using different k-SIGs for the geodesic distance

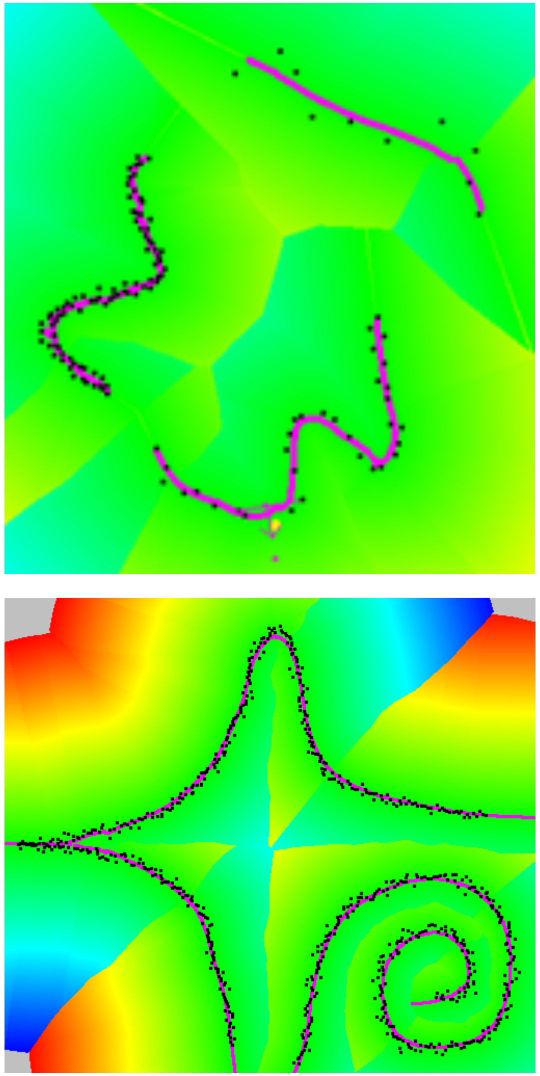


1-SIG | 2-SIG | 3-SIG

Weighted MLS surface
with Euclidean distance
and fixed bandwidth in kernel

Weighted MLS surface
with proximity graph-based distance
and automatic bandwidth estimation in kernel





More info in [Klein & Zachmann, 2004] on cgvr.cs.uni-bremen.de → Publications

Kurzer Exkurs über Quaternionen

[Hamilton, 1843]

- Erweiterung der komplexen Zahlen (geht leider nicht kommutativ):

$$\mathbb{H} = \left\{ q \mid q = w + a\cdot\mathbf{i} + b\cdot\mathbf{j} + c\cdot\mathbf{k} \;,\;\; w, a, b, c \in \mathbb{R} \right\}$$

- Alternative Schreibweise:

$$q = (\, w, \mathbf{v} \,)$$

- Axiome für die 3 imaginären Einheiten:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

$$(\mathbf{ij})\mathbf{k} = \mathbf{i}(\mathbf{jk})$$

- Daraus folgen sofort diese Rechengesetze:

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k} \qquad \mathbf{jk} = -\mathbf{kj} = \mathbf{i} \qquad \mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$

Rechenregeln für Quaternionen

- Addition: $q_1 + q_2 = (w_1 + w_2) + (a_1 + a_2)\mathbf{i} + (b_1 + b_2)\mathbf{j} + (c_1 + c_2)\mathbf{k}$
- Multiplikation:

$$
\begin{aligned}
q_1 \cdot q_2 &= (w_1 + a_1\mathbf{i} + b_1\mathbf{j} + c_1\mathbf{k}) \cdot (w_2 + a_2\mathbf{i} + b_2\mathbf{j} + c_2\mathbf{k}) \\
&= (w_1 w_2 - a_1 a_2 - b_1 b_2 - c_1 c_2) + \\
&\quad (w_1 a_2 + w_2 a_1 + b_1 c_2 - c_1 b_2)\,\mathbf{i} + \\
&\quad (\quad \ldots \quad \ldots \quad)\,\mathbf{j} + \\
&\quad (\quad \ldots \quad \ldots \quad)\,\mathbf{k}
\end{aligned}
$$

- Konjugation: $q^* = w - a\mathbf{i} - b\mathbf{j} - c\mathbf{k}$
- Betrag (Norm): $|q|^2 = w^2 + a^2 + b^2 + c^2 = q \cdot q^*$
- Inverse eines Einheitsquaternions: $|q| = 1 \ \Rightarrow \ q^{-1} = q^*$

- Bemerkung: manchmal ist es zweckmäßig, die Multiplikation zweier Quaternionen auch mit Hilfe einer Matrix-Multiplikation darzustellen

$$\underset{\uparrow}{q_1 \cdot q_2} = \underbrace{\begin{pmatrix} w_1 & -a_1 & -b_1 & -c_1 \\ a_1 & w_1 & -c_1 & b_1 \\ b_1 & c_1 & w_1 & -a_1 \\ c_1 & -b_1 & a_1 & w_1 \end{pmatrix}}_{Q_1} \underset{\uparrow}{q_2} = \underbrace{\begin{pmatrix} w_2 & -a_2 & -b_2 & -c_2 \\ a_2 & w_2 & c_2 & -b_2 \\ b_2 & -c_2 & w_2 & a_2 \\ c_2 & b_2 & -a_2 & w_2 \end{pmatrix}}_{Q_2} \underset{\uparrow}{q_1}$$

Quaternionen-Mult. – nicht Skalarmult.!

Als Spaltenvektor geschrieben!

Spaltenvektor!

- Außerdem gilt:

$$q_1 \cdot q_2^* = \underset{\uparrow}{Q_2^*} q_1 = Q_2^{\mathsf{T}} q_1$$

Matrix zum Quaternion q_2^*

Einbettung des 3D-Vektorraumes in $\mathbb{H}$

- Den Vektorraum $\mathbb{R}^3$ kann man in $\mathbb{H}$ so einbetten:

$$\mathbf{v} \in \mathbb{R}^3 \;\mapsto\; q_v = (0, \mathbf{v}) \in \mathbb{H}$$

- Definition:
  Quaternionen der Form $(0, \mathbf{v})$ heißen reine Quaternionen (*pure quaternions*)

Darstellung von Rotationen mittels Quaternionen

- Gegeben sei Axis & Angle $(\varphi, \mathbf{r})$ mit $\|\mathbf{r}\| = 1$
- Definiere das dazu gehörige Quaternion als

$$q = \left(\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}\mathbf{r} \right) = \left(\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}r_x, \sin\frac{\varphi}{2}r_y, \sin\frac{\varphi}{2}r_z \right)$$

- Beobachtung: $|q| = 1$

- Satz: Rotation mittels eines Quaternions

  Sei $\mathbf{v} \in \mathbb{H}$ ein pures Quaternion (= Vektor in 3D) und $q \in \mathbb{H}$ ein Einheitsquaternion. Dann beschreibt die Abbildung

$$\mathbf{v} \mapsto q \cdot \mathbf{v} \cdot q^* = \mathbf{v}'$$

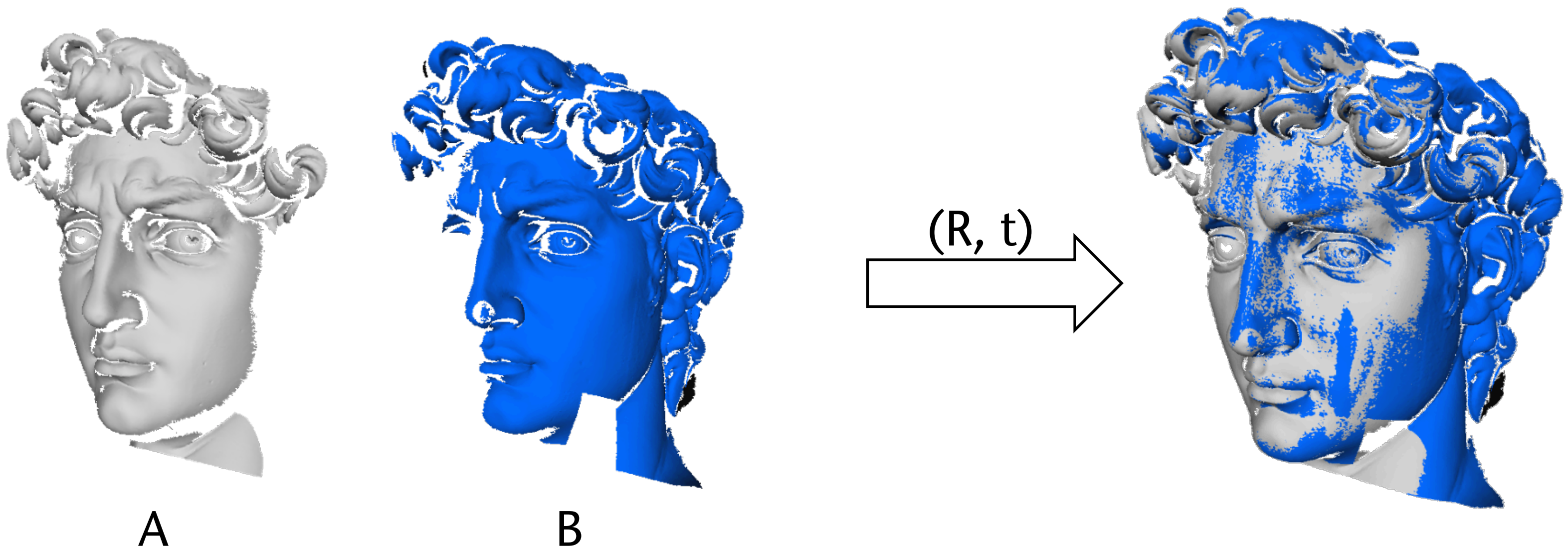
  eine (rechtshändige) Rotation von $\mathbf{v}$ um den Winkel φ und Achse $\mathbf{r}$ bestimmt sind, bei der das reine Quaternion $\mathbf{v}'$ entsteht.

Alignment / Registration of Shapes

- Siehe Manuskript

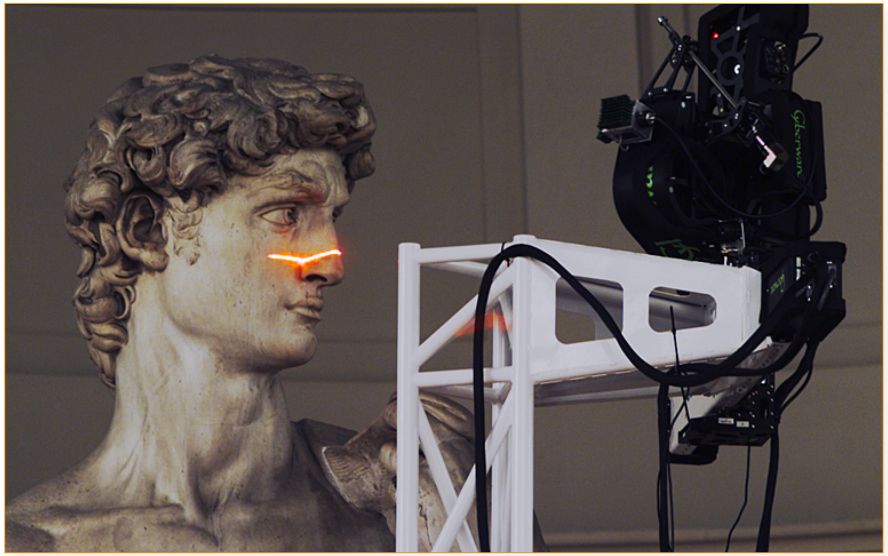
The Iterative Closest Point Algorithm

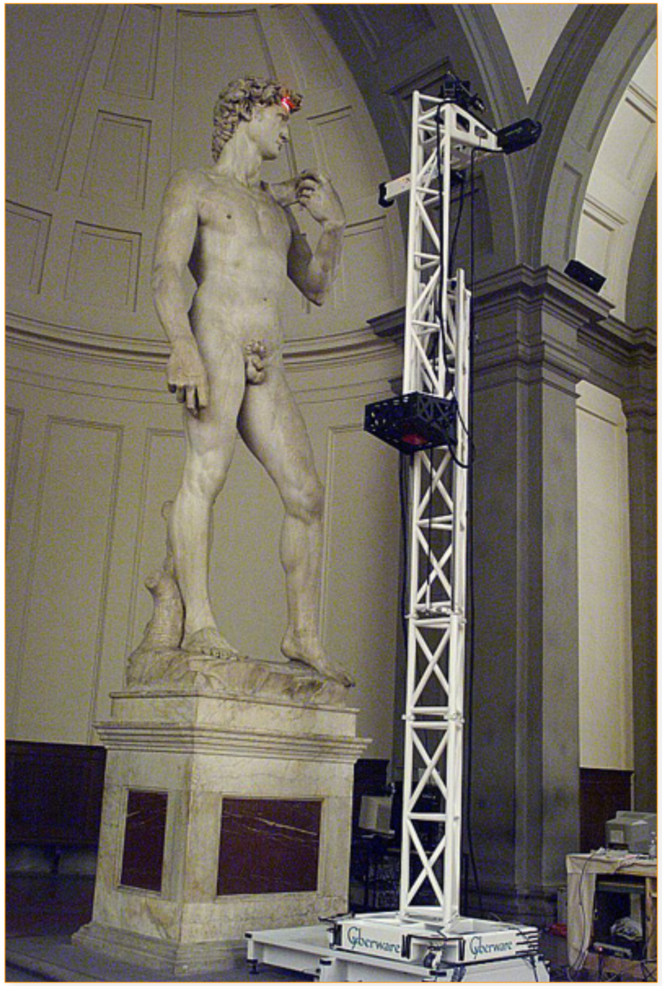
- Task:
  - Given two shapes (point clouds) A and B that partially overlap
  - Find a registration = rigid transformation (R, t) such that the squared distance between A and B is minimized



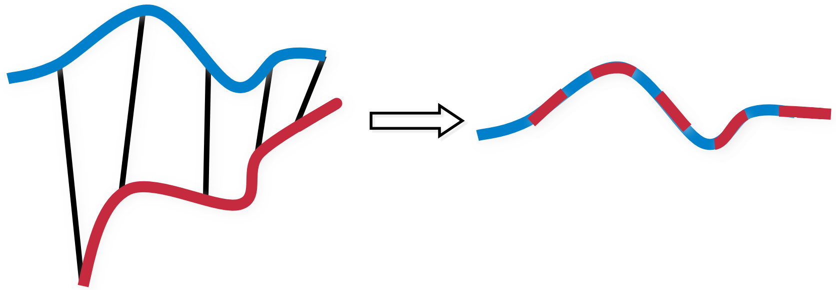
Motivation

- Registration of point clouds

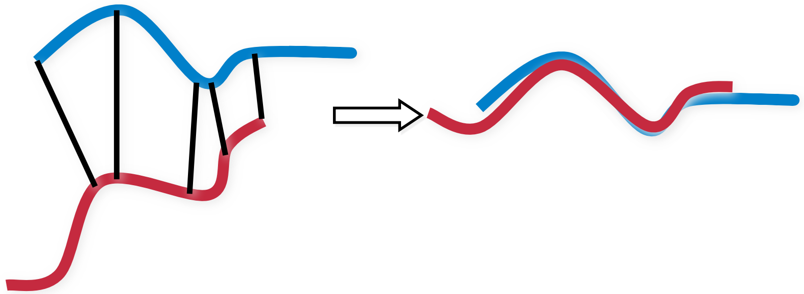




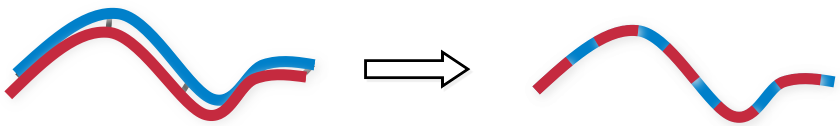
- We know: if correct correspondences are known, we can find correct relative rotation/ translation



- How to find correspondences: User input? Feature detection?
- Alternative: assume *closest points* correspond



- Converges (provably) provided initial position is "close enough"



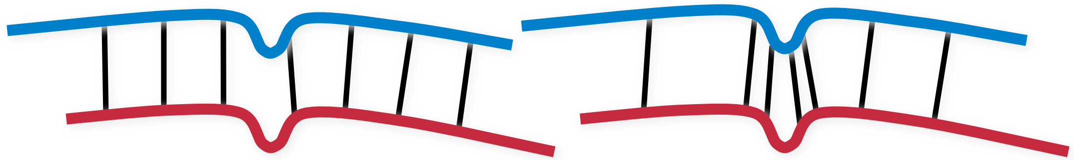
The Iterative Closest Point Algorithm (ICP)

```
repeat
  forall bi in B: find NN in A  ⟶  Y ⊆ A
  compute optimal alignment transformation (R,t) from B and Y
  B := R(B) + t
until error E² < threshold
```

- Optimization:
  - When starting the kd-tree traversal, initialize the candidate NN with the NN as of last iteration of the ICP
  - Makes the initial ball for the "ball overlaps bounds" test (hopefully) relatively small
  - The traversal does not descend into subtrees way off of the true NN

Variants / Optimizations

- Select only a sample of the points (of one or both shapes):
  - Uniform subsampling [Turk 94]
  - Random sampling in each iteration [Masuda 96]
  - Ensure that samples have normals distributed as uniformly as possible [Rusinkiewicz 01]



- Use other ways to establish correspondences:
  - Restrict matches to compatible points (color, intensity , normals , curvature, ...) [Pulli 99]

- Weight correspondences: replace the old least squares error measure by

$$E''^2 = q^\mathsf{T} \left(\sum_i w_i B_i^\mathsf{T} A_i \right) q$$

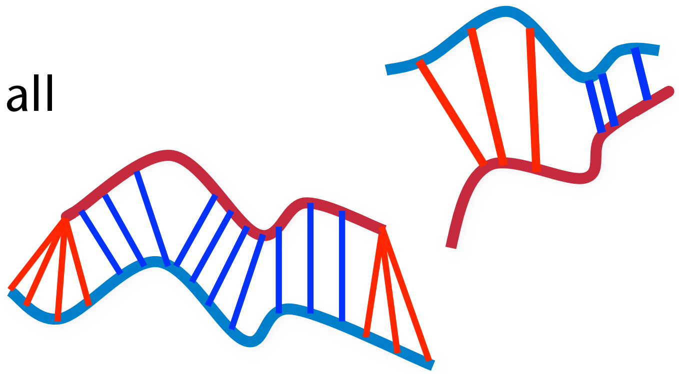
  - As weight, you could consider:
    - Distance between corresponding points

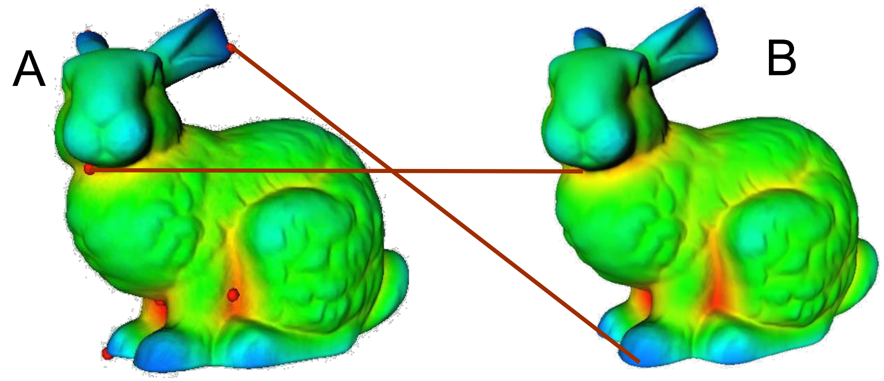
$$w_i = 1 - \frac{\|b_i - a_i\|}{\text{max dist}}$$

    - Scanner uncertainty

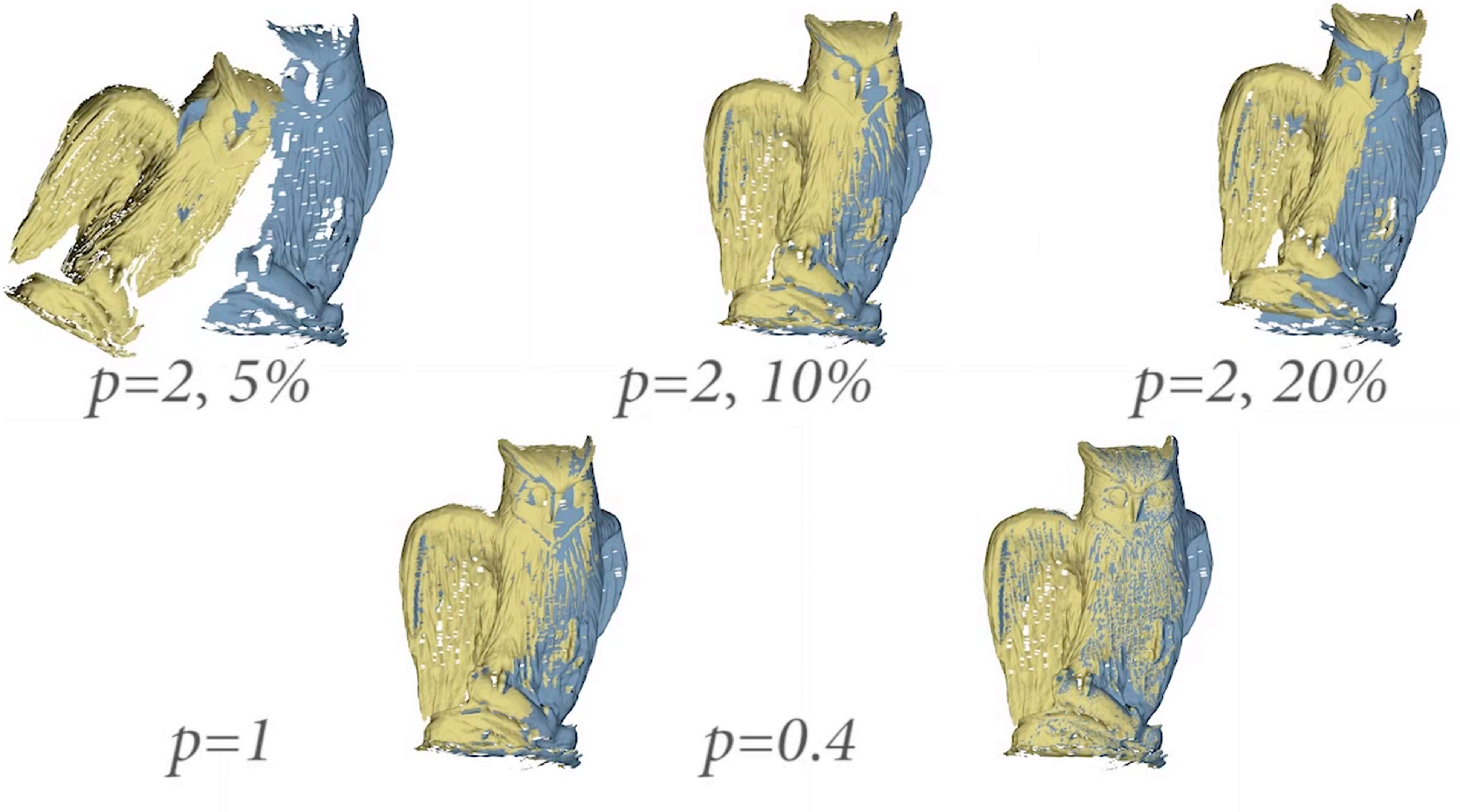
- Reject "bad" point pairs:
  - Reject pairs whose distance is in the top *x*% of all distances
  - Points on end vertices
  - Reject pairs that are *not consistent* with their neighboring pairs [Dorai 98]:
    - Two pairs (a_1,b_1) and (a_2,b_2) are not consistent if

$$\Big| \|a_1 - a_2\| - \|b_1 - b_2\| \Big| > \theta$$





Video



Sofien Bouaziz, Andrea Tagliasacchi, Mark Pauly: "Sparse Iterative Closest Point"
Symposium on Geometry Processing 2013

Stackless kd-tree traversal for ray-tracing

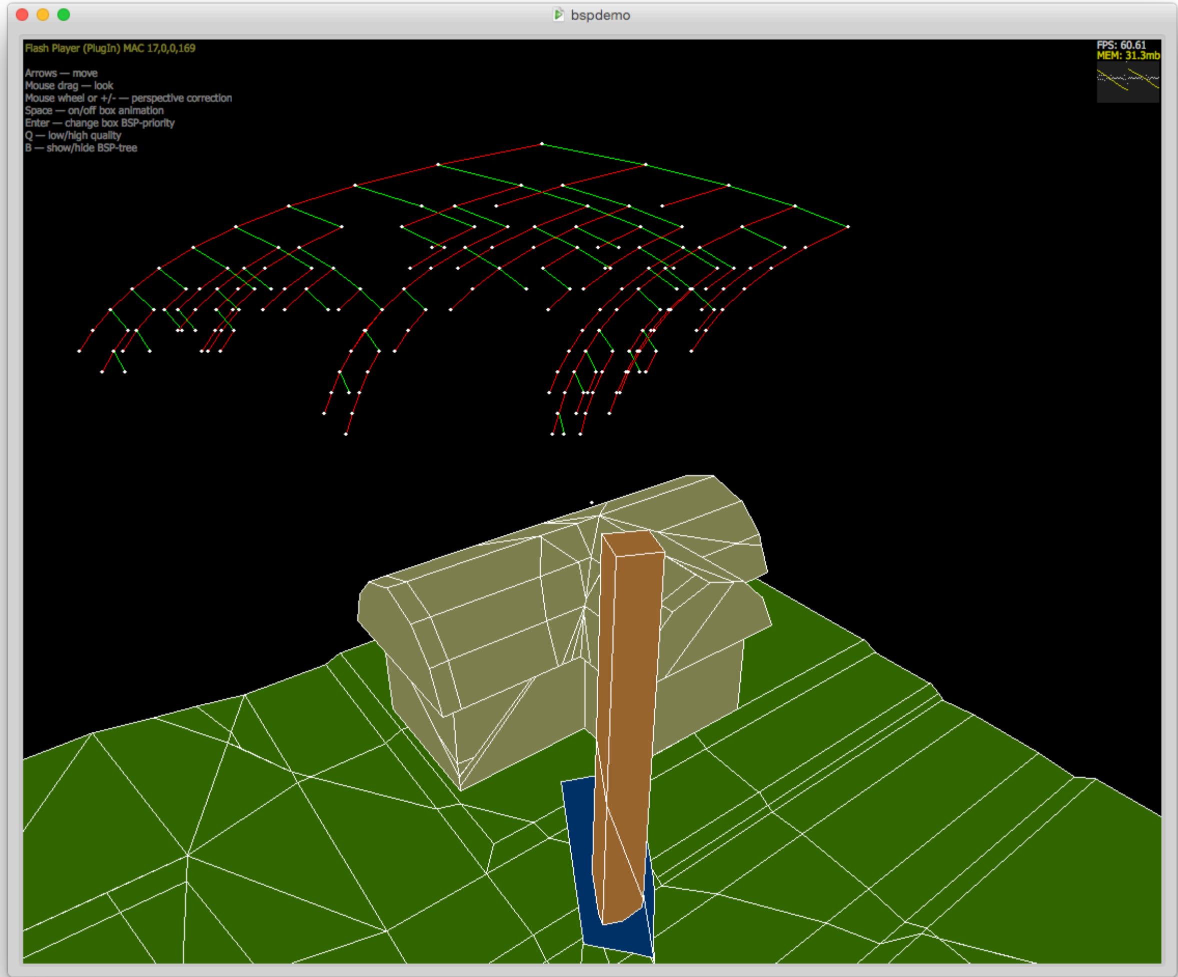
Stefan Popov, Johannes Günther, Hans-Peter Seidel, and Philipp Slusallek.
Nvidia GeForce 8800GTX, CUDA, 2007.

Daniel Horn, Jeremy Sugerman, Mike Houston, Pat Hanrahan
ATI X1900XTX, PixelShader 3.0, 2007

Real-Time KD-Tree Construction on Graphics Hardware

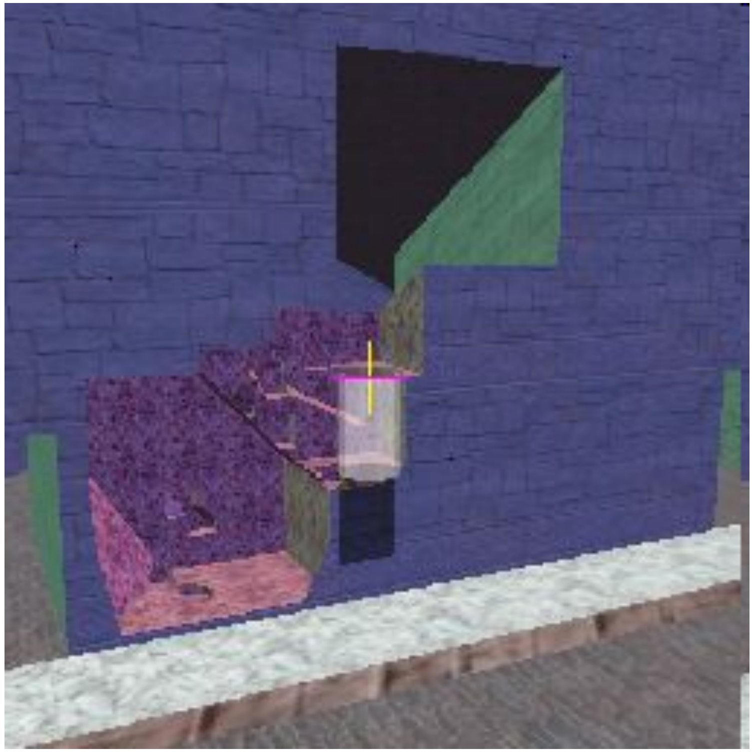
Kun Zhou, Qiming Hou, Rui Wang, Baining Guo; SIGGRAPH Asia 2008

BSP Demo


bspdemo
Flash Player (PlugIn) MAC 17,0,0,169
Arrows — move
Mouse drag — look
Mouse wheel or +/- — perspective correction
Space — on/off box animation
Enter — change box BSP-priority
Q — low/high quality
B — show/hide BSP-tree
FPS: 60.61
MEM: 31.3mb

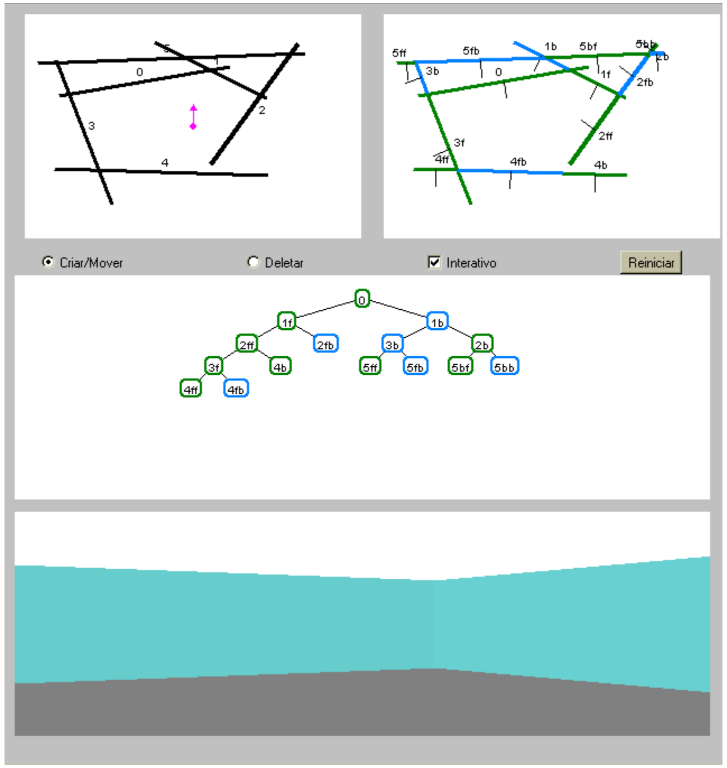
Applications of the BSP

[Boolen Operations]



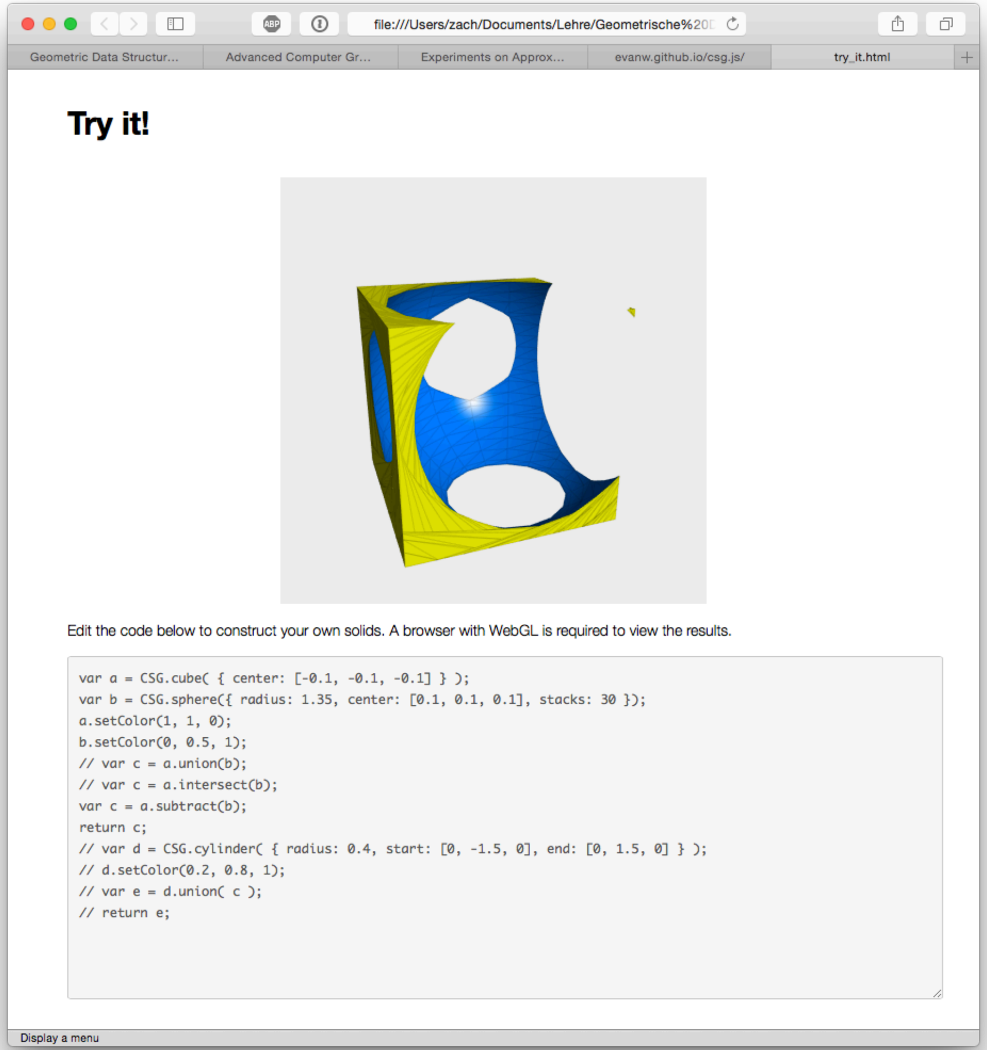
Stan Melax

[Painter's Algorithm]



Paton J. Lewis

With BSPs one can do CSG quite easily



http://evanw.github.io/csg.js/

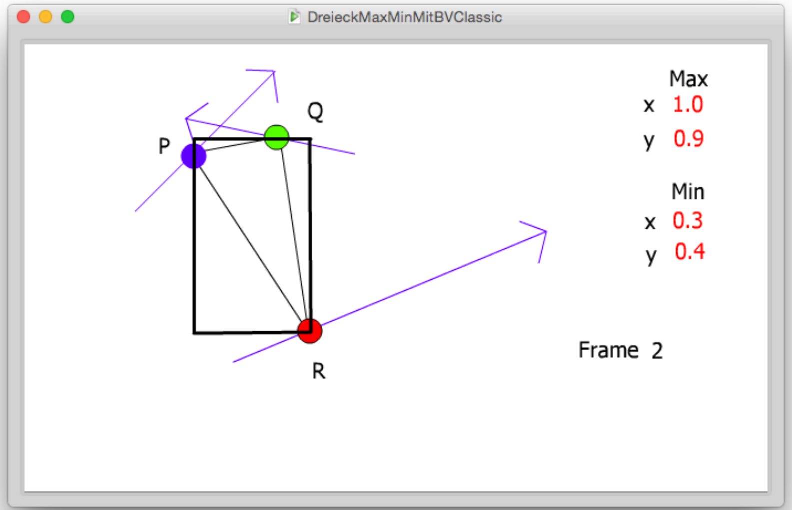
Shadow Volume Checking with BSPs

Q	Quit
1	Load 1st scene (simple room, 1 light source)
2	Load 2nd scene (random objects 1)
3	Load 3rd scene (simple room, 4 light sources)
4	Load 4th scene (cubes, 1 light source)
5	Load 5th scene (random objects 2)
W, A, S, D	Translate viewpoint
Cursor keys	Rotate viewpoint
+/-	Pan up/down
R	Reset current scene and rebuild BSP tree
L	Toggle labels
T	Toggle usage of BSP tree
U	Toggle depth buffer
E	Toggle shadows

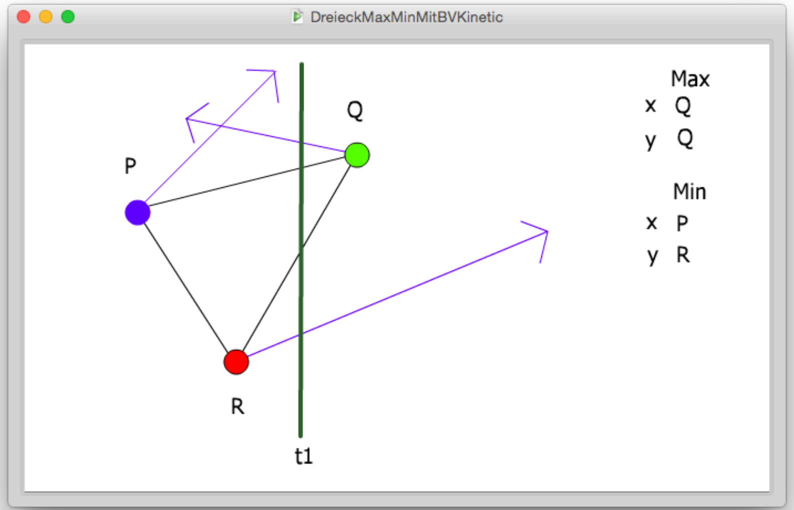
http://bastian.rieck.ru/uni/bsp/

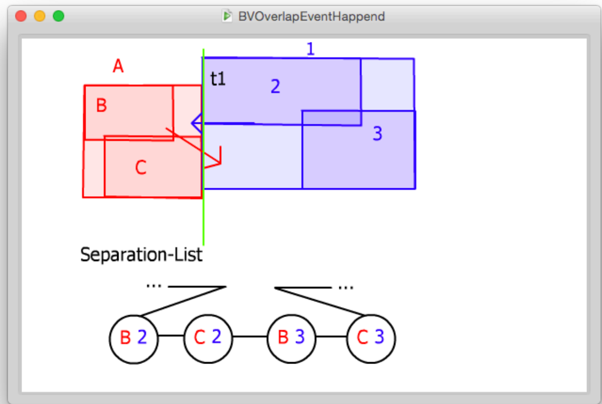
Kinetic Data Structures – Motivation

Brute force update of bbox



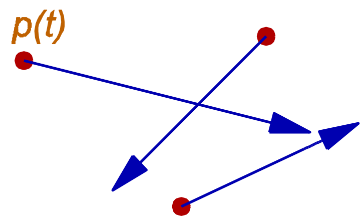
Kinetic update of bbox



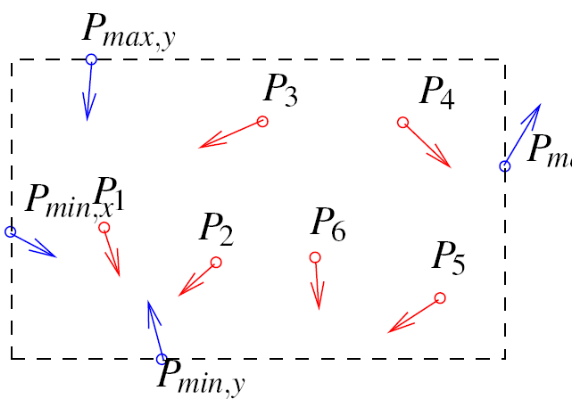


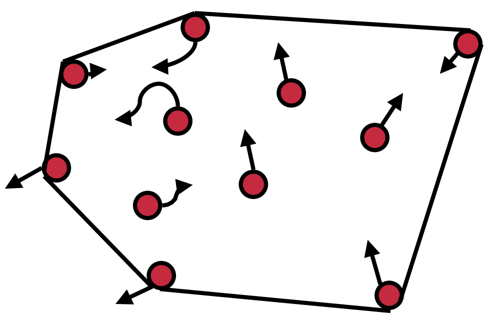
Kinetic Data Structures (in General)

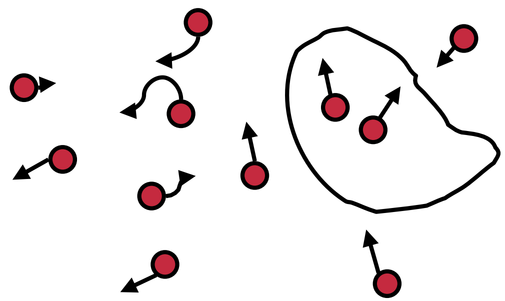
- Given:
  - A number of objects (points, lines, polygons, boxes, ...)
  - A flight path for each of these objects, given by an algebraic function
    - Mostly assume linear motion



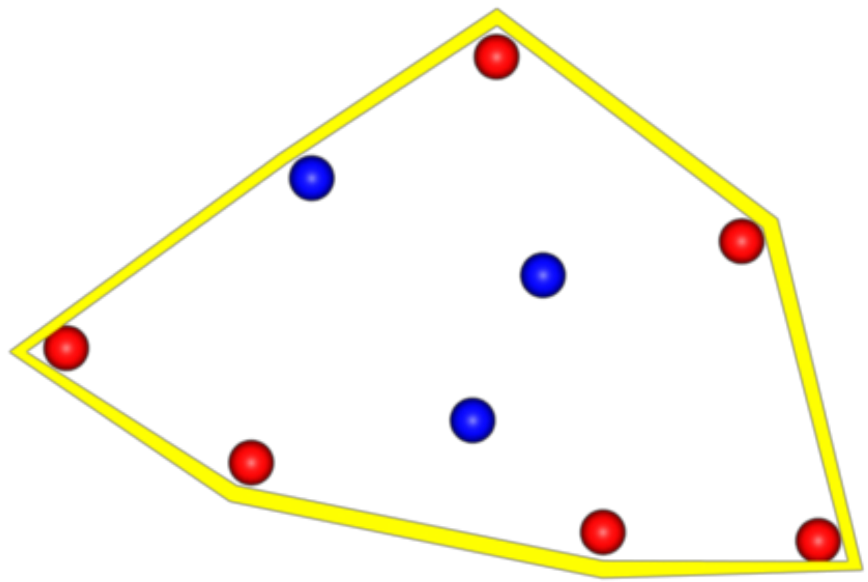
- Attribute = the task / purpose of a KDS
  - Examples: convex hull over a number of points, bbox of a number of points, kd-tree over a number of points, ...

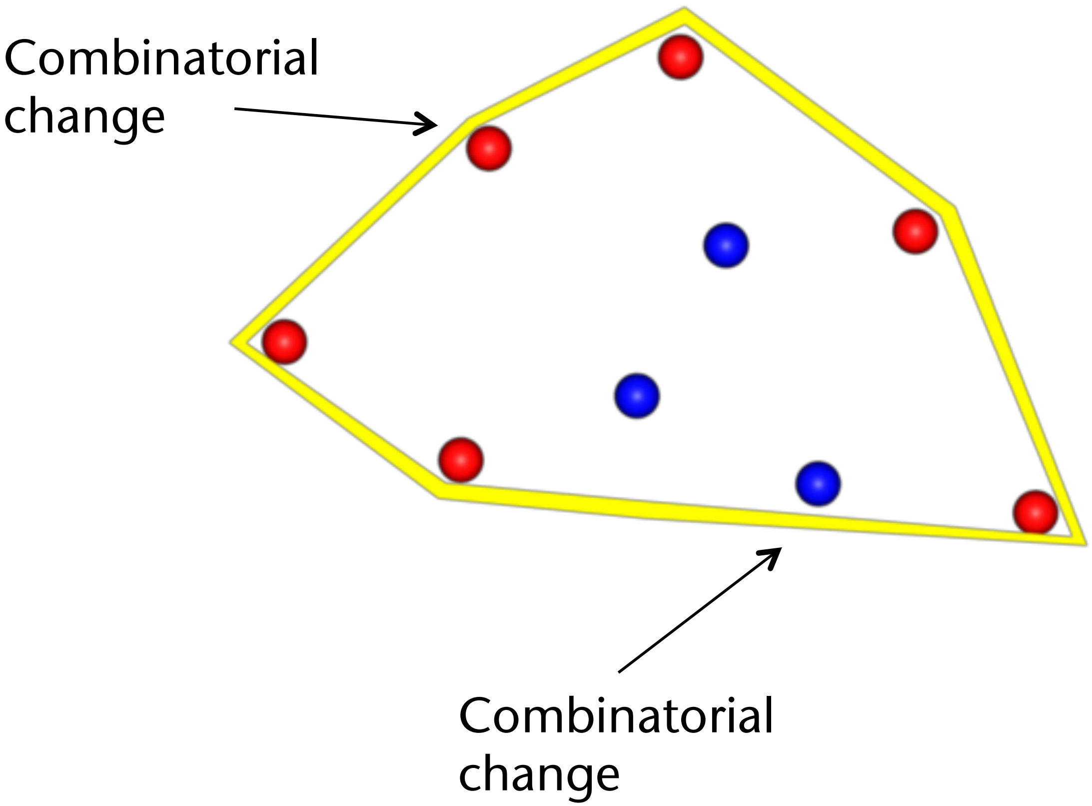


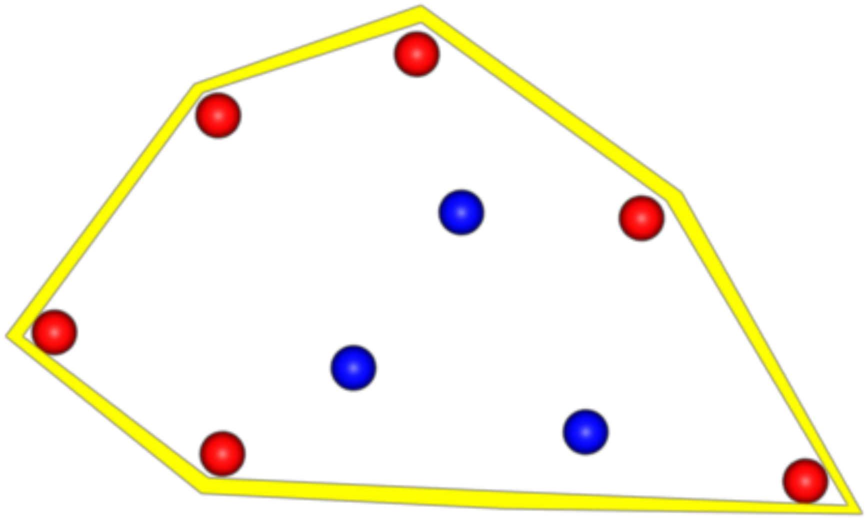


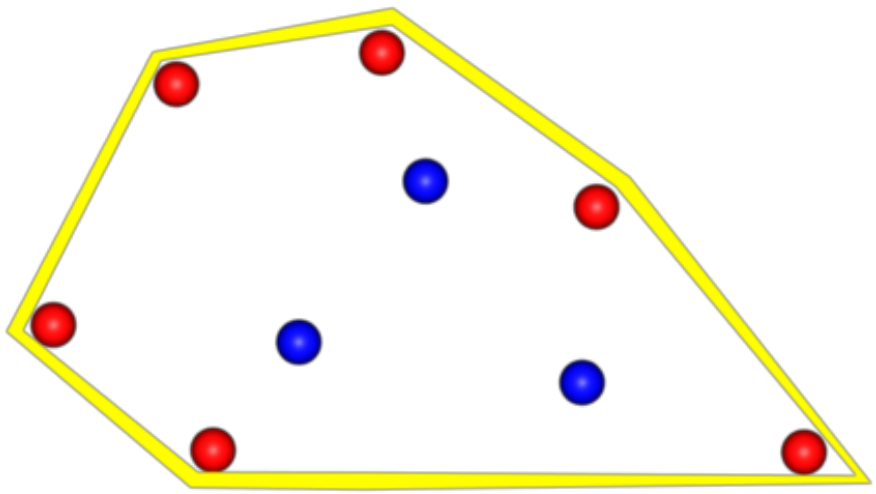


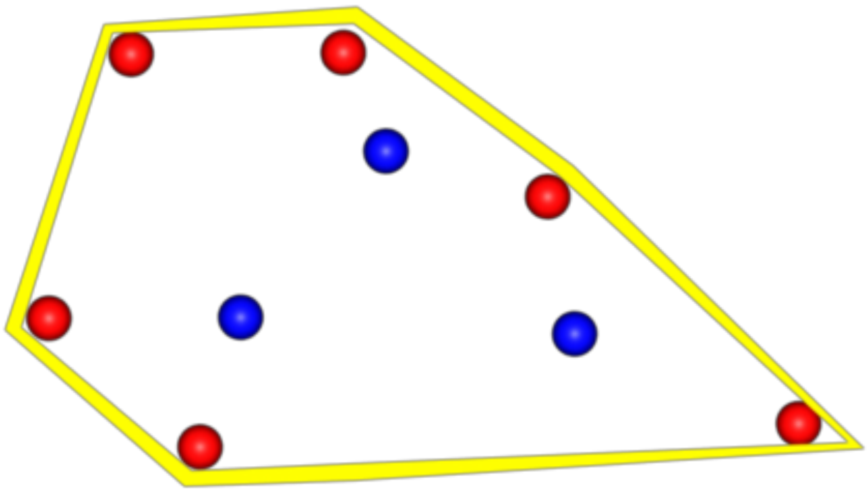
- Combinatorial structure = "everything that descriibes the attribute *except* concrete coordinates"
  - Examples:
    - Convex hull: those points that form the corners of the convex hull
    - Bbox: those points that realize the min/max at least on one of the coord axes
    - Kd-tree: all the pointers that make up the tree, and pointers to points

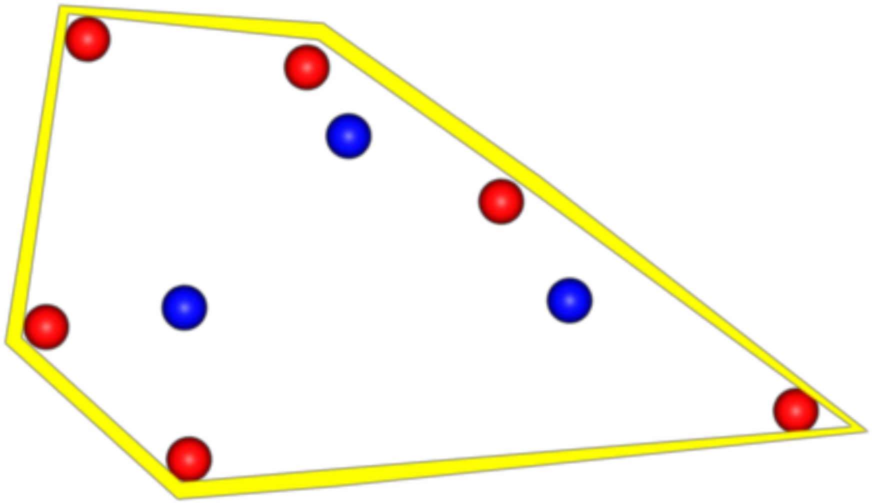


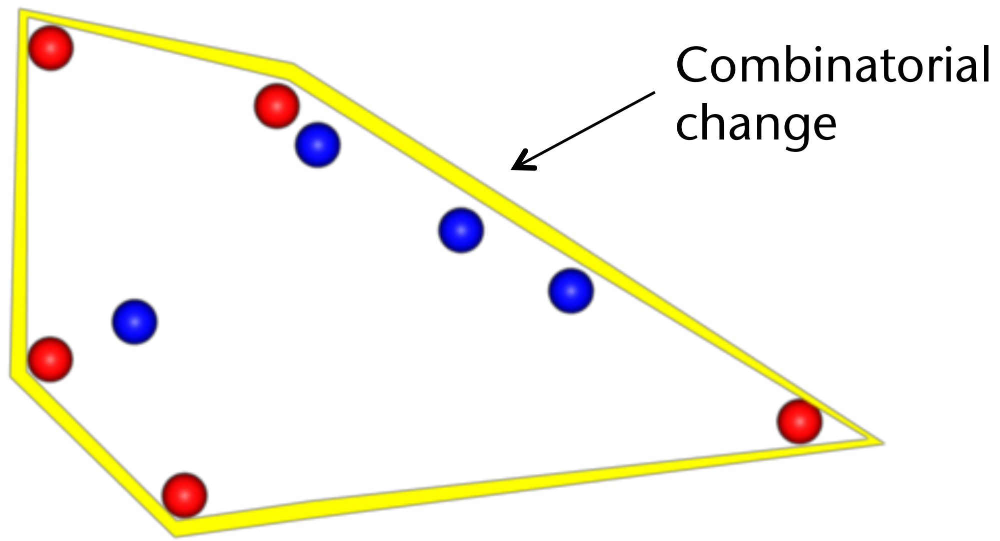

Combinatorial change
Combinatorial change

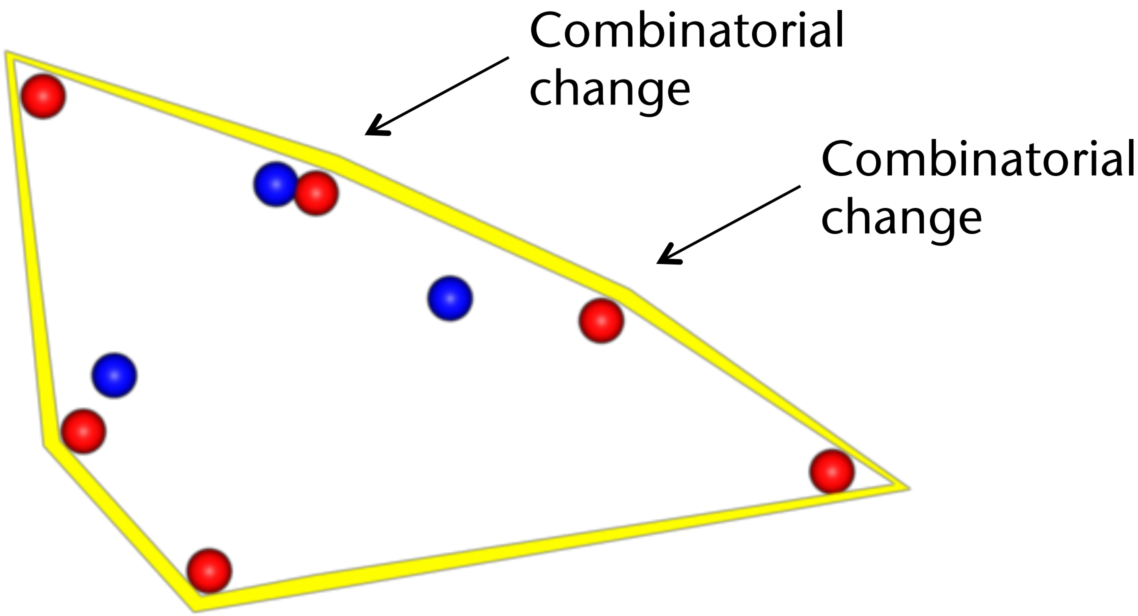


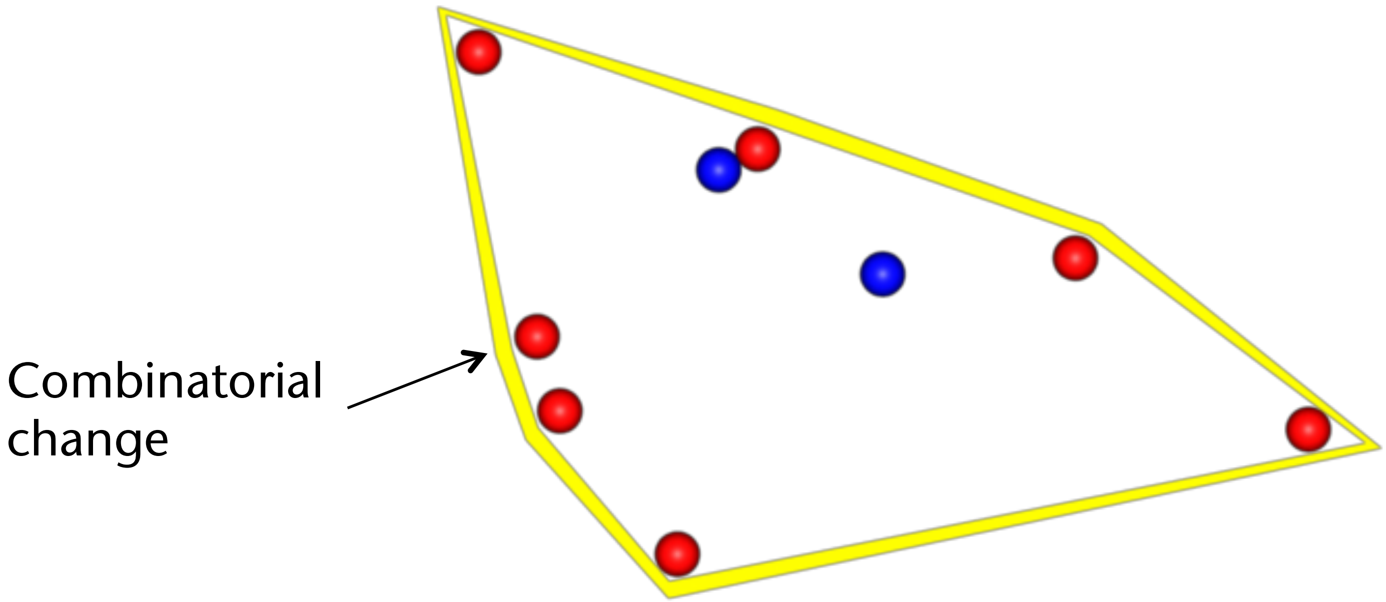


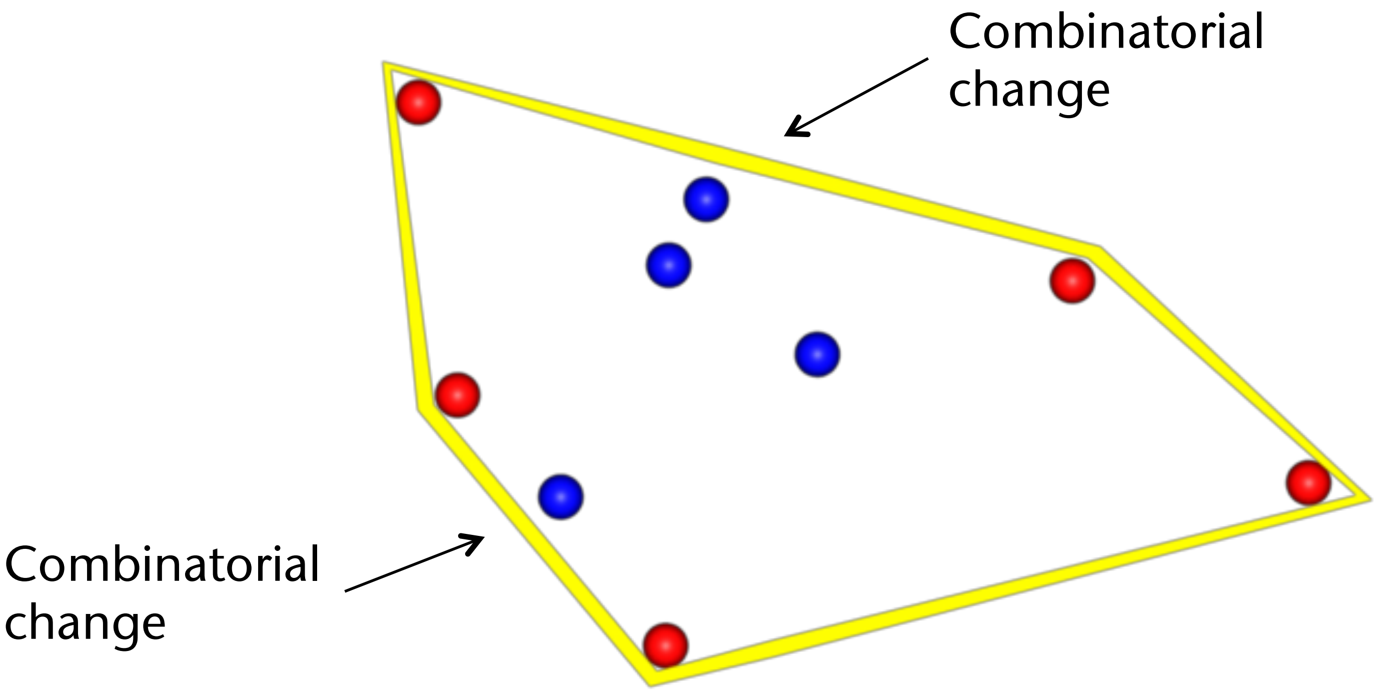





Combinatorial
change


Combinatorial
change
Combinatorial
change


Combinatorial
change


Combinatorial
change
Combinatorial
change

- Certificate = simple geometric relation (a.k.a. geometric predicate) involving a few of the objects
  - Example: $p \cdot n < 0$, where p is an input point and n is a normal
- Event: a specific point in the future where one of the certificates *fails*, i.e., its truth value is false, due to the motion of the objects
  - External event = event where the combinatorial structure of the attribute changes
  - Internal event = event where the combinatorial structure remains the same, but the set of certificates changes
- Kinetic data structure (KDS) for a geometric attribute =
  1. A set of certificates that a true whenever the combinatorial structure of the attribute is valid, as well as
  2. A set of rules for repairing the attribute and the set of certificates in case of an event

Main Loop of a KDS

```
Initialize the attribute for the input objects
Initialise the set of certificates for the attribute
Compute all events (failure times) of all certificates
   (usually only up to some time in the future)
Initialize the p-queue for all events, sorted by failure time
Loop forever
  get front event from the event queue
  if external event:
    change the attribute
  update the set of certificates:
    some failure times of later events might change
    some certificates may need to be deleted
    maybe, some new certificates need to be created
```

In reality, of course, a KDS does not have its own main loop usually ...

```
initialization ...
while simulation runs
  determine time t of next rendering
  get nearest event from the event queue
  while timestamp(event) < t:
    update KDS
    get next event from the event
  use the attribute of the KDS (e.g., bbox, kd-tree, BVH, …)
  render scene
```

Performance Measures for KDS

1. Responsiveness:
   A KDS is responsive, if the cost to update the set of certificates and the attribute in case of an event is "small"
   - Usually, "small" = $O(\log^s n)$ or $O(n^\varepsilon)$
2. Efficiency:
   A KDS is efficient, if the ratio of #(total events) / #(external events) is small
   - I.e., the #(internal events), where the attribute's combinatorial structure does not change, is small
   - I.e., the #events is comparable to the #(attribute changes) over time
3. Compactness:
   A KDS is compact, if the number of certificates is close to linear in the number of input objects

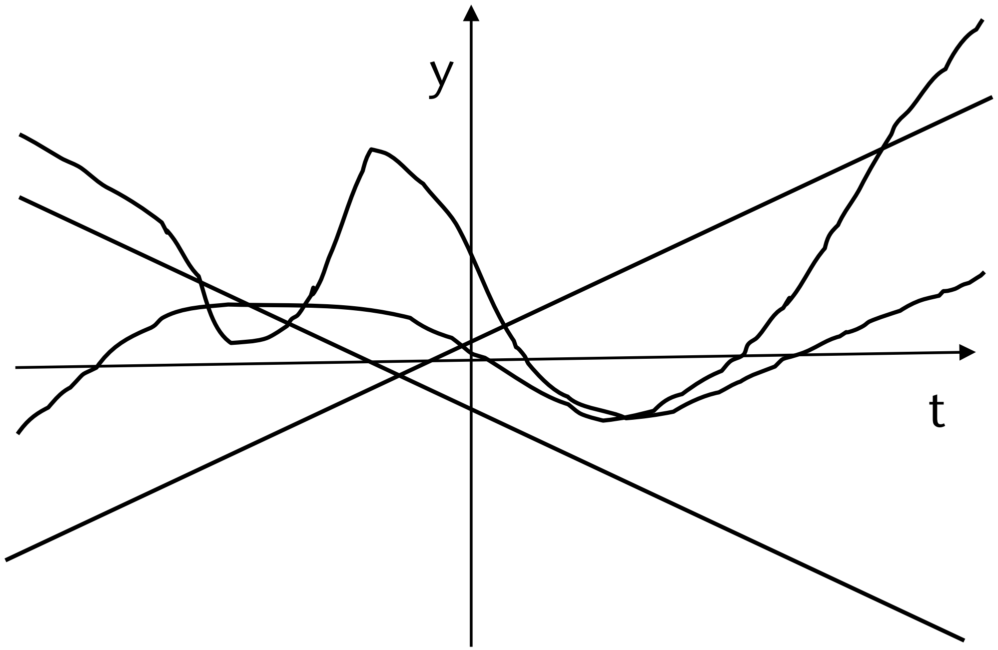
4. Locality:

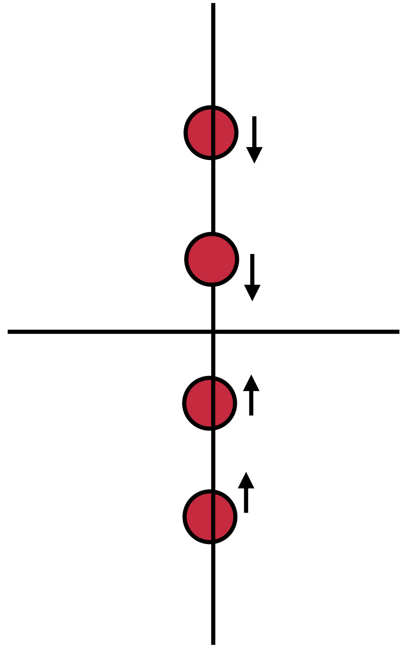
   A KDS is local, if all objects participate only in a small number of certificates

   - Advantage: if an object changes its flight path, then the cost for updating all events affected by it is not too high

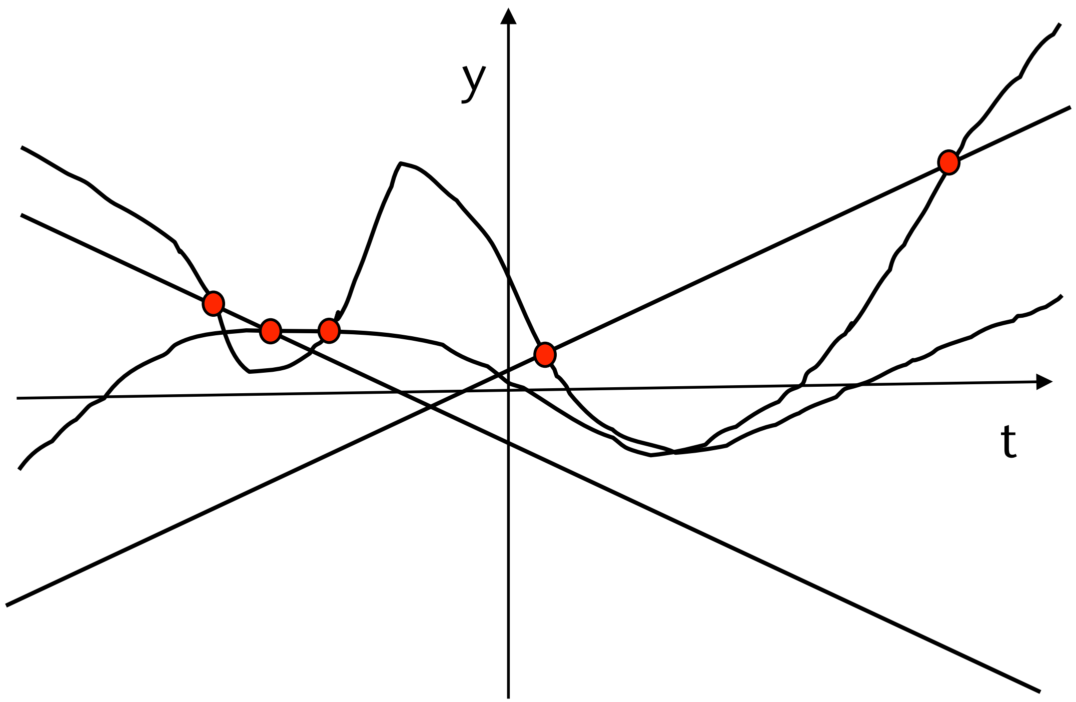
A Simple Example

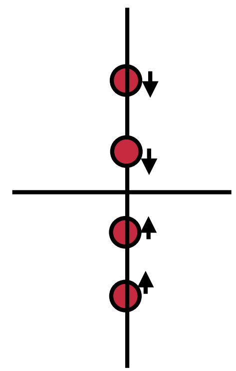
- Maintain the topmost among points moving along the y-axis
- Look at the *ty*-plane (flight paths)





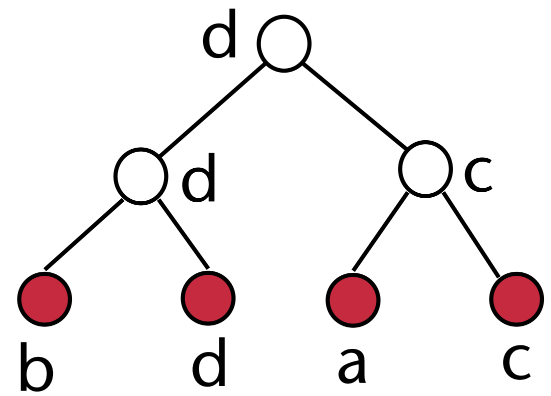
- We are interested in the *upper envelope*

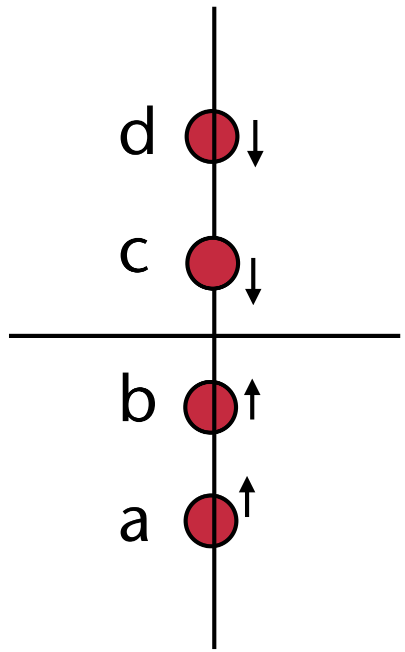




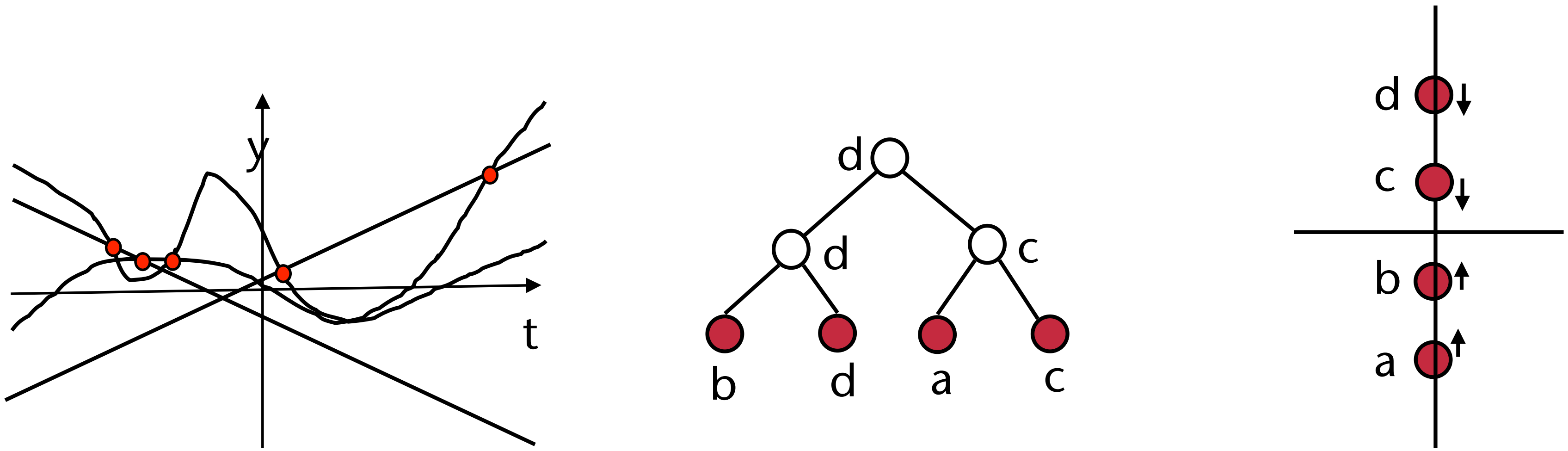
- Theorem (Sharir, Hart, Agarwal and others):
  If any pair of flight paths intersect at most *s* times, then the complexity of computing the upper envelope is in $O(n \log n)$

- Problem: change of flight path ⟶ recomputation of the envelope
  - Takes $O(n \log n)$
  - Can we update the envelope / topmost point faster?
- Solution: the tournament tree (kinetic heap)
  - Leaves = points
  - Inner node = topmost of its two children





- Certificate (for inner nodes) = left point is below right point
- Event = left/right point flip order along y axis
- Processing an event:
  - Replace the winner and replace $O(\log n)$ events in the event queue
  - Takes $O(\log^2 n)$ time ⟶ *responsive*
- Number of certificates (inner nodes) = $O(n)$ ⟶ *compact*
- Each point participates in $O(\log n)$ events ⟶ *local*

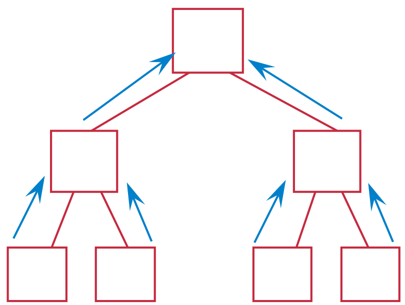


The Kinetic AABB

- Problem with deformable objects:
  BVH becomes invalid

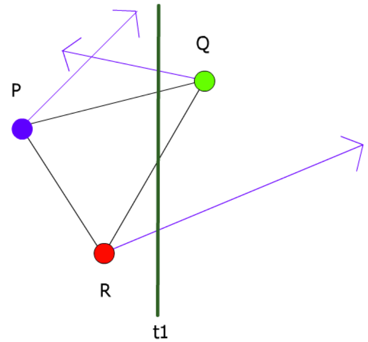
Classic BVH update:

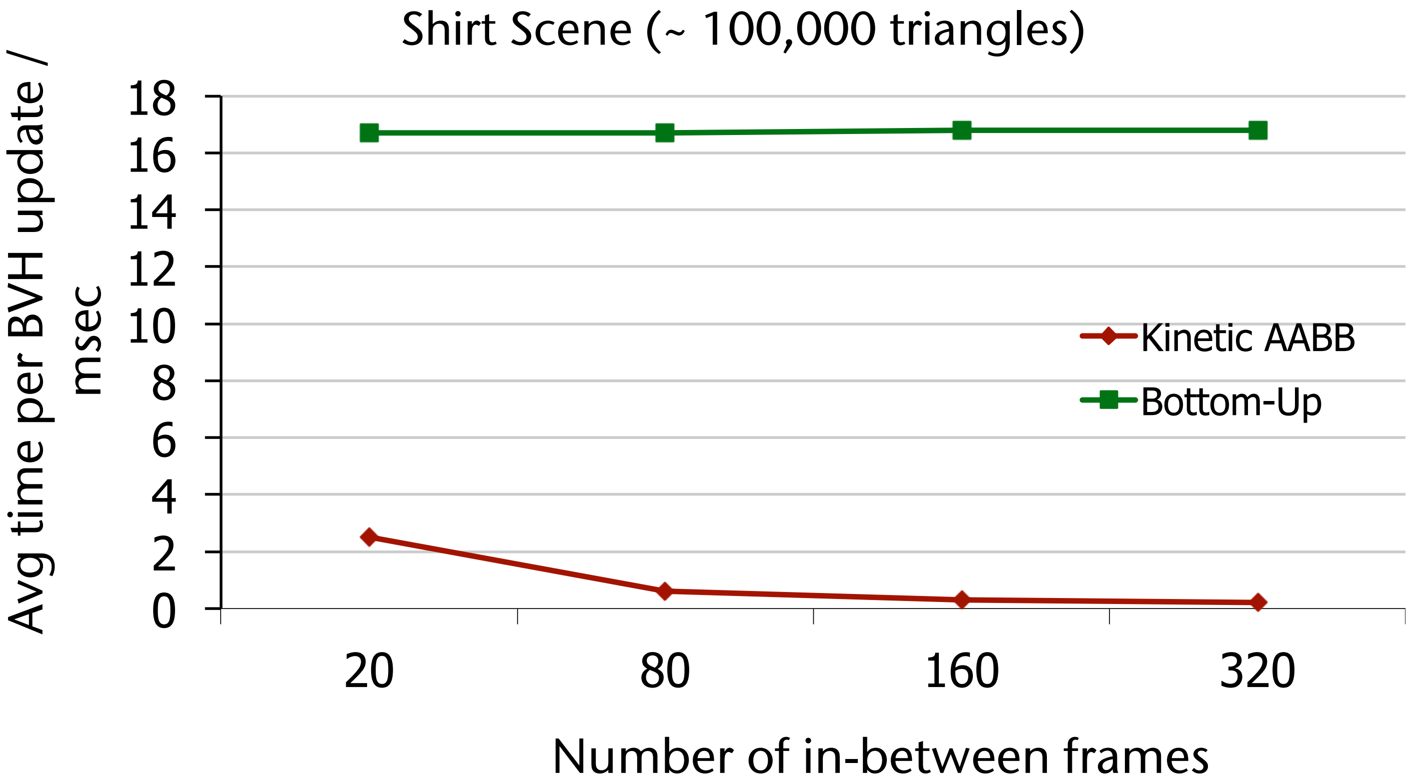
- Brute-force, bottom-up, i.e., for every query / anim. step
- $O(n \cdot \#\text{anim. steps})$ where $n = \#\text{pgons}$



Kinetic BVH update:

- Event-based (do work only, if something essential changed)
- $O(n \log n) \rightarrow$ *independent* of query/sim. frequency!



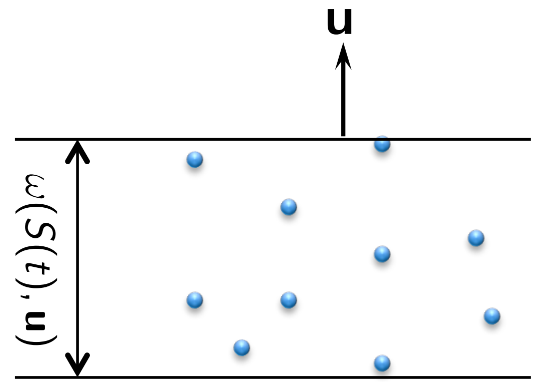

Shirt Scene (~ 100,000 triangles)
Avg time per BVH update / msec
18
16
14
12
10
8
6
4
2
0
20
80
160
320
Number of in-between frames
Kinetic AABB
Bottom-Up

Problems of KDS

- Too many events for many KDS
- Computing event times is expensive
- Querying moving objects
  - No need to maintain the structure at all times

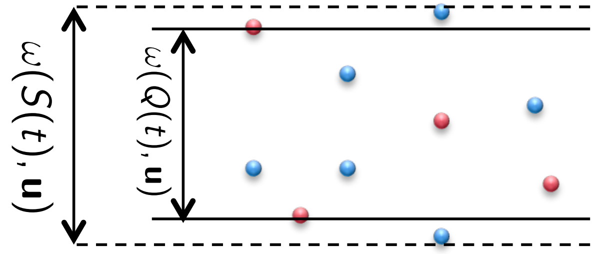
One Possible Approach by Way of an Example

- Definition: directional width
  Let S = set of moving points.
  Define the width in direction **u**
  at time *t* as $\omega(S(t), \mathbf{u})$.



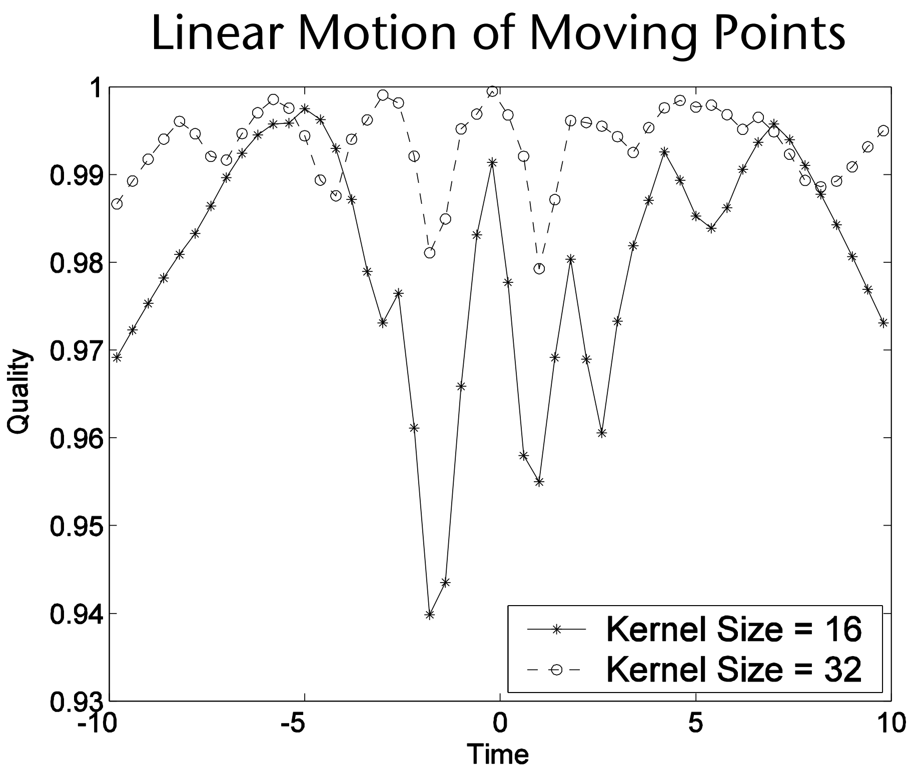
- Definition: ε -kernel
  Let $Q \subseteq S$.
  Q is called an ε -kernel of S iff

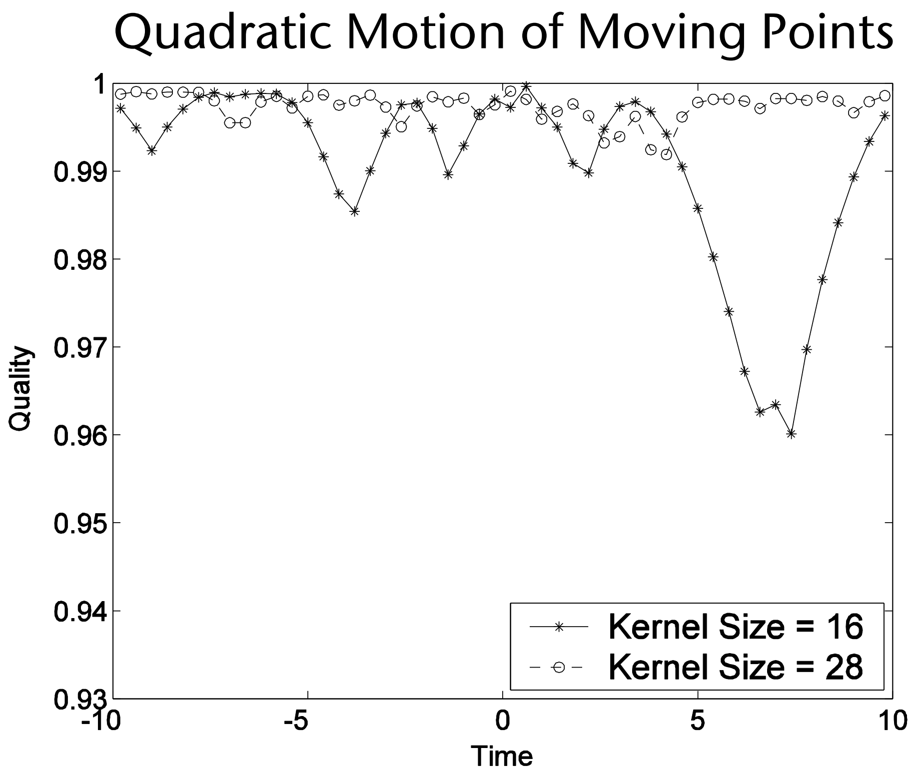
$$\forall t : \omega(S(t), \mathbf{u}) \leq (1 + \varepsilon)\omega(Q(t), \mathbf{u})$$



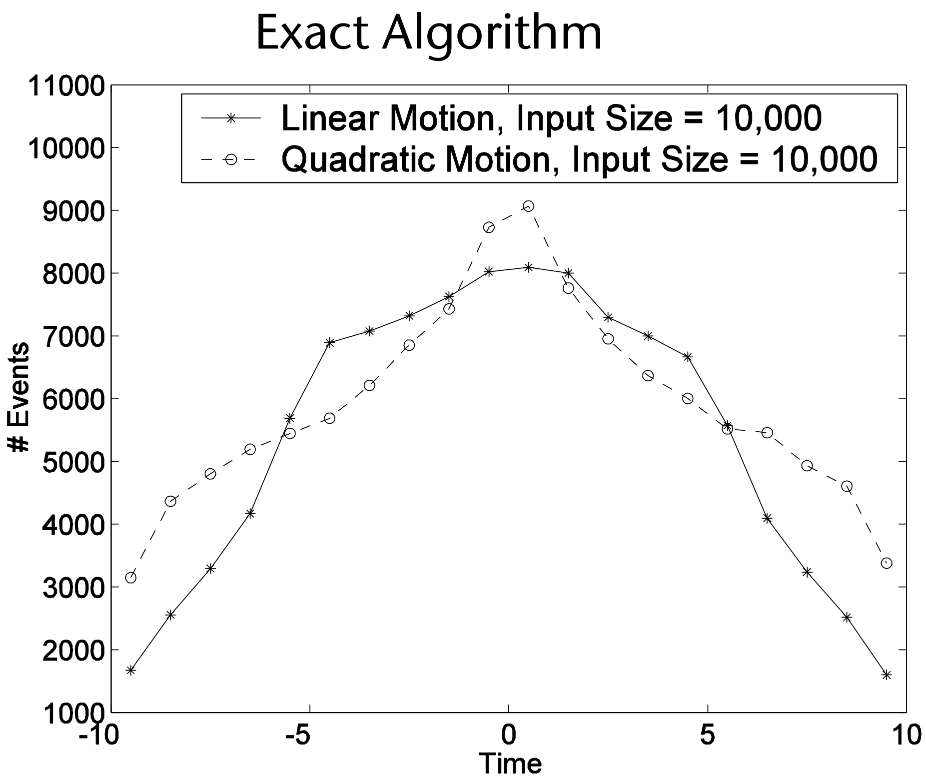
- Theorem [Agarwal, Har-Peled, Varadarajan]:
  For *n* points moving with fixed velocity in 2D, and any ε > 0, one can compute an ε -kernel of size $O\left(\frac{1}{\varepsilon^{\frac{3}{2}}}\right)$ in time $O\left(n + \frac{1}{\varepsilon^3}\right)$.

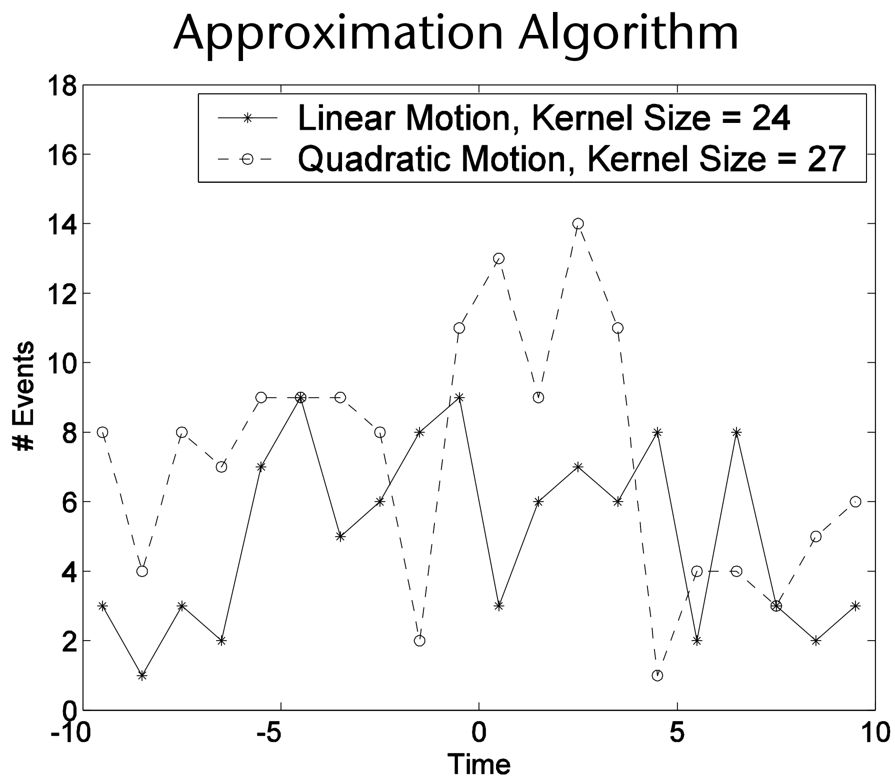
Results for BBox Maintained by E-Approximate KDS



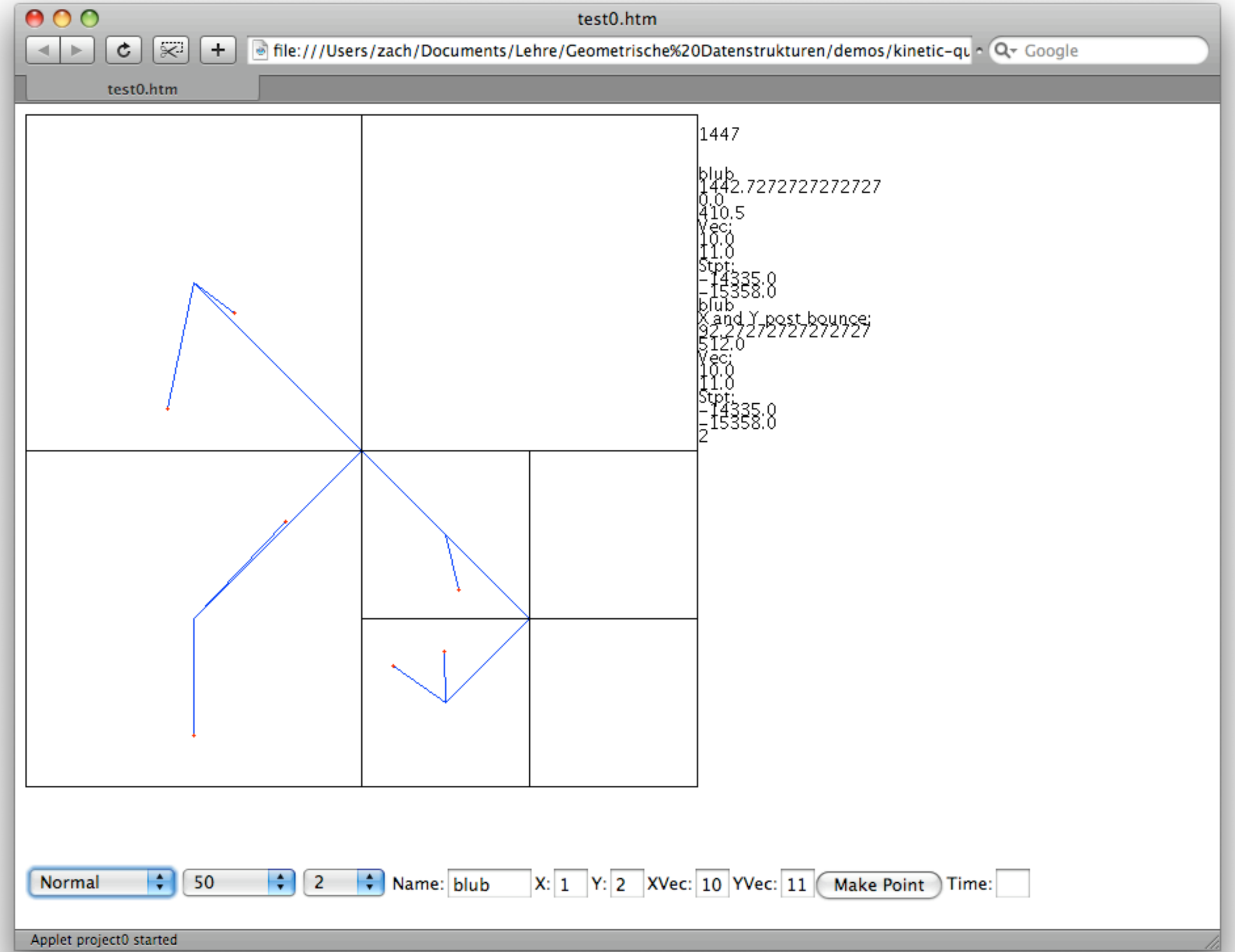


10,000 moving points
Error < 0.02 for kernel of size 32

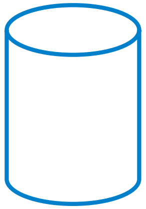

Exact Algorithm
Linear Motion, Input Size = 10,000
Quadratic Motion, Input Size = 10,000
Events
Time


Approximation Algorithm
Linear Motion, Kernel Size = 24
Quadratic Motion, Kernel Size = 27
Events
Time

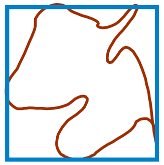
Kinetic Quadtree Demo



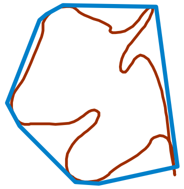
Examples of Bounding Volumes



Cylinder
[Weghorst et al., 1985]

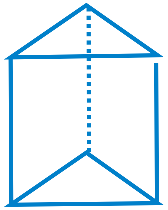


Box, AABB (R*-trees)
[Beckmann, Kriegel, et al., 1990]

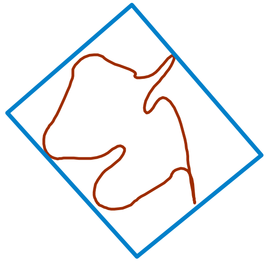


Convex hull
[Lin et. al., 2001]

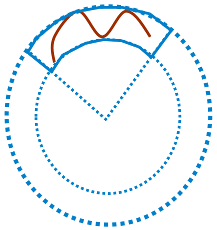
Sphere
[Hubbard, 1996]



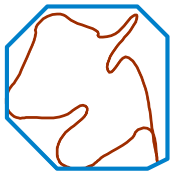
Prism
[Barequet, et al., 1996]



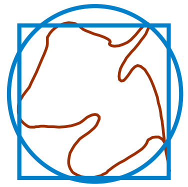
OBB (oriented bounding box)
[Gottschalk, et al., 1996]



Spherical shell
[...]

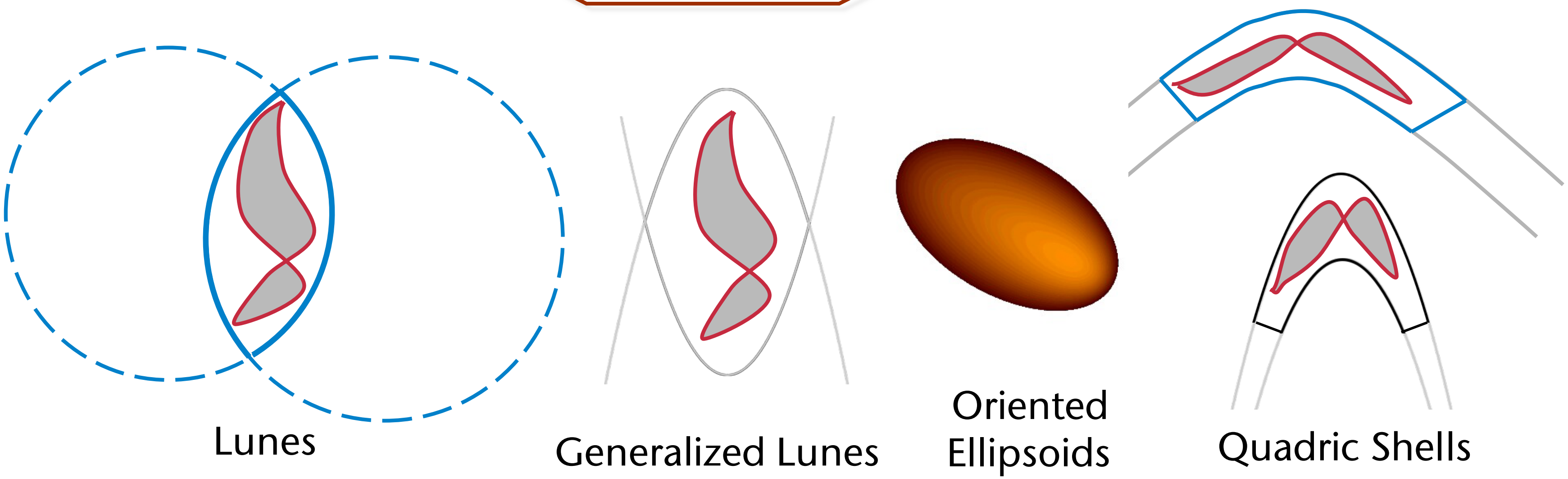


k-DOPs / Slabs
[Zachmann, 1998]



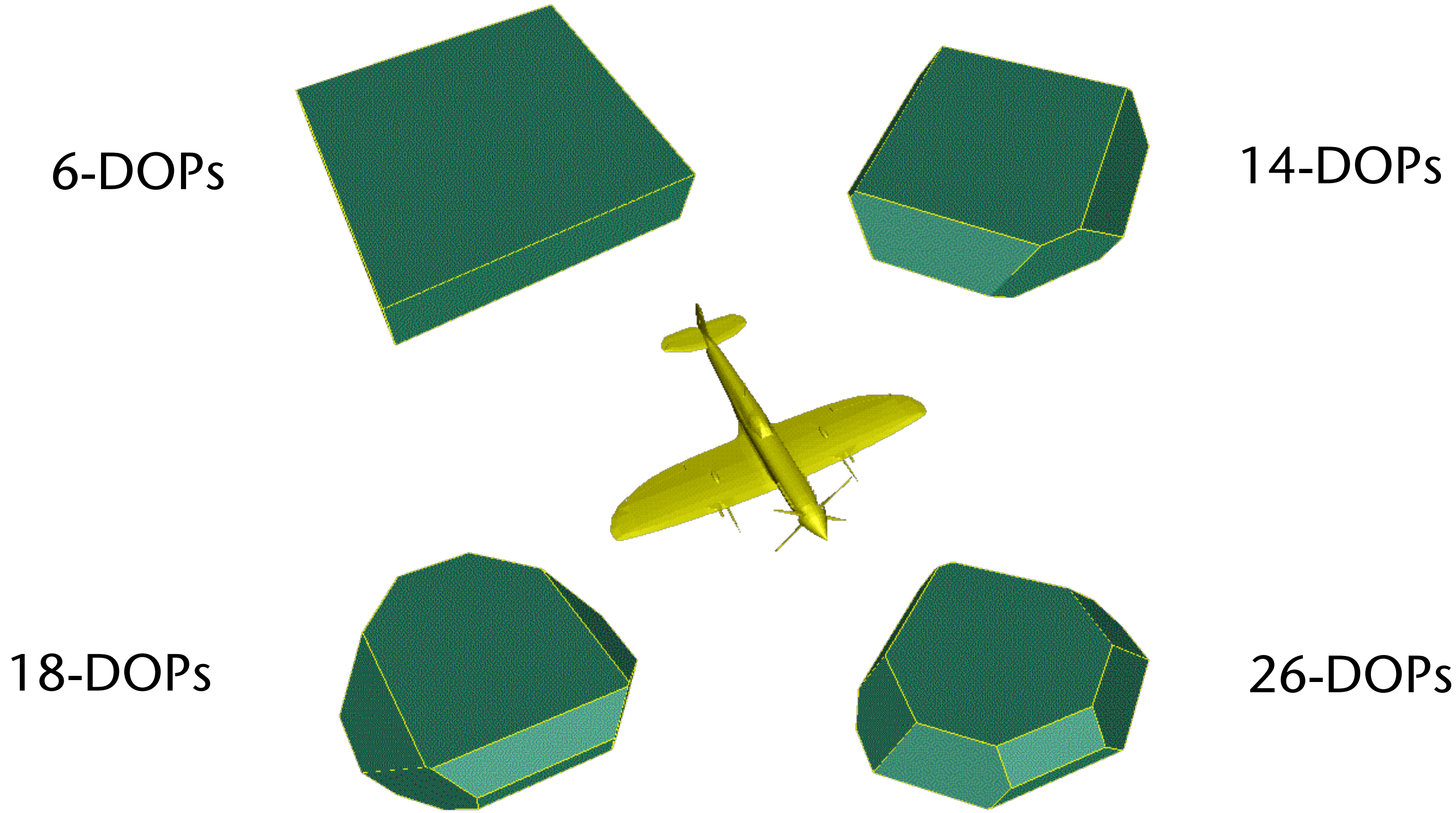
Intersection of
several, other BVs

- Some ideas for several Master's theses ...

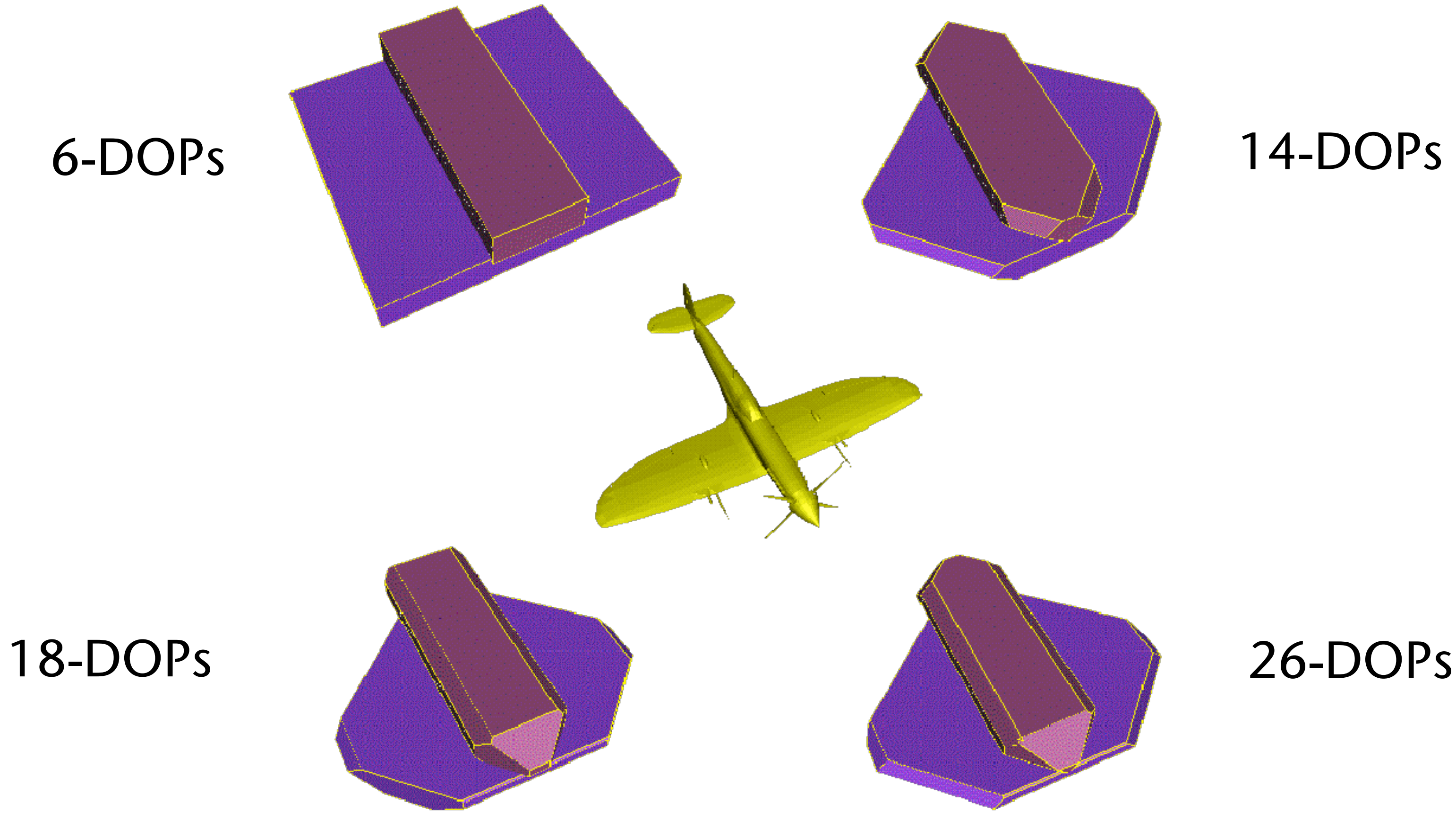


- Research questions:
  - Fast intersection of two BVs for collision detection?
    - Compute is cheap, memory transfer is expensive ⟶ BV compression?
    - Exact / approximate (biased) intersection tests?
  - Fast intersection test for rays against such BVs?
  - Efficient BVH construction? (for fast queries at runtime)

BVH with *k*-DOPs



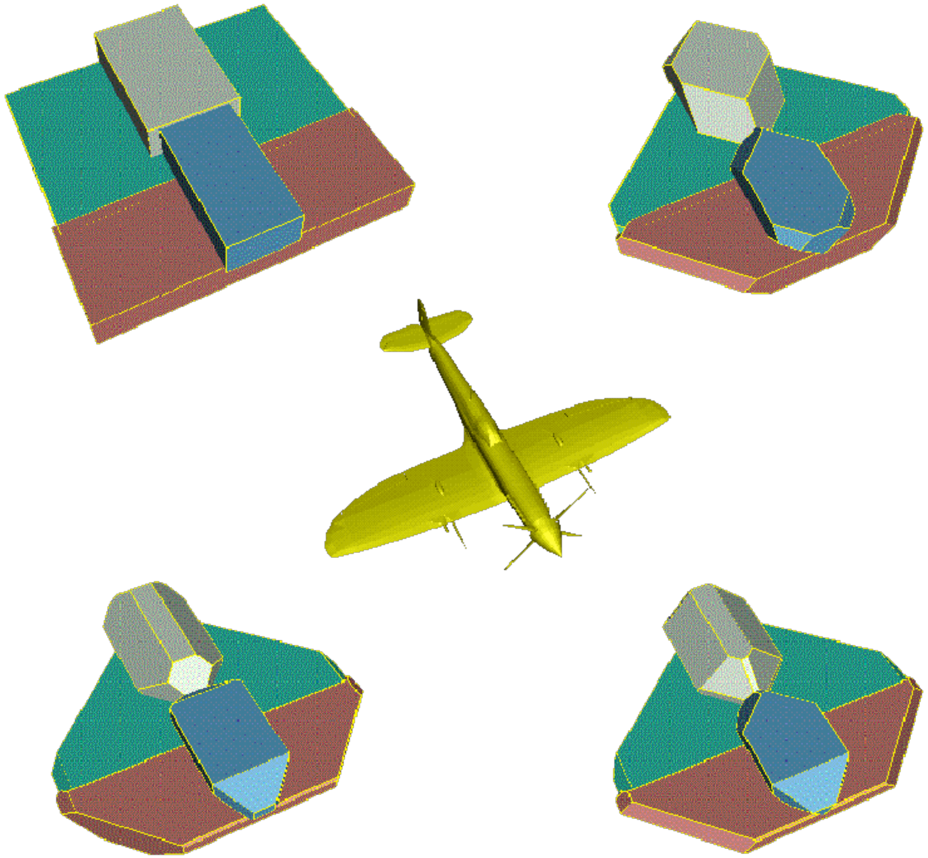
Level 0



Level 1

6-DOPs

14-DOPs



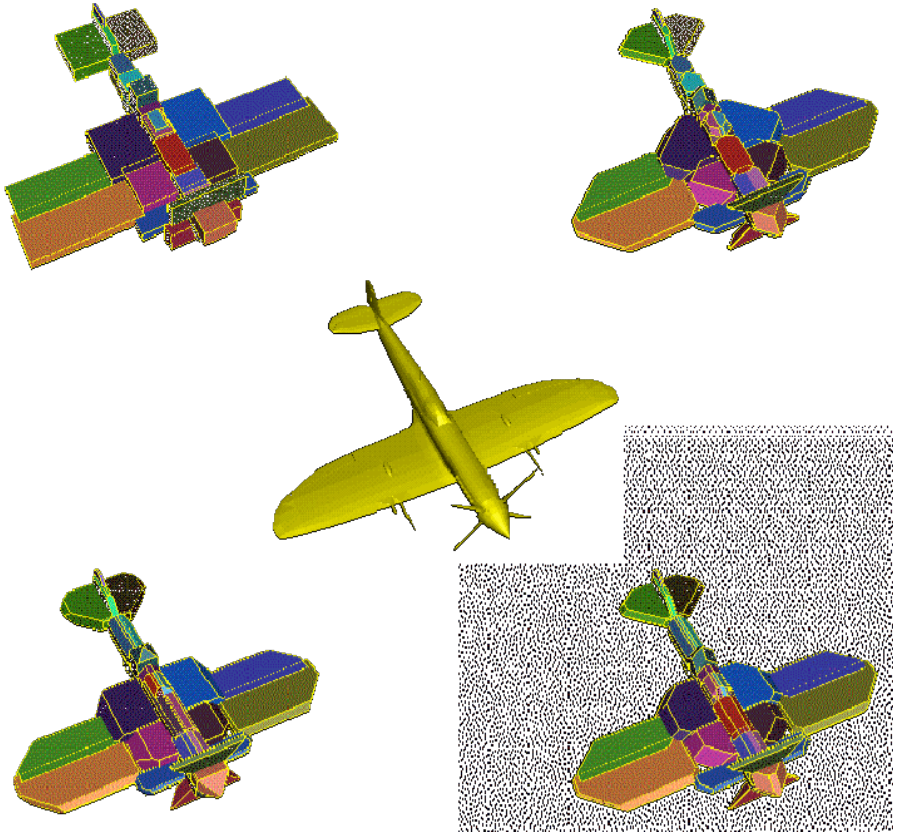
18-DOPs

26-DOPs

Level 2

6-DOPs

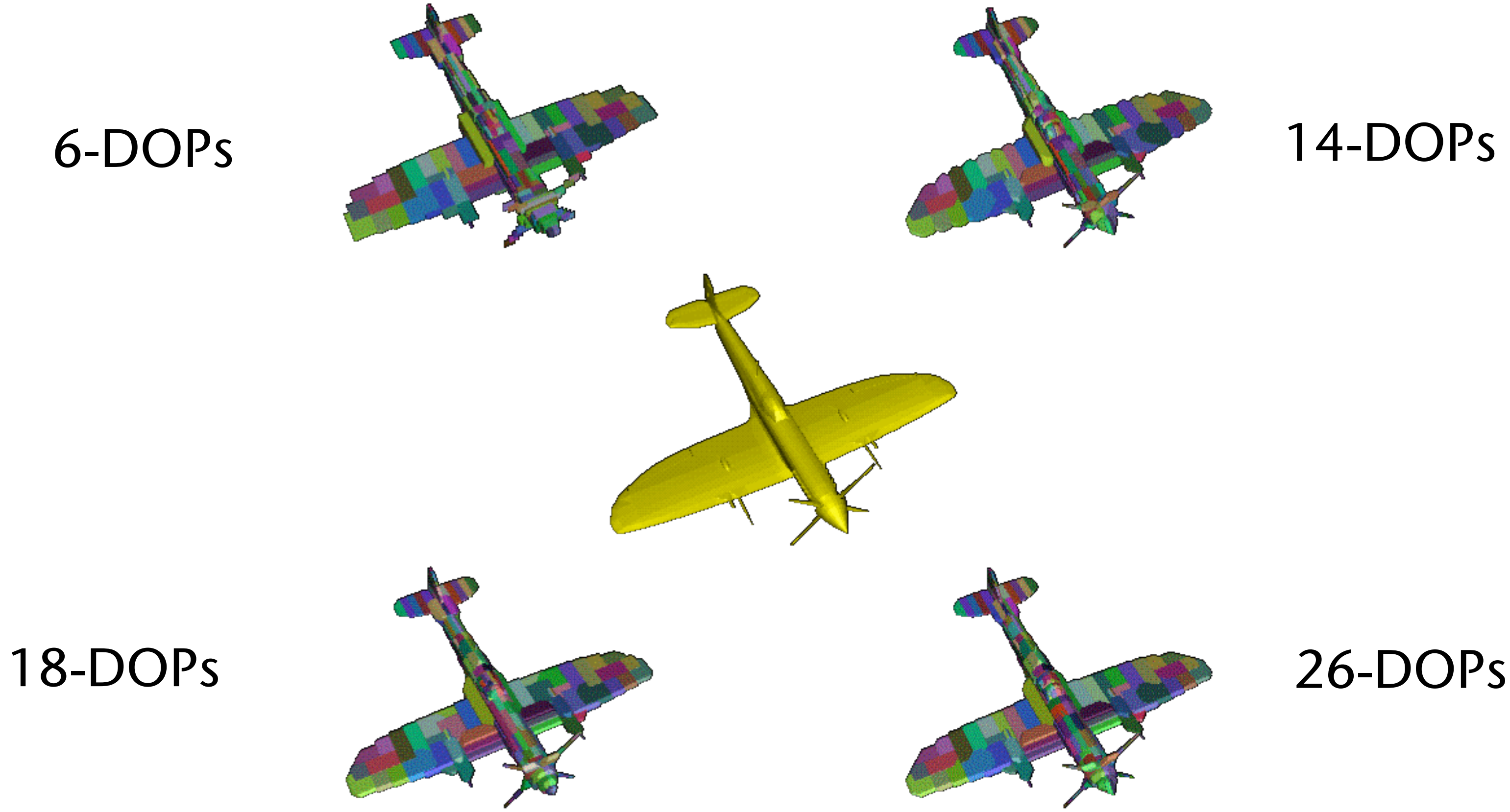
14-DOPs



18-DOPs

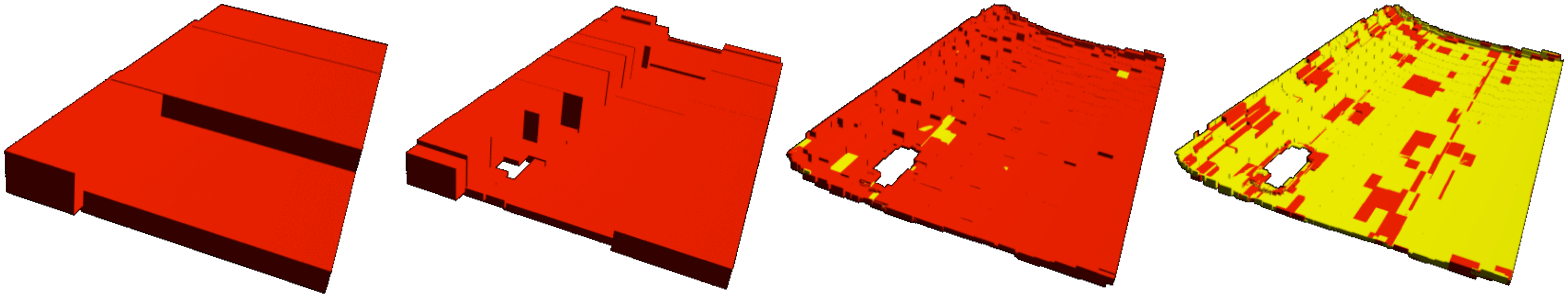
26-DOPs

Level 5



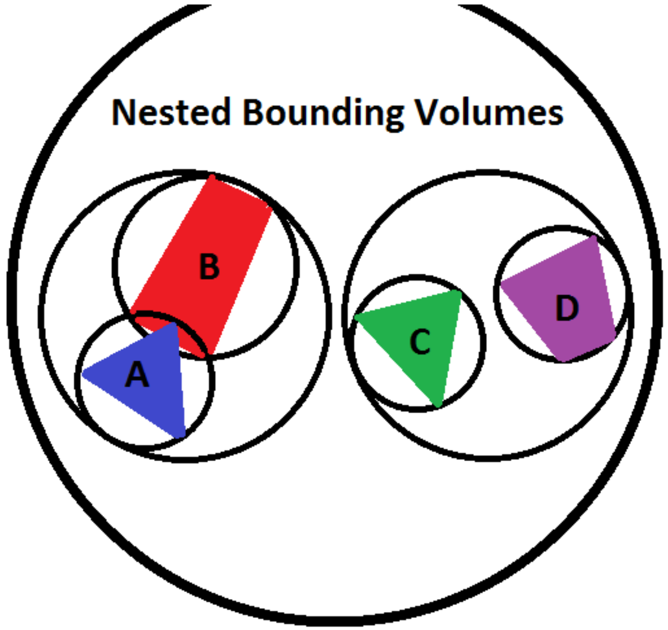
Level 8

BVH with AABBs



Wrapped vs Layered BVH

Wrapped BVH:
a BV bounds its associated primitives,
but not necessarily its child BVs



Layered BVH:
a BV must bound its child BVs

Hierarchical Collision Detection using BVHs

<u>traverse(X, Y)</u>

if X,Y do not overlap **then**

  return

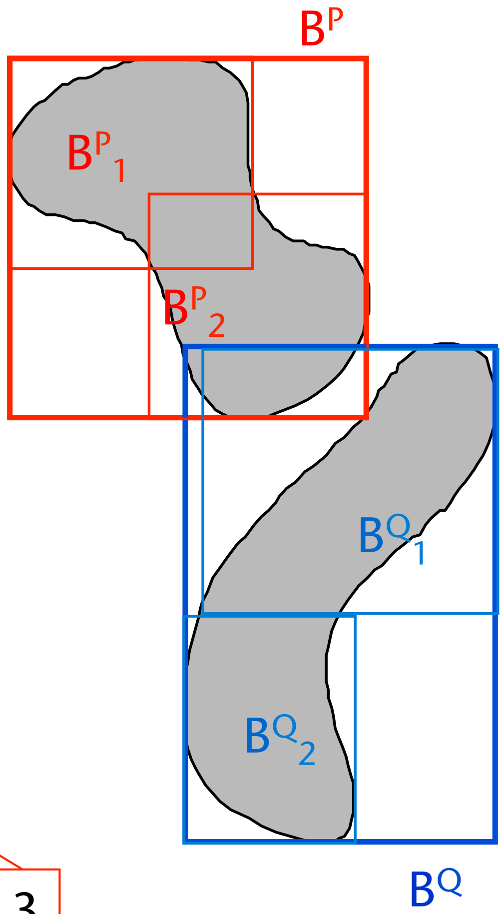
if X,Y are leaves **then**

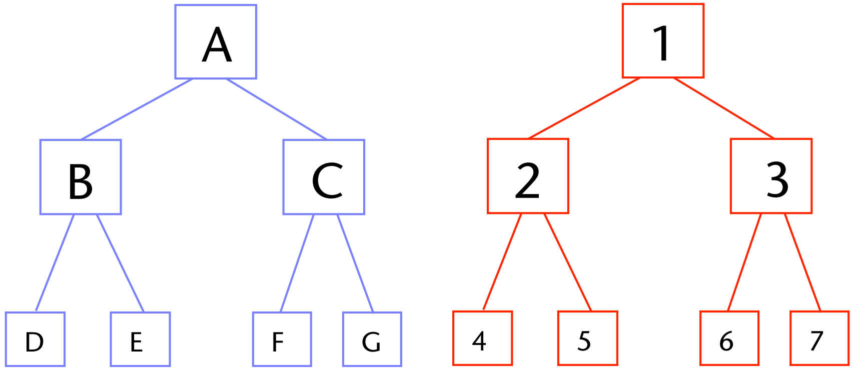
  check polygons

else

  for all children pairs **do**

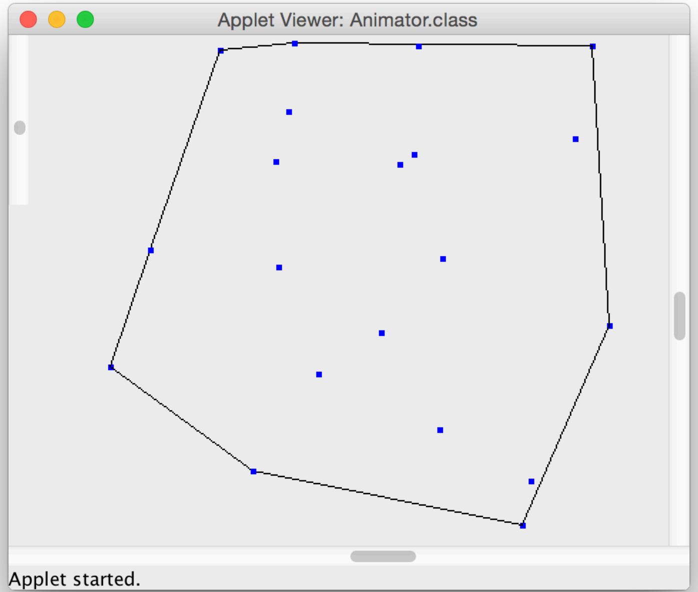
  traverse(X_i, Y_j)



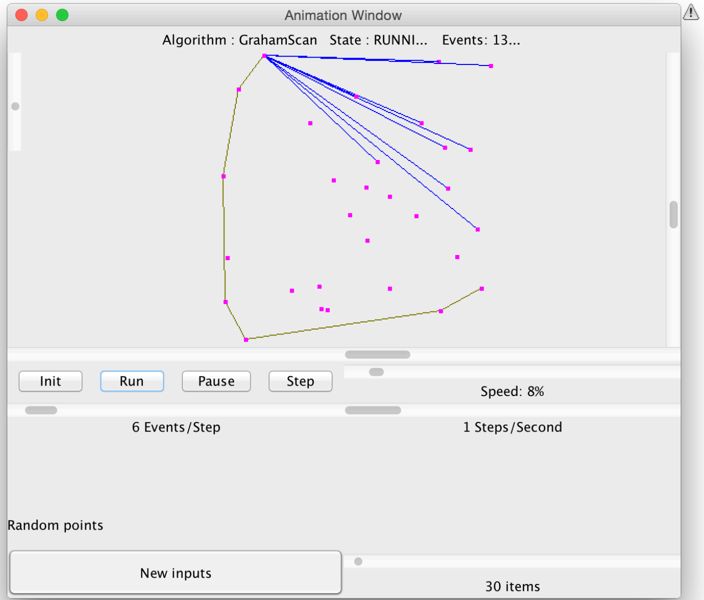


Applications using Distance Fields

Convex Hull Demos in 2D

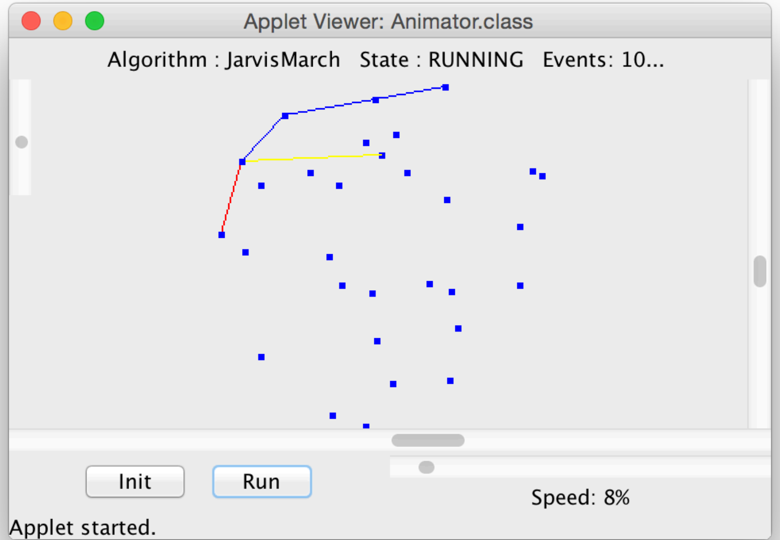


Rubber band metaphor

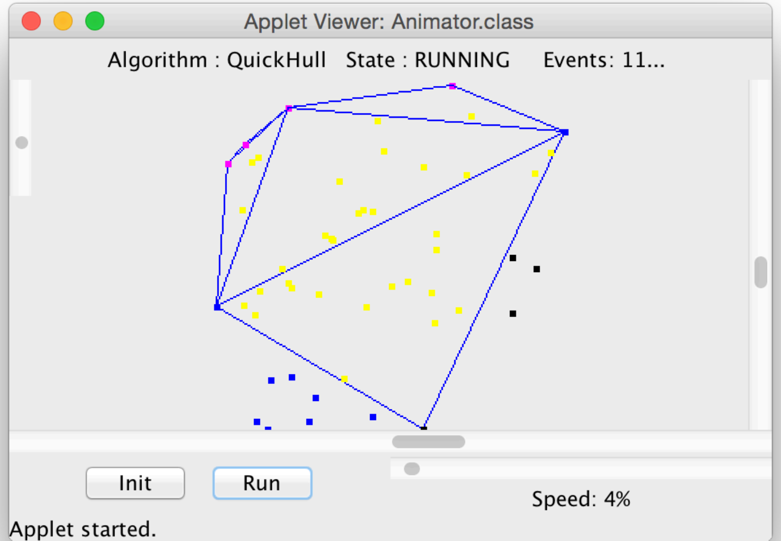


Graham's scan (with slightly different sorting order)

Alejo Hausner - http://www.cs.princeton.edu/~ah/alg_anim/version1/GrahamScan.html



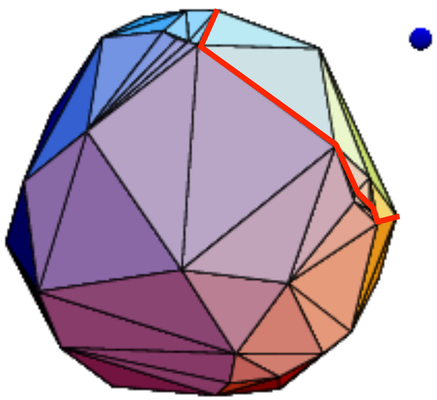
Jarvis' March

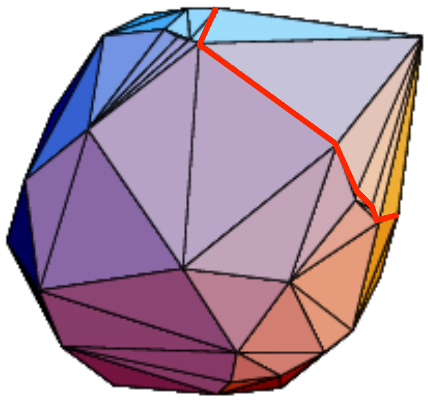


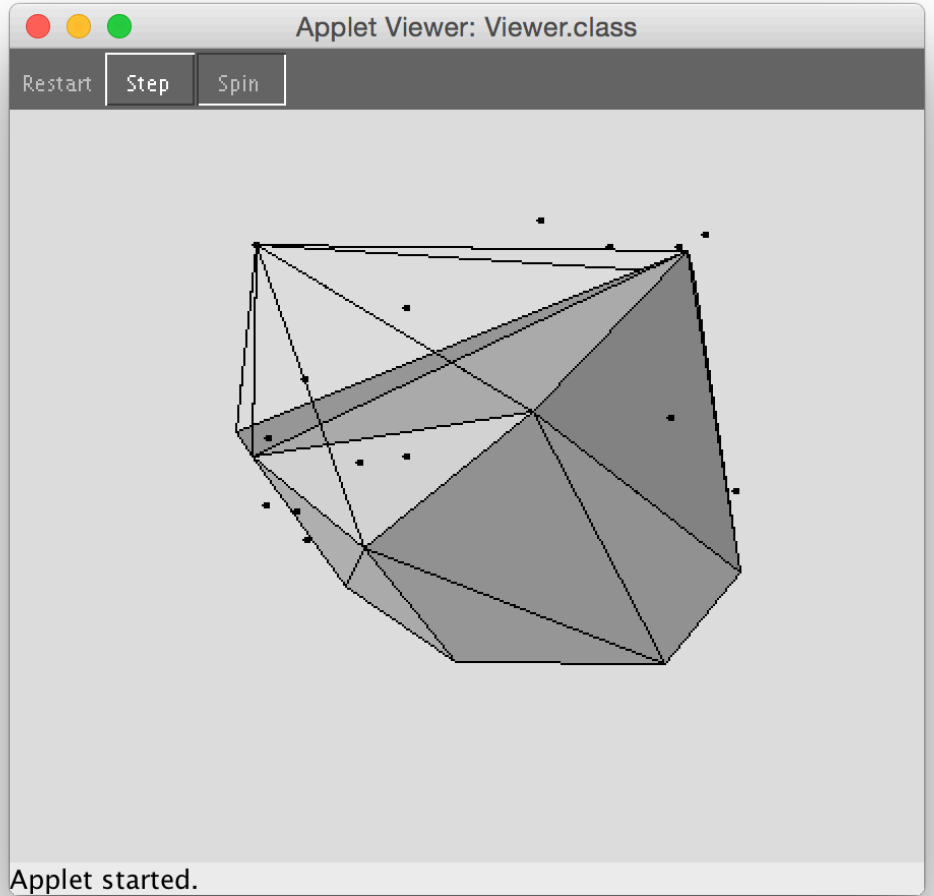
QuickHull

Convex Hull in 3D

- Ein Schritt des inkrementellen Algorithmus':

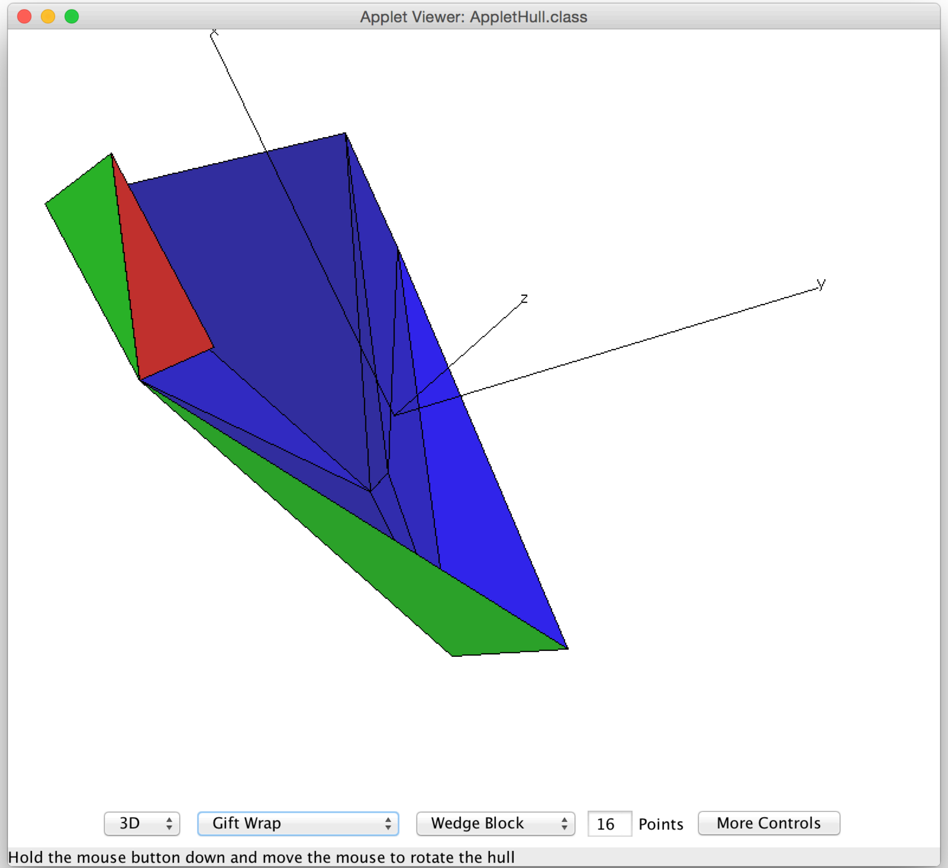






Clarkson-Shor-Algorithm (randomized incremental)

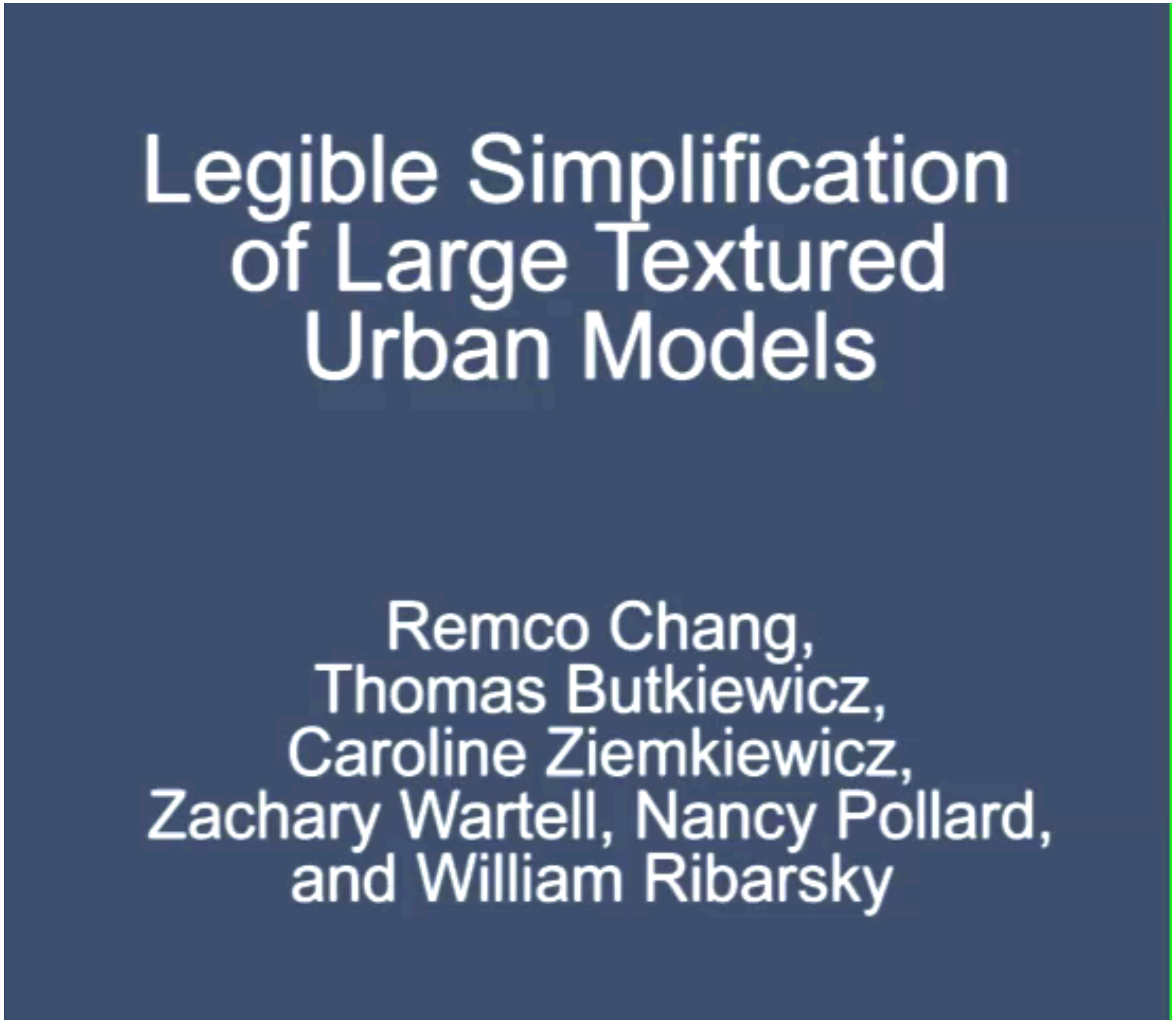
Michael Horn - http://www.eecs.tufts.edu/~mhorn01/comp163/



Different algorithms, e.g., *gift wrapping*

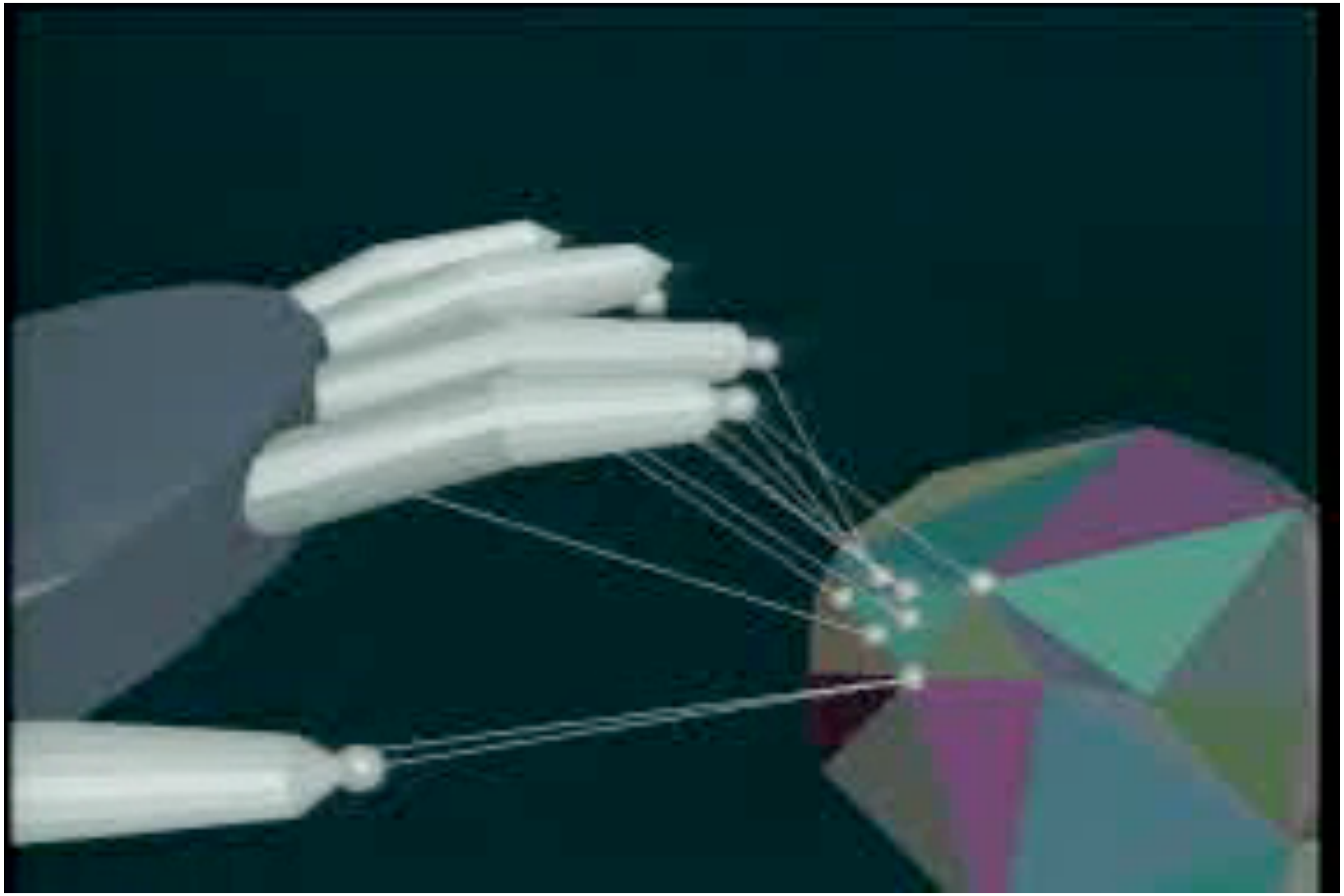
Tim Lambert - http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html

Simplification of Urban Models



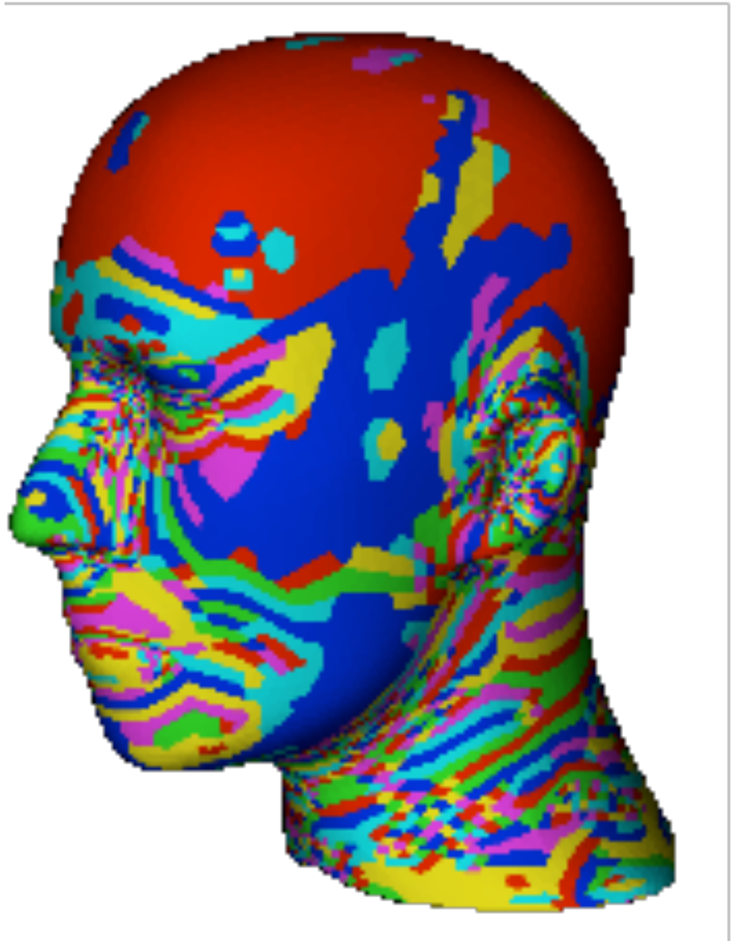
Remco Chang, Thomas Butkiewicz, Caroline Ziemkiewicz, Zachary Wartell, Nancy Pollard, William Ribarsky

Convex Collision Detection



Achtung: der hier demonstrierte Algo ist in Wahrheit
etwas komplexer als der in der Vorlesung dargestellte!
(aber möglicherweise nicht schneller …)

Convex Surface Decomposition

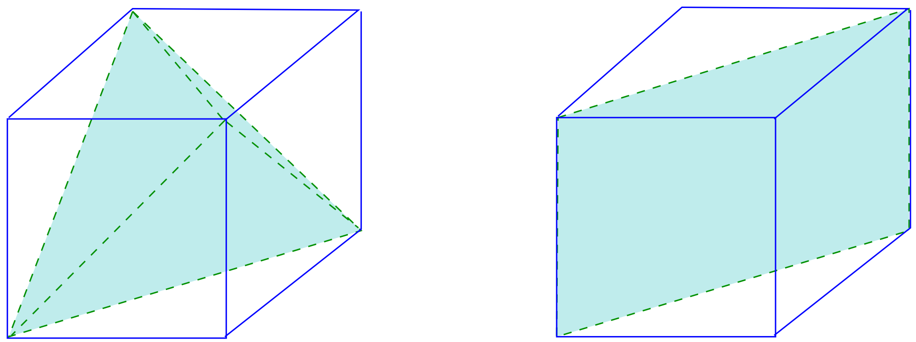


Zerlegung in
konvexe Surface-Patches

Konvexe Stücke auf einem
mittleren Level der Hierarchie
(grün = orig. Fläche, rot = freie Fläche,
gelb = "contained")

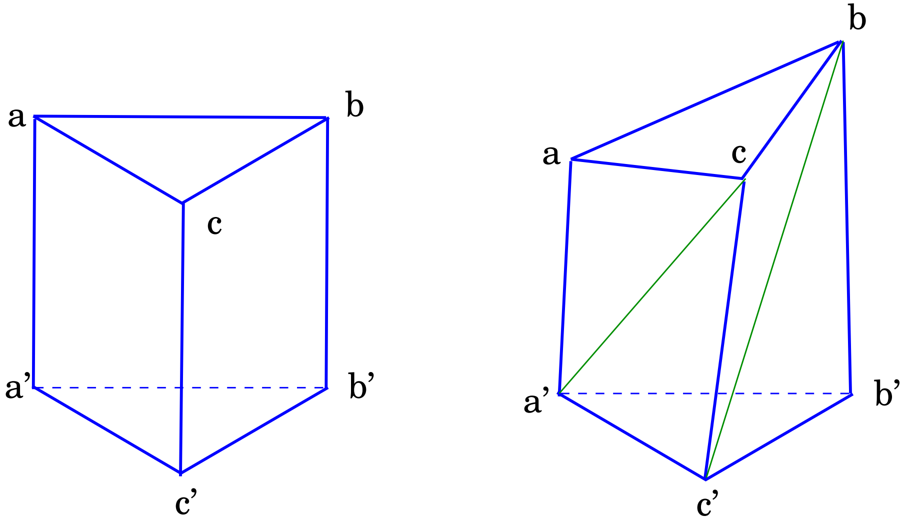
Zum Vergleich: Triangulation in 3D (="Tetraedrisierung")

- Verschiedene Triangulierung → verschiedene Anzahl Tetraeder:

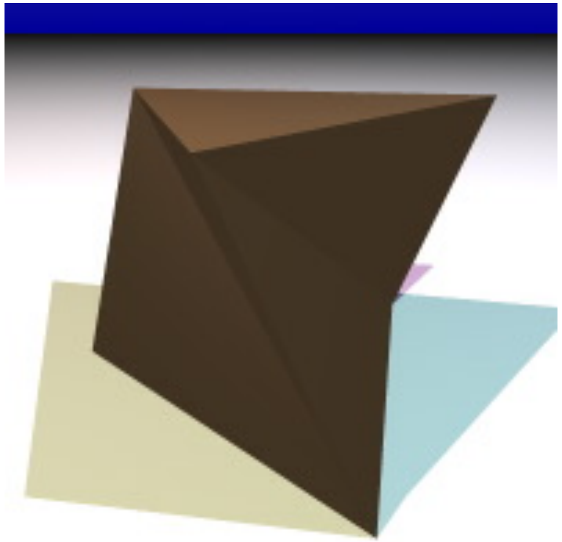


5 Tetraeder   6 Tetraeder

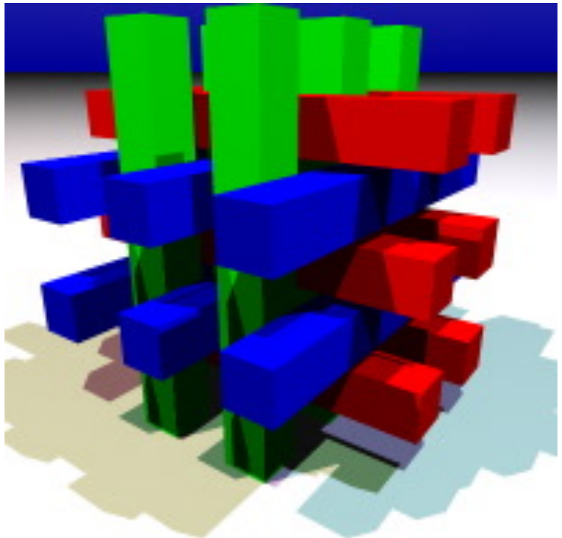
- Ein untriangulierbares ("un-tetraedrisierbares") Polyeder:



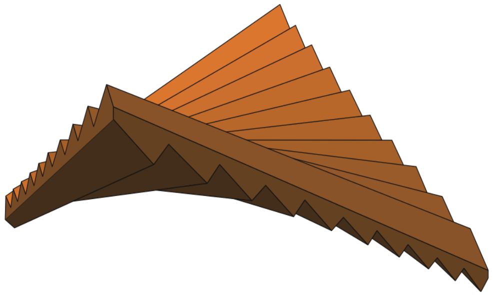
Untetrahedralizable Objects



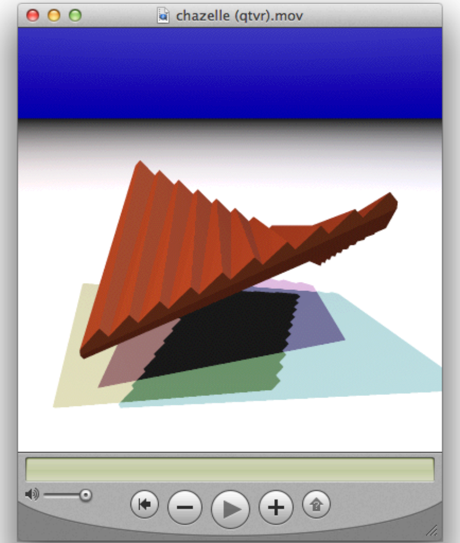
Schönhardt's Polyeder (1928)



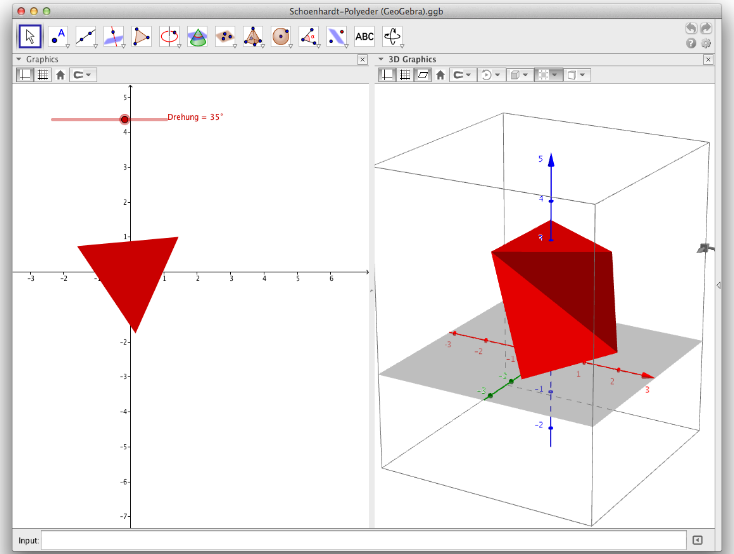
Thurston-Polyeder (1971)



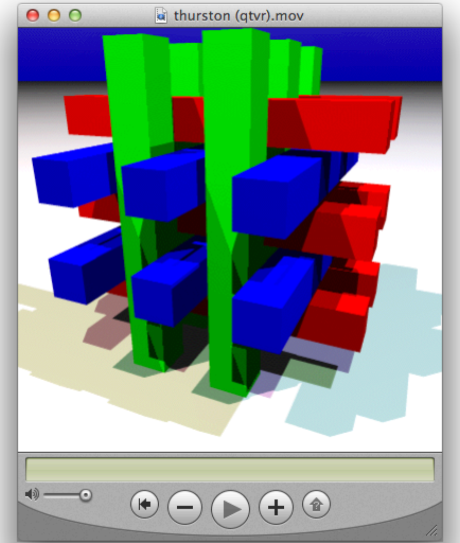
Chazelle's Polyeder (1984)



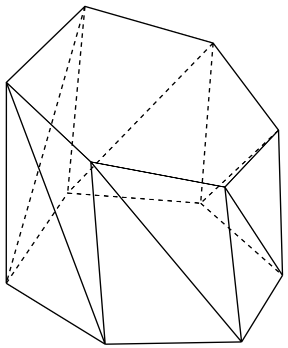
Chazelle Polyhedron



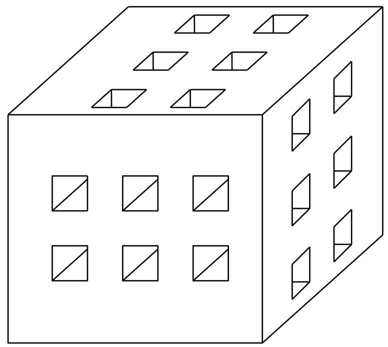
Schönhardt Polyhedron

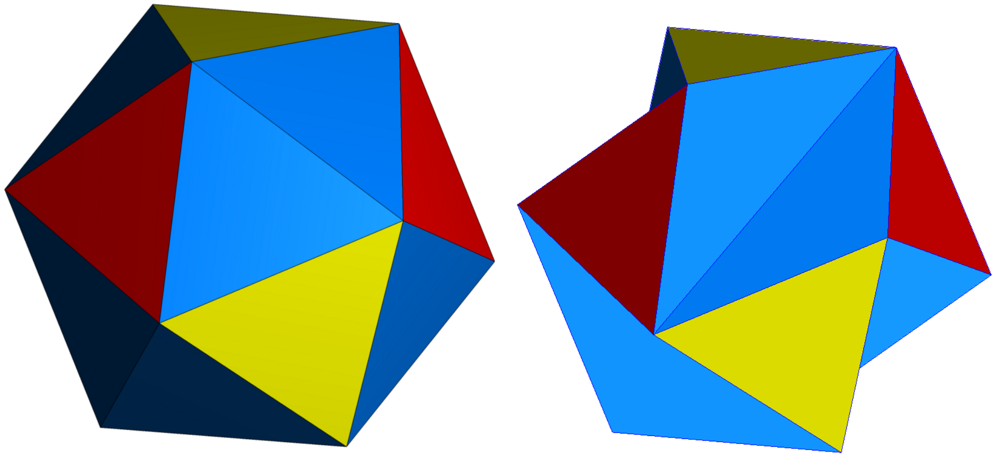


Thurston Polyhedron

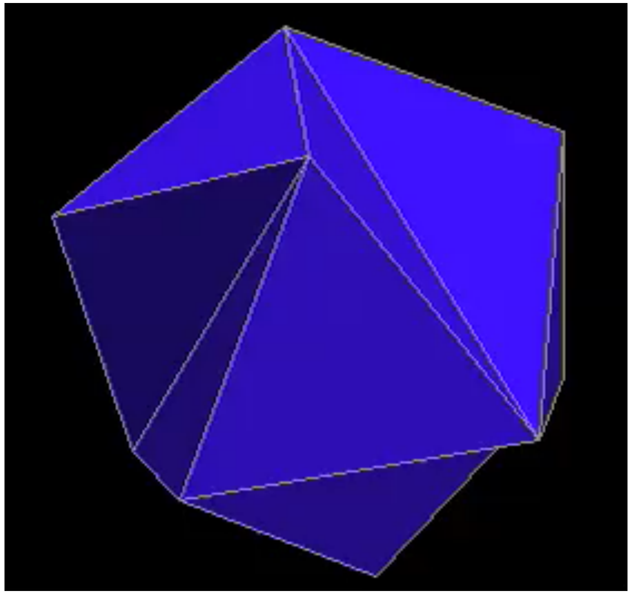


Generalization of
Schönhardt by Rambau



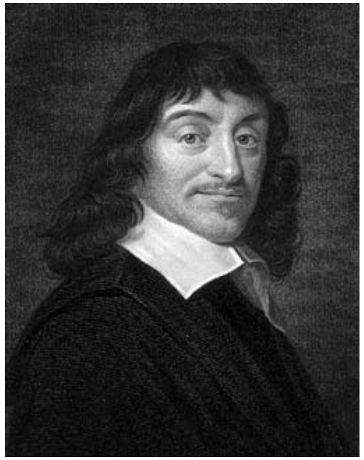


Jessen's Ikosaeder

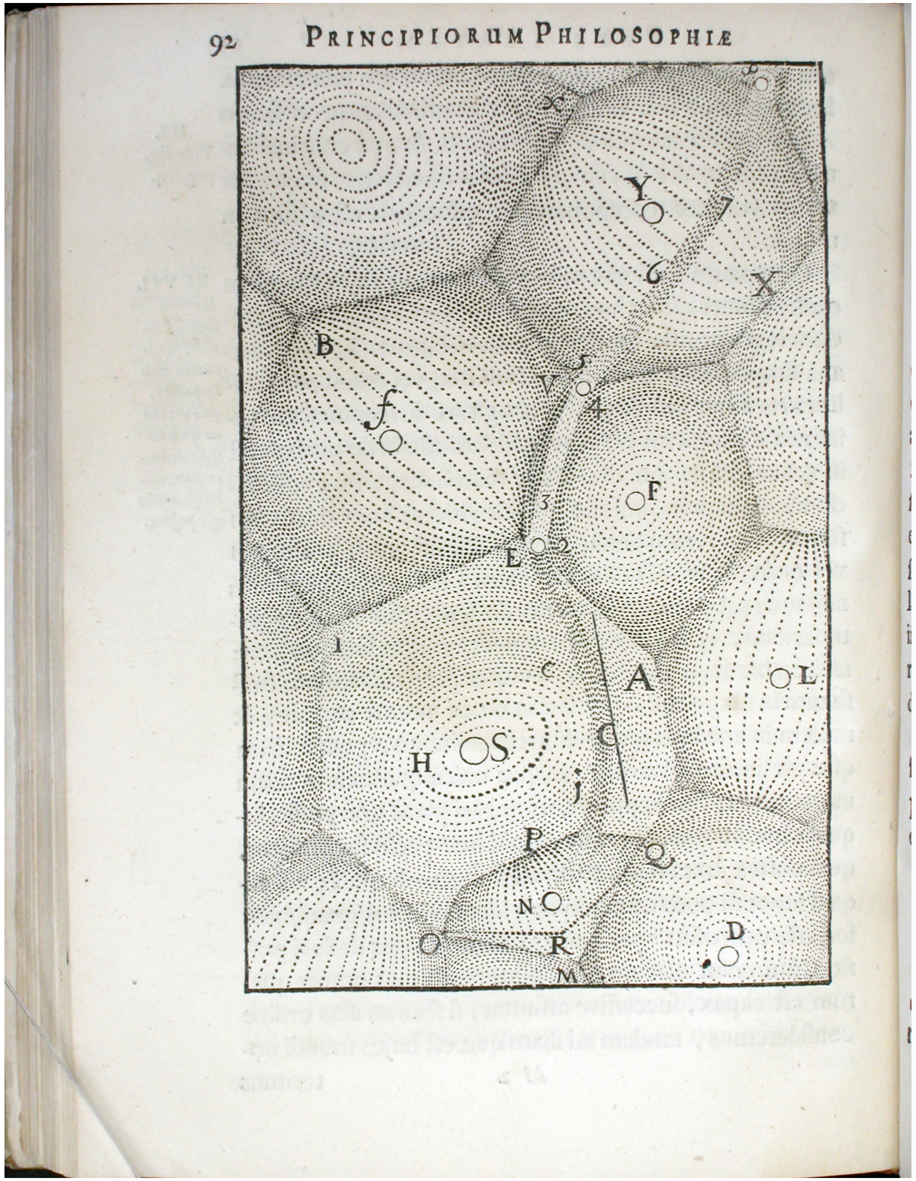


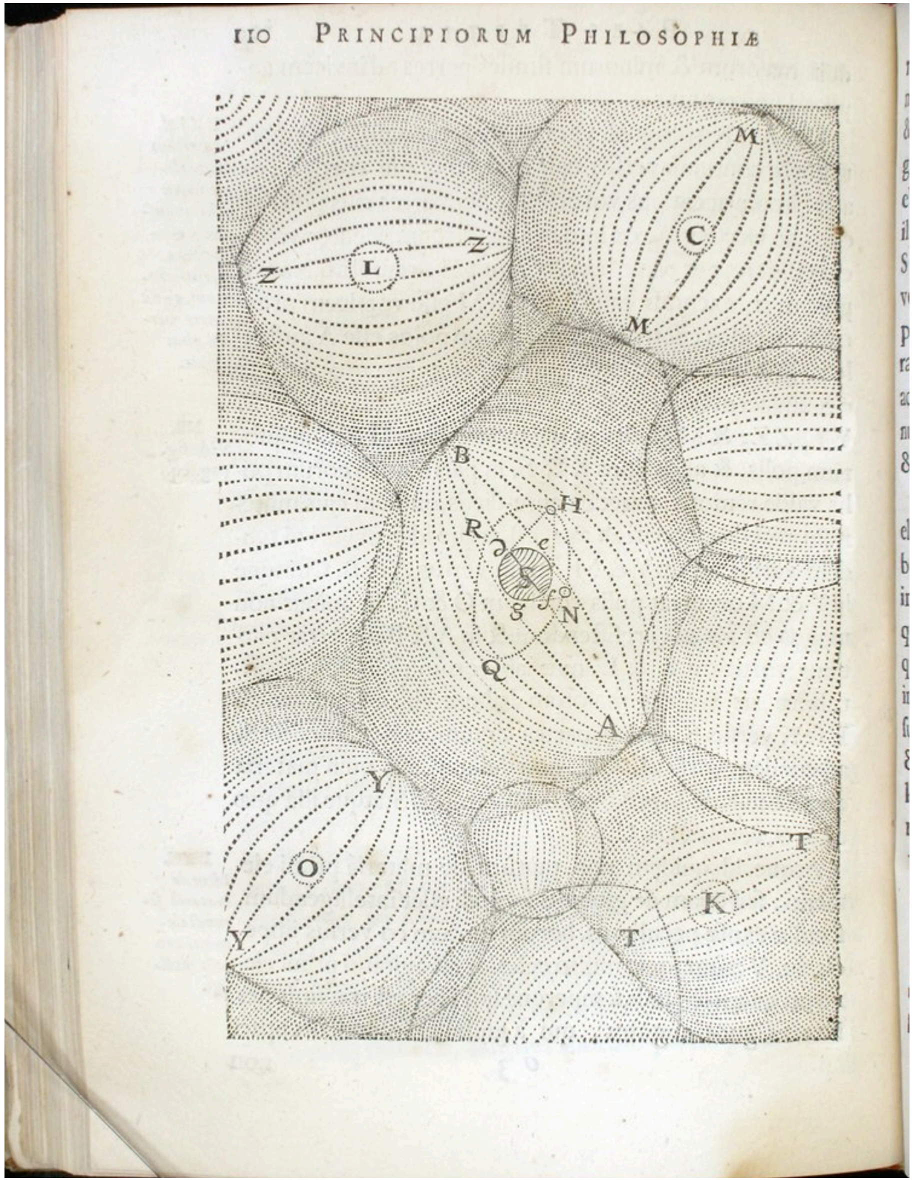
Voronoi-Diagramme

- Eine der ersten Erwähnungen von René Descartes (Cartesius; 1596-1650) in seiner *Principia Philosophiae*, 1644:
  - Stellte sich vor, daß das Universium mit Materie gefüllt ist, die von den Sternen angezogen wird und um diese herumwirbelt



- Georgy F. Voronoy (Георгий Ф. Вороной) 1868 – 1908
  - Geboren in Russland, heutige Ukraine
  - Professor in Warschau
  - Schüler: Delaunay


92 PRINCIPIORUM PHILOSOPHIÆ


110 PRINCIPIORUM PHILOSOPHIÆ

Independent Discoveries in Other Fields

Descartes	Astronomy	1644	“Heavens”
Dirichlet	Math	1850	Dirichlet tesselation
Voronoi	Math	1908	Voronoi diagram
Boldyrev	Geology	1909	area of influence polygons
Thiessen	Meteorology	1911	Thiessen polygons
Niggli	Crystallography	1927	domains of action
Wigner & Seitz	Physics	1933	Wigner-Seitz regions
Frank & Casper	Physics	1958	atom domains
Brown	Ecology	1965	areas potentially available
Mead	Ecology	1966	plant polygons
Hoofd et al.	Anatomy	1985	capillary domains

Delaunay (1890 – 1980)

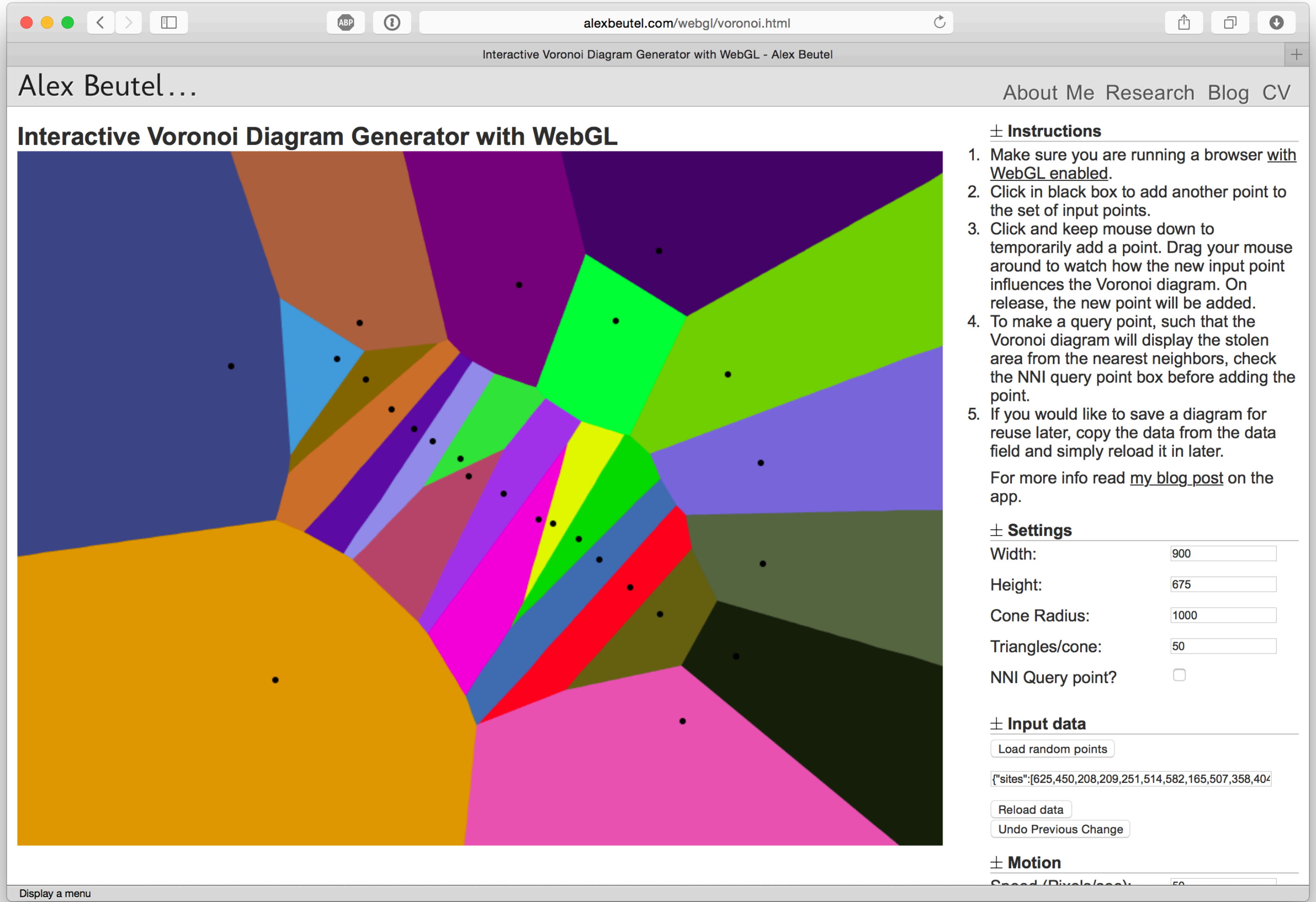
- Schüler von Voronoy (und Grave)
- Einer der 3 besten russischen Bergsteiger um 1930
- Russische Schreibweise: Борис Николаевич Делоне
    - Damals war Französisch (und Deutsch) die Wissenschaftssprache!

- Nicht zu verwechseln mit dem Maler Robert Delaunay !
  - 1885 – 1941 ; wirklich französisch

Champs de Mars. La Tour rouge. 1911

Homage à Bleriot, 1914

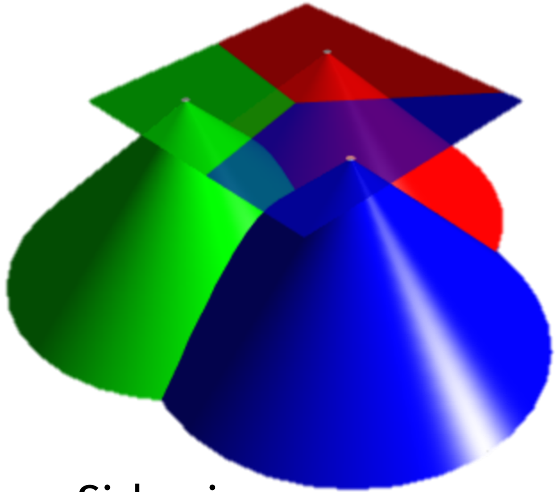
Demos



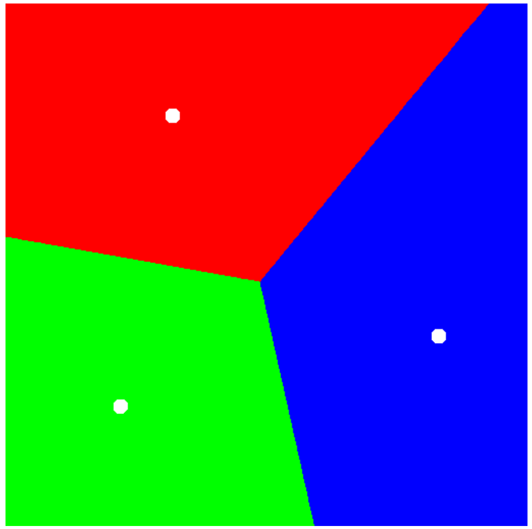
http://alexbeutel.com/webgl/voronoi.html

The "Cones Trick" to Generate Approximate 2D Voronoi Diagrams

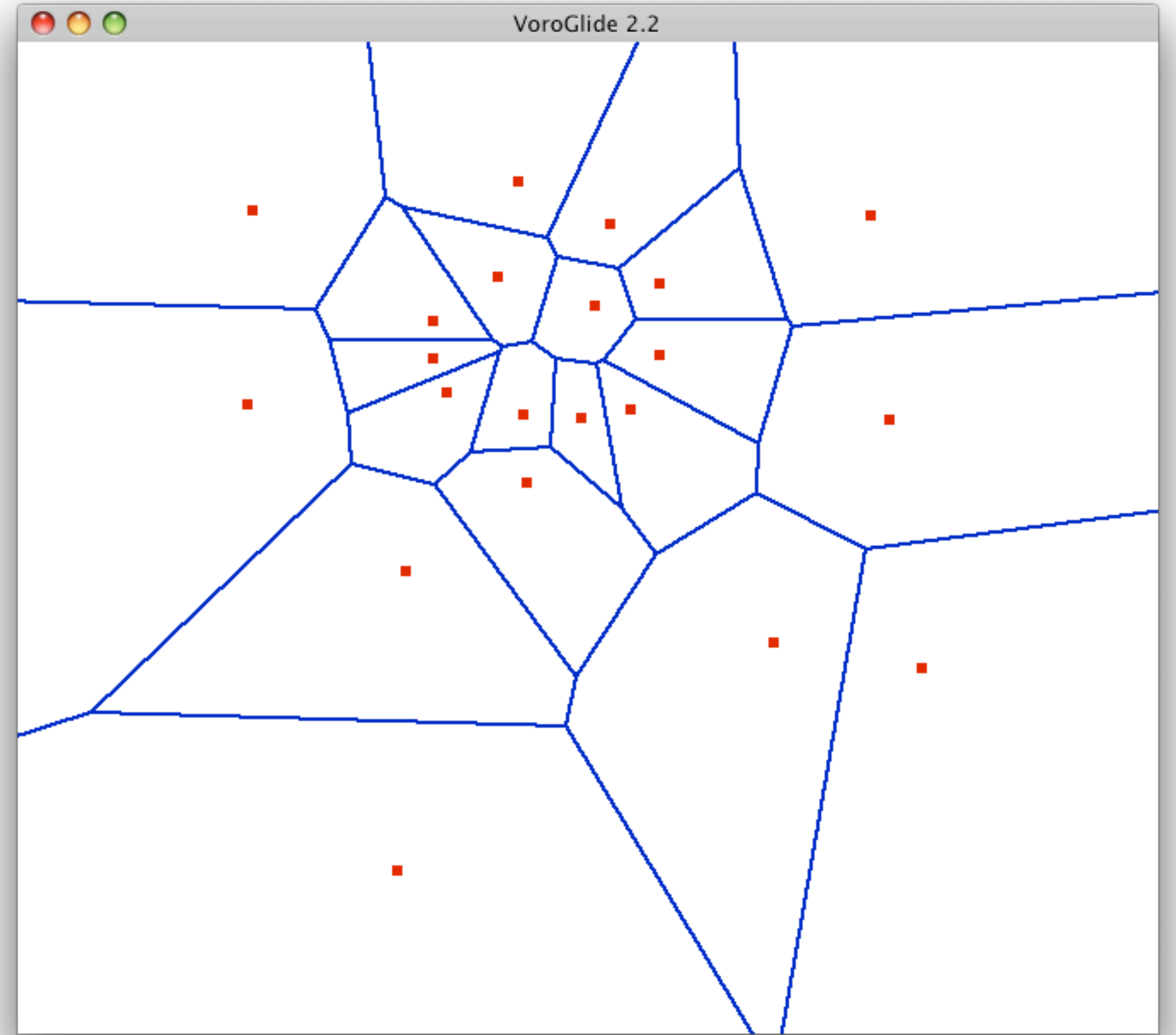
- Observation:
  - Place a cone at every Voronoi site with 90° angle
  - Distance of a point X from Voronoi site = height of cone above X
- Method:
  - For each site, render a cone with different color (= ID)
  - Borders in color buffer = Voronoi edges
  - Value in Z-buffer = distance from site
- Already noticed by Dirichlet & Voronoi



Side view

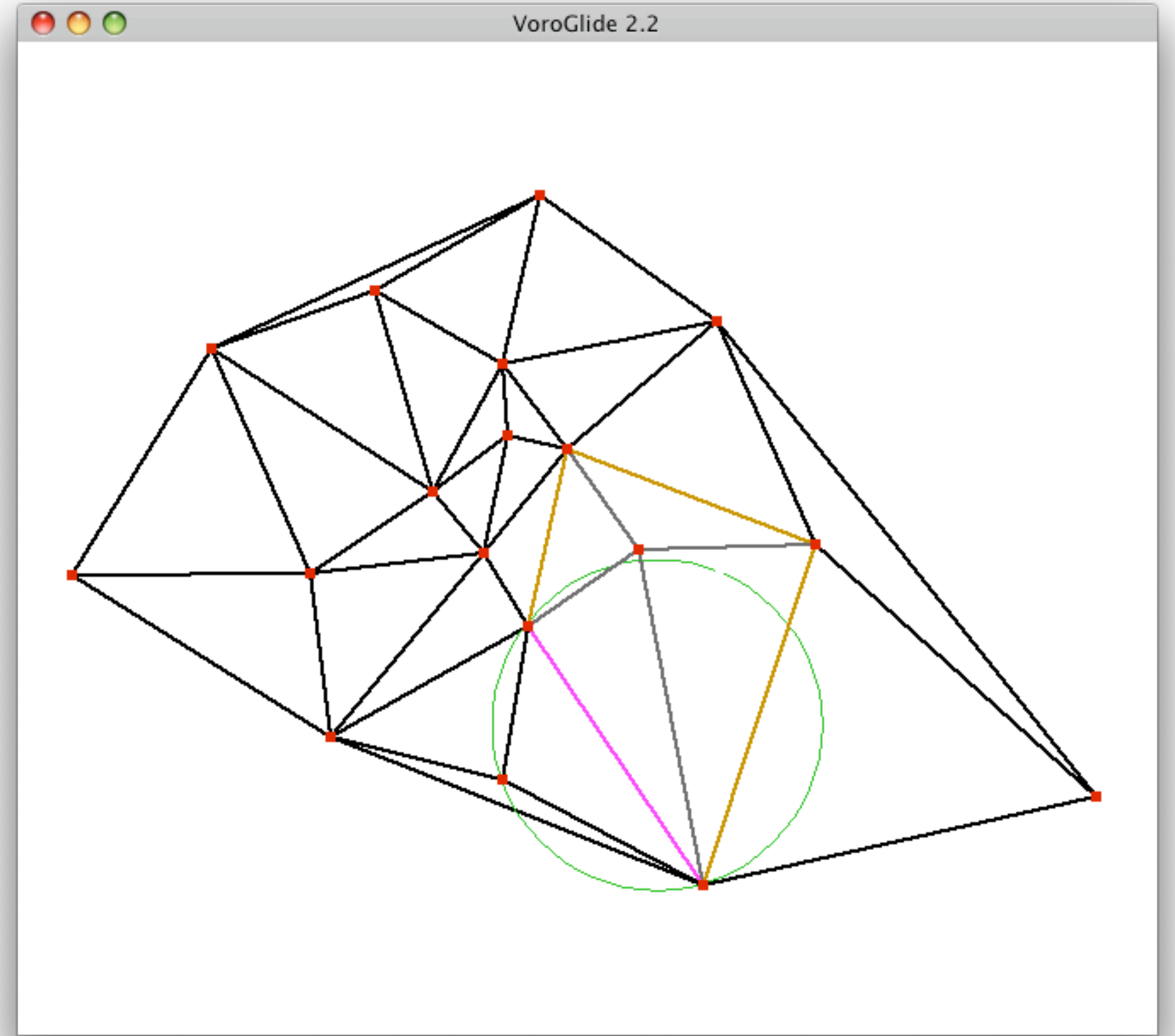


Top view



http://www.geometrylab.de/VoroGlide/

Inkrementelle Konstruktion der Delaunay-Triangulierung

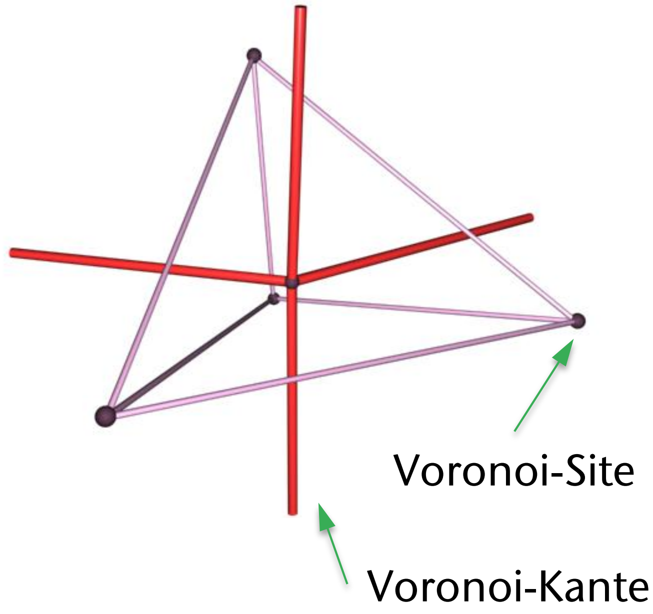


Verallgemeinerung des Voronoi-Diagramms

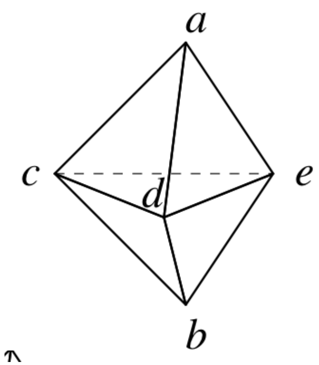
- Andere Distanz-Funktionen
- Andere Objekte als Sites
- Höhere Dimension
- Andere Äquivalenzklassen
- …

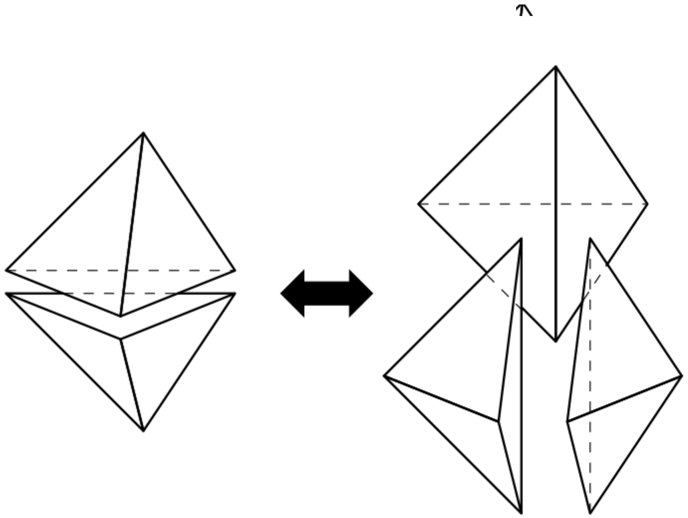
Voronoi / Delaunay in 3D

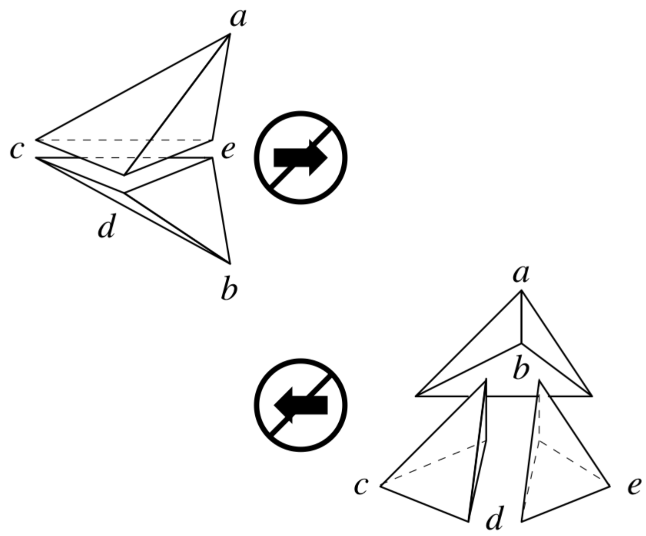
- Delaunay-Tetraeder
- Bisektoren = Ebenen



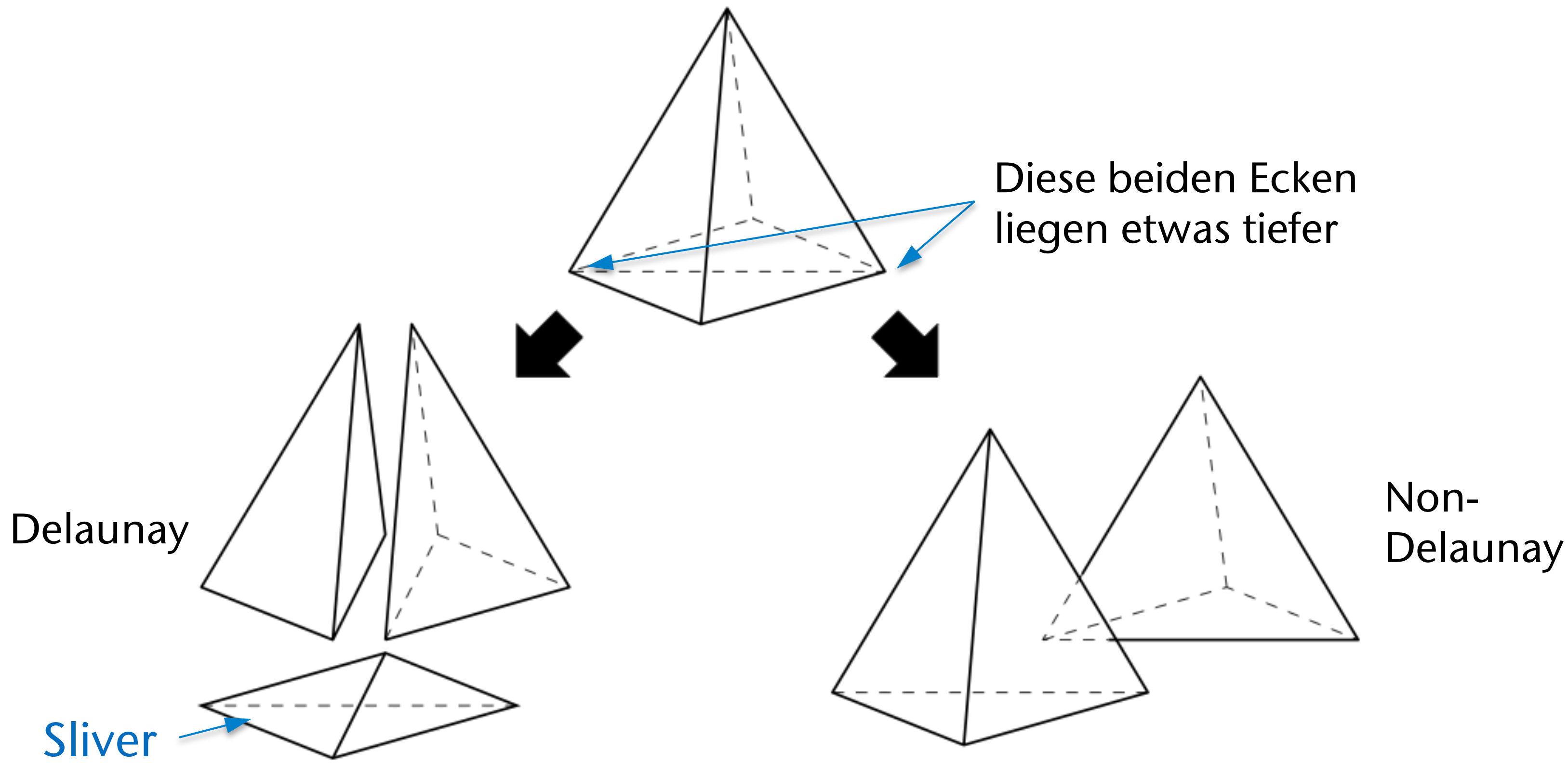
- Edge-Flip:







- Slivers in 3D Delaunay Tetrahedralizations:



- Fazit: die max-min-Winkel-Eigenschaft gilt nur in 2D! ☹

- Komplexität:
  Ein Voronoi-Diagramm über n Punkten im d-dim. Raum enthält in jeder Dimension j, $0 \leq j \leq d$-1, eine Anzahl f_j von Facetten, wobei alle

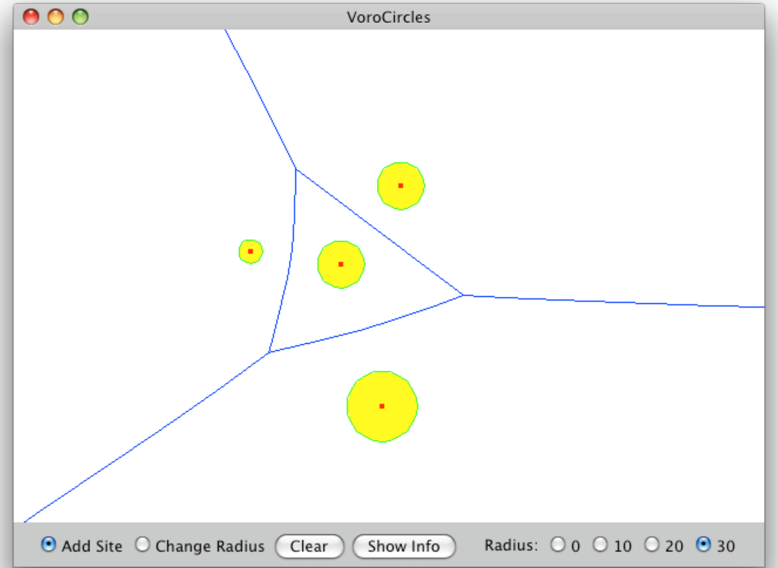
$$f_j \in O\left(n^{\lceil \frac{d}{2} \rceil}\right)$$

Das Voronoi-Diagramm mit additiven Gewichten

- Distanz-Funktion zwischen Punkt **x** und Site $\mathbf{p}_i$ =

$$d(\mathbf{x}, \mathbf{p}_i) = \|\mathbf{x} - \mathbf{p}_i\| - r_i$$

- A.k.a. Appolonius-Diagramm
- Bisektoren = hyperbolische Bögen
- Beispiel:



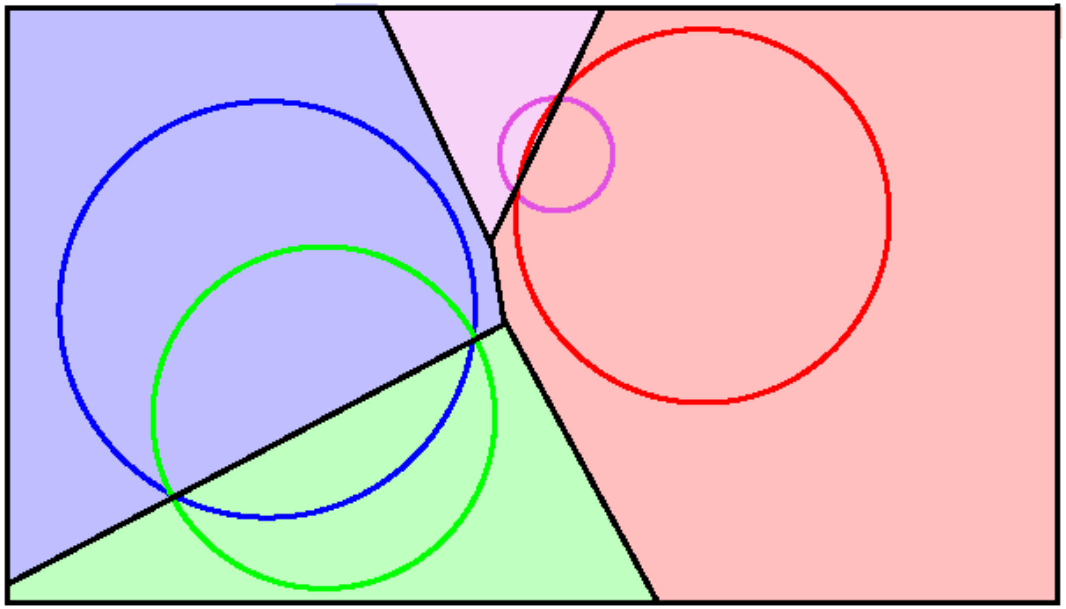
http://www.geometrylab.de/VoroAdd/index.html

Das Power-Diagram

- Distanzfunktion:

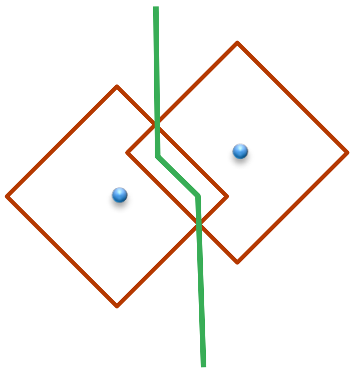
$$d(\mathbf{x}, \mathbf{p}_i) = (\mathbf{x} - \mathbf{p}_i)^2 - r_i$$

- Bisektoren = Geraden
- Beispiel:

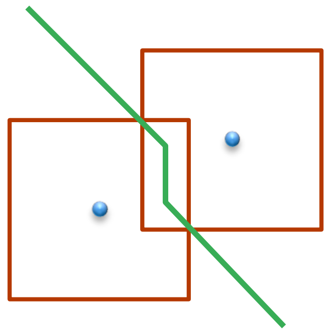


Andere Distanz-Funktionen

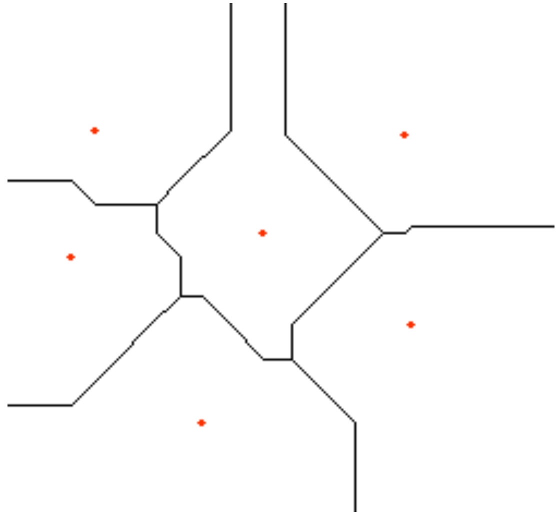
- Voronoi-Diagramm mit L_1- und L_∞-Norm:

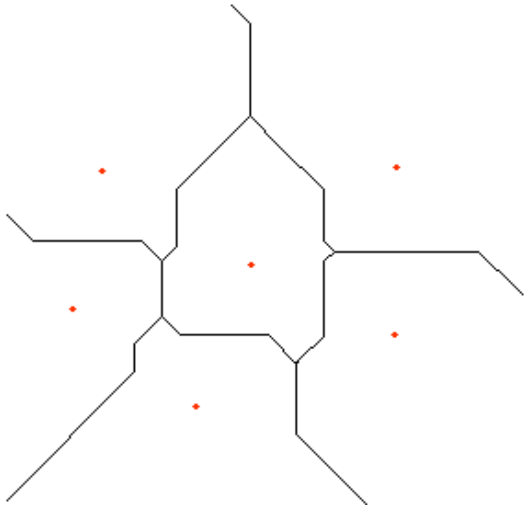


L_1-Norm
(Manhattan norm)



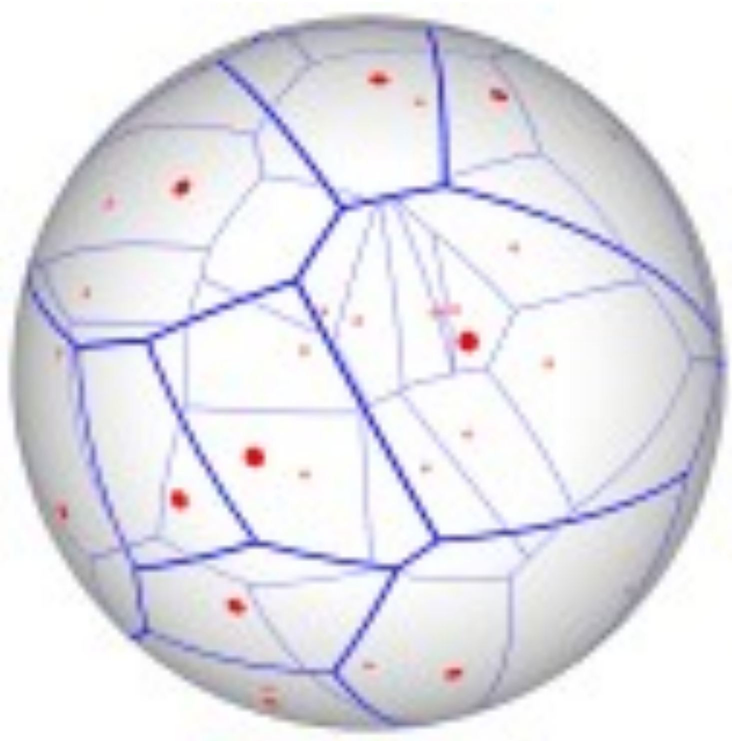
L_∞ - Norm
(supremum/max-norm)





Voronoi-Diagramme auf anderen Mannigfaltigkeiten

- Z.B. auf der Kugel:
  - Bisektoren = Großkreise



Higher-Order Voronoi Diagrams

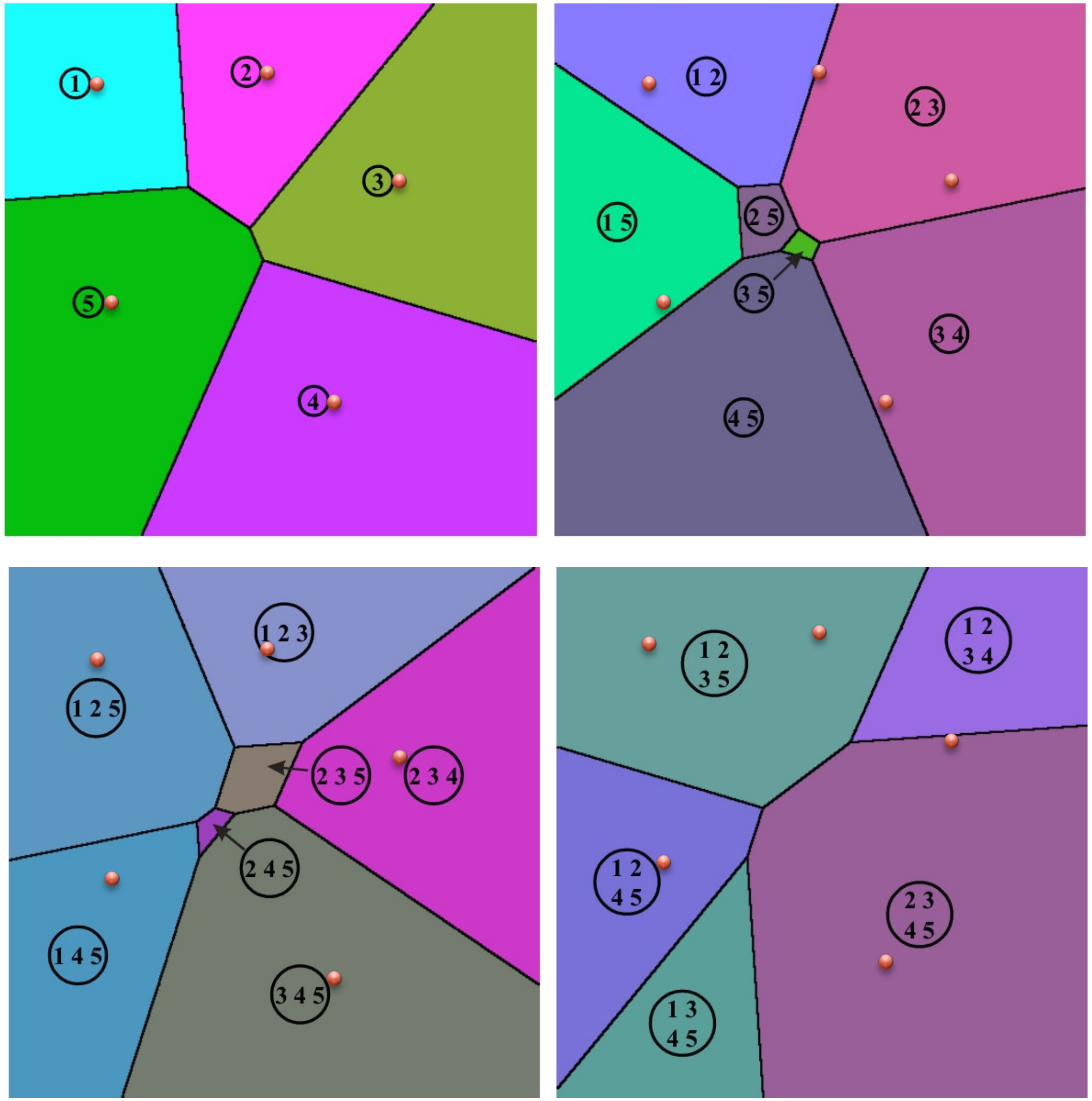
- In einem Voronoi-Diagramm *k*-ter Ordnung $V_k(S)$ gehören alle diejenigen Punkte des Raumes zur selben Voronoi-Region, die die selben *k* nächsten Nachbarn aus *S* haben

- Unterschiede zum klassischen Voronoi-Diagramm:
  - Ein Bisektor kann zu mehreren Begrenzungskanten (-ebenen) beitragen
  - Eine Voronoi-Region muß ihre Generatoren (Sites) nicht mehr enthalten

- Beispiel:

1-st order

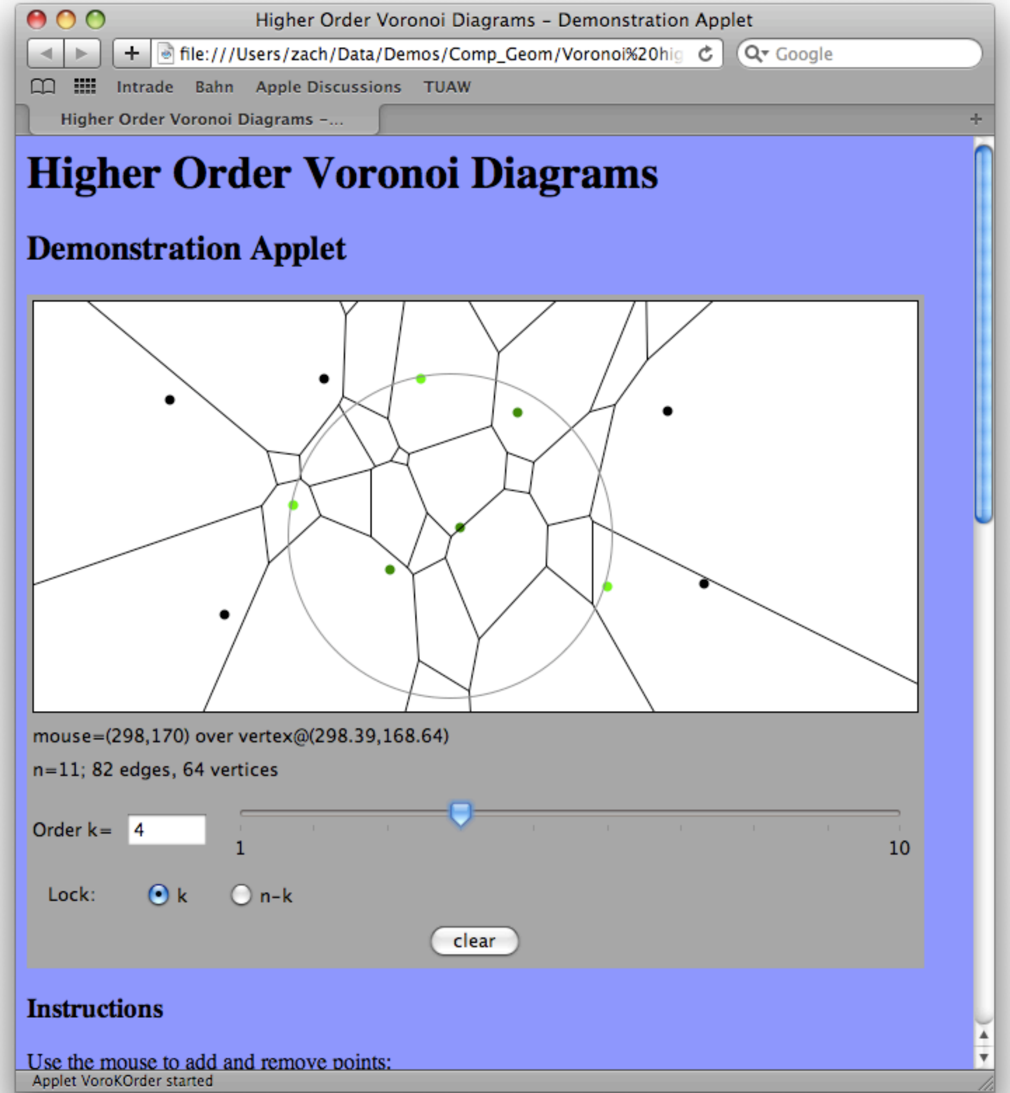
2-nd order



3-rd order

4-th order

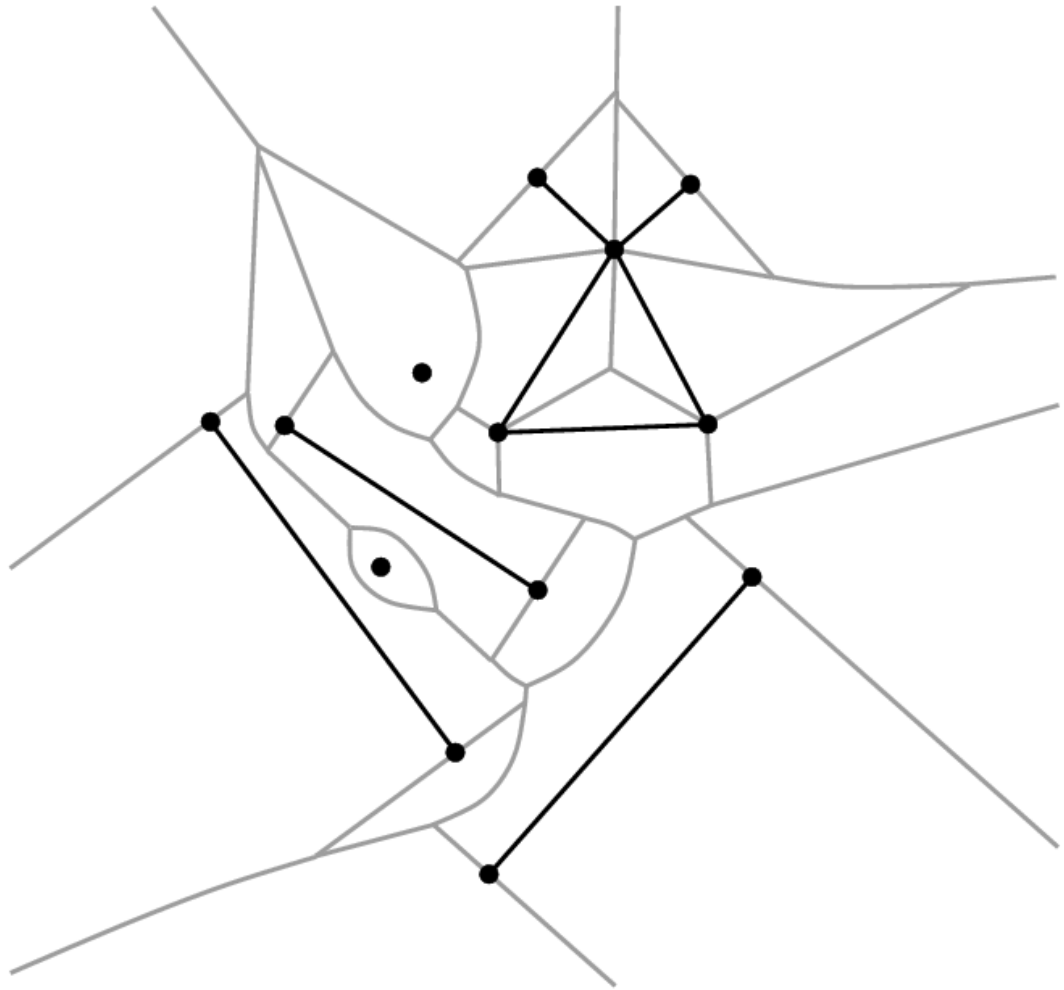
Demo



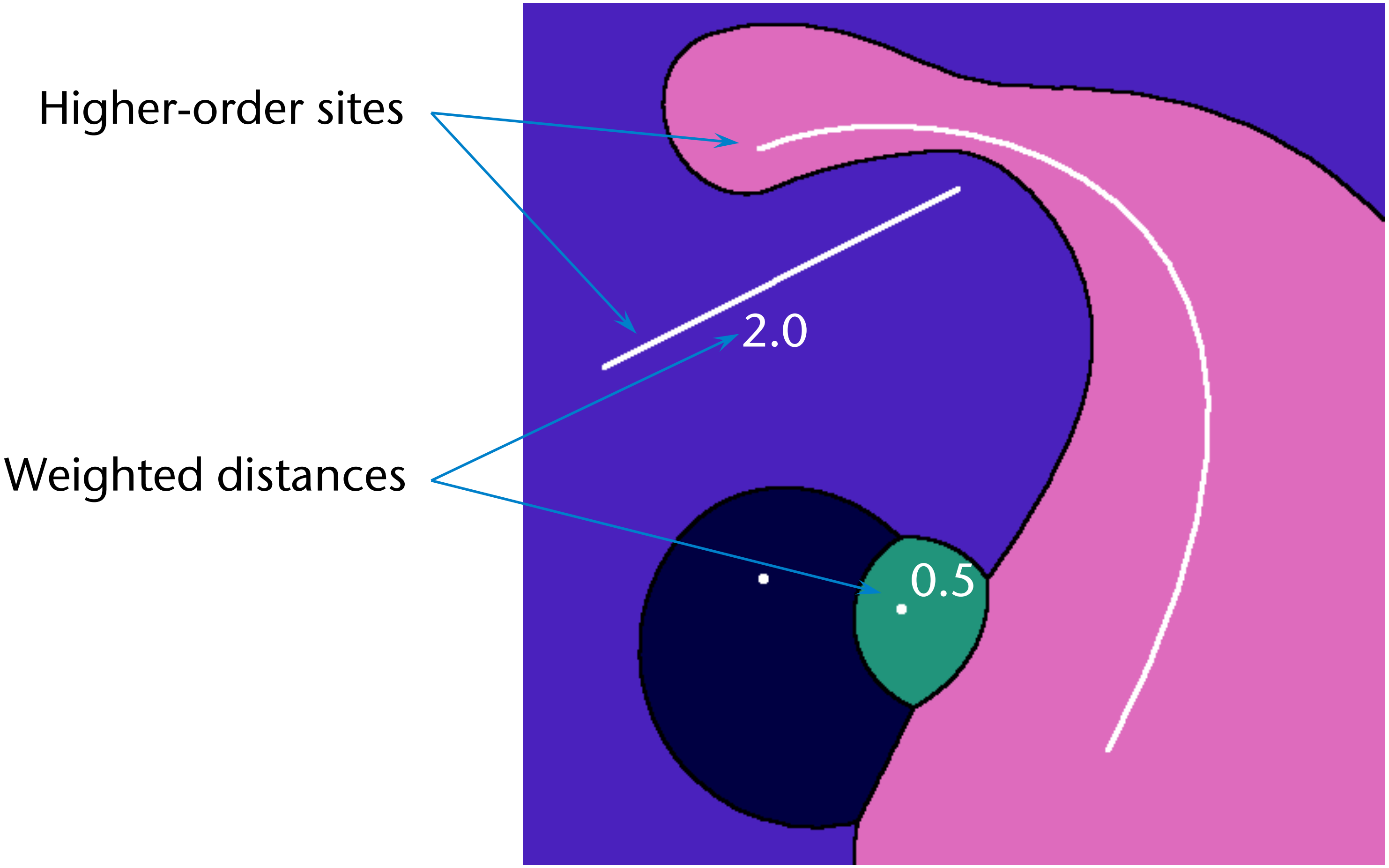
Andreas Pollack - http://www.pollak.org/en/otherstuff/informatics/voronoi/

Voronoi-Diagramm von Liniensegmenten

- Sites sind jetzt Punkte + Liniensegmente
- Bisektoren = Geraden + Parabeln
- Beispiel:



- Example with weighted sites and higher-order sites:



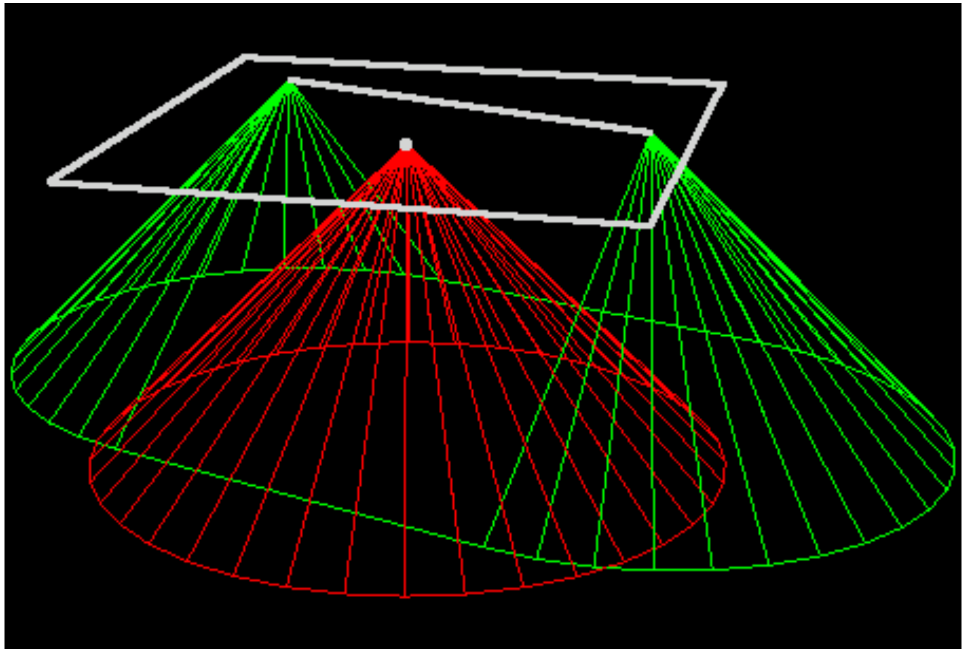
The "Cone Trick" for Higher-Order Sites

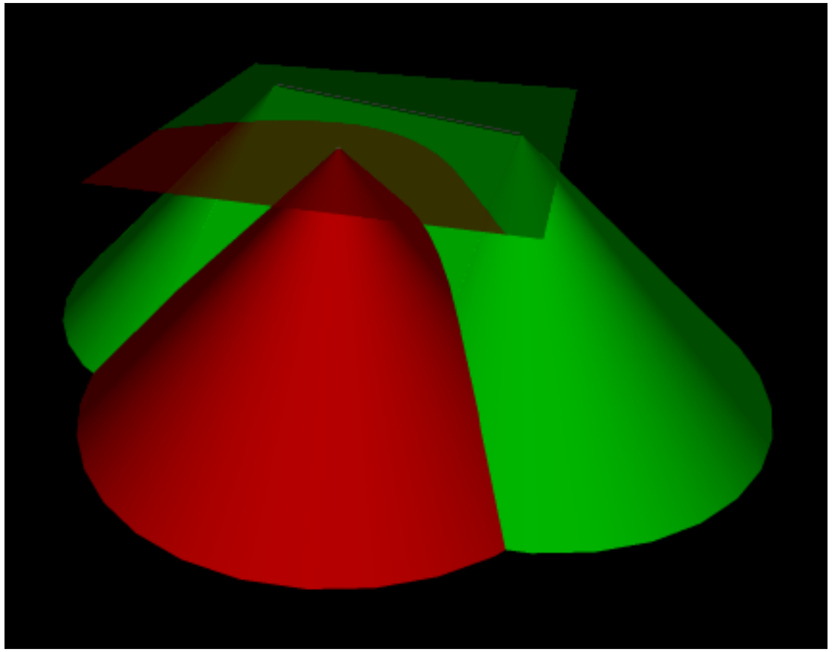
- Observation: the surface in 3D, generated by

$$f(x, y) = (x, y, d(x, y))$$

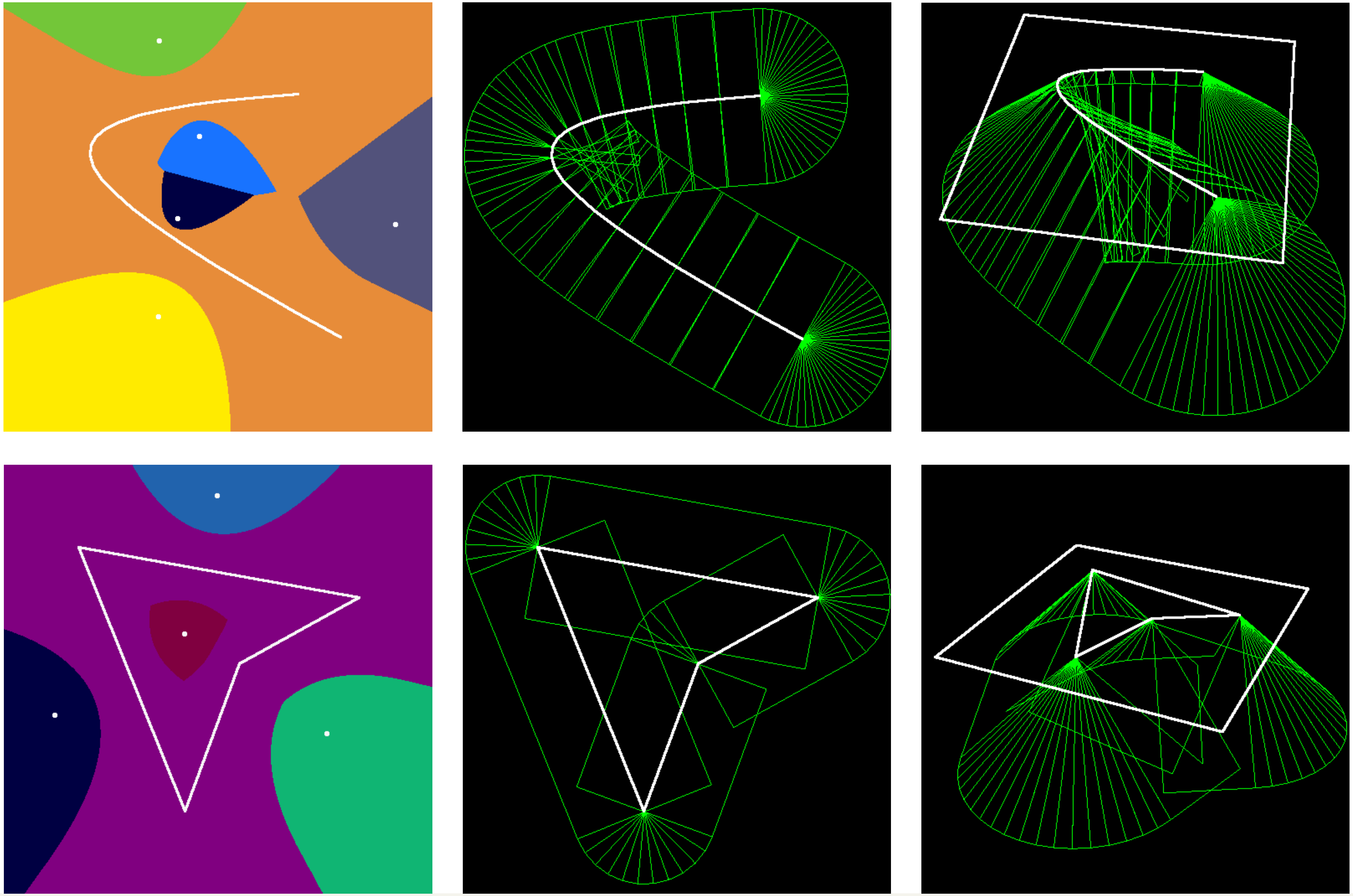
where $d(x,y)$ = distance from the Voronoi site is a swept cone

- Idea: approximate distance function by a mesh

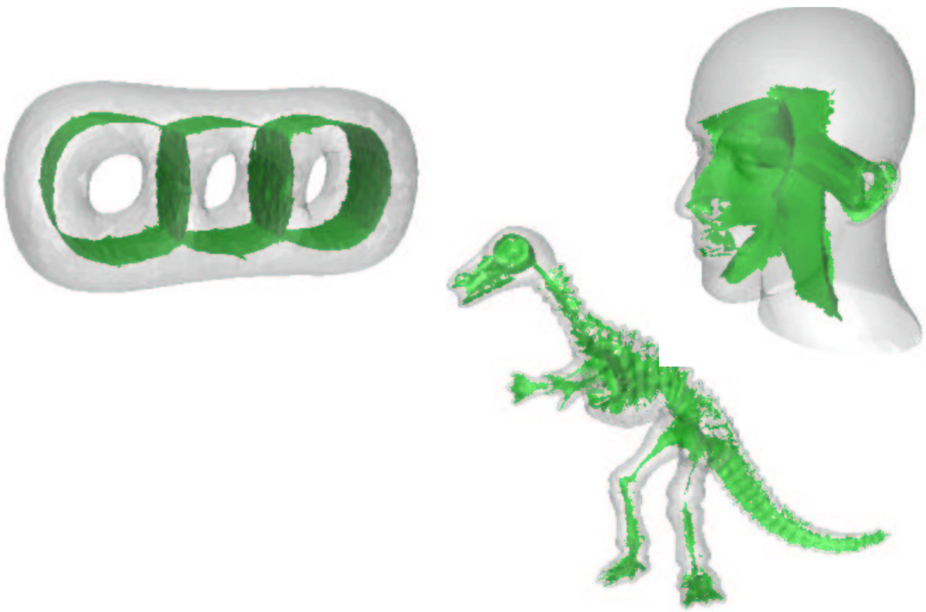


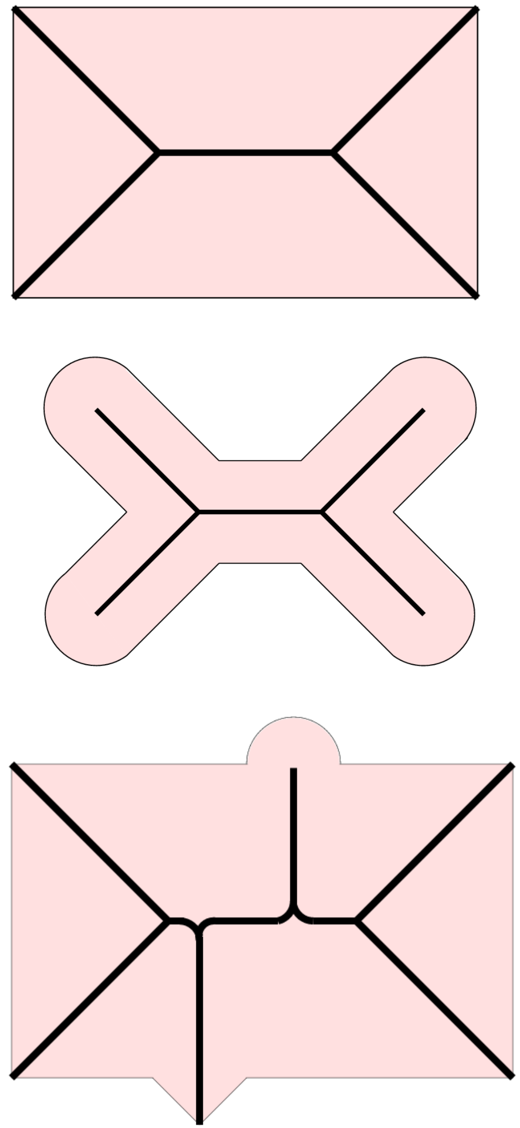


More Example Distance Meshes

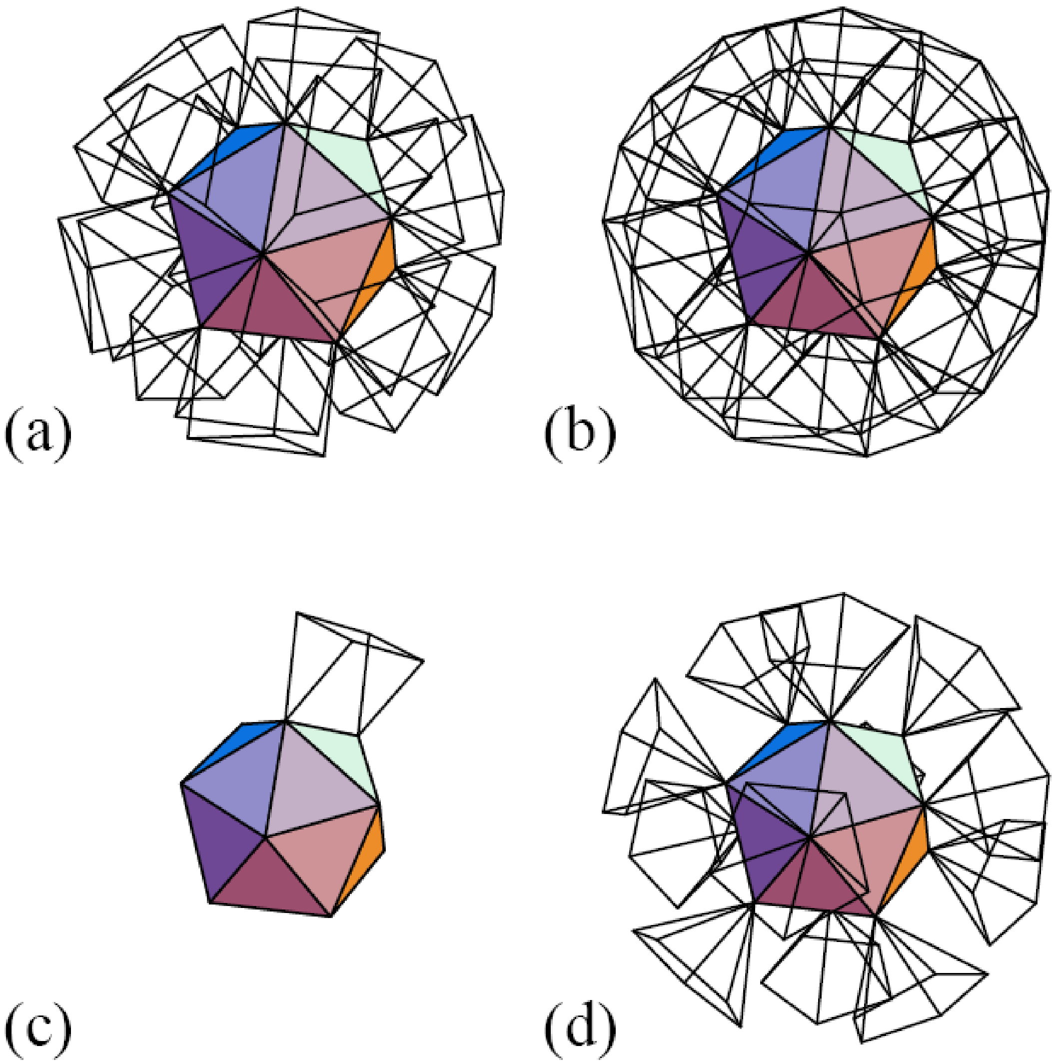


- Skeleton oder Medial Axis
  - Besonders im Fall von geschlossenen Objekten
  - Alle Punkte, die gleich weit von 2 Punkten des Randes eines Objektes entfernt sind
  - Problem: Stabilität





Äußere Voronoi-Regionen eines konvexen Polyeders



The external Voronoi regions of …

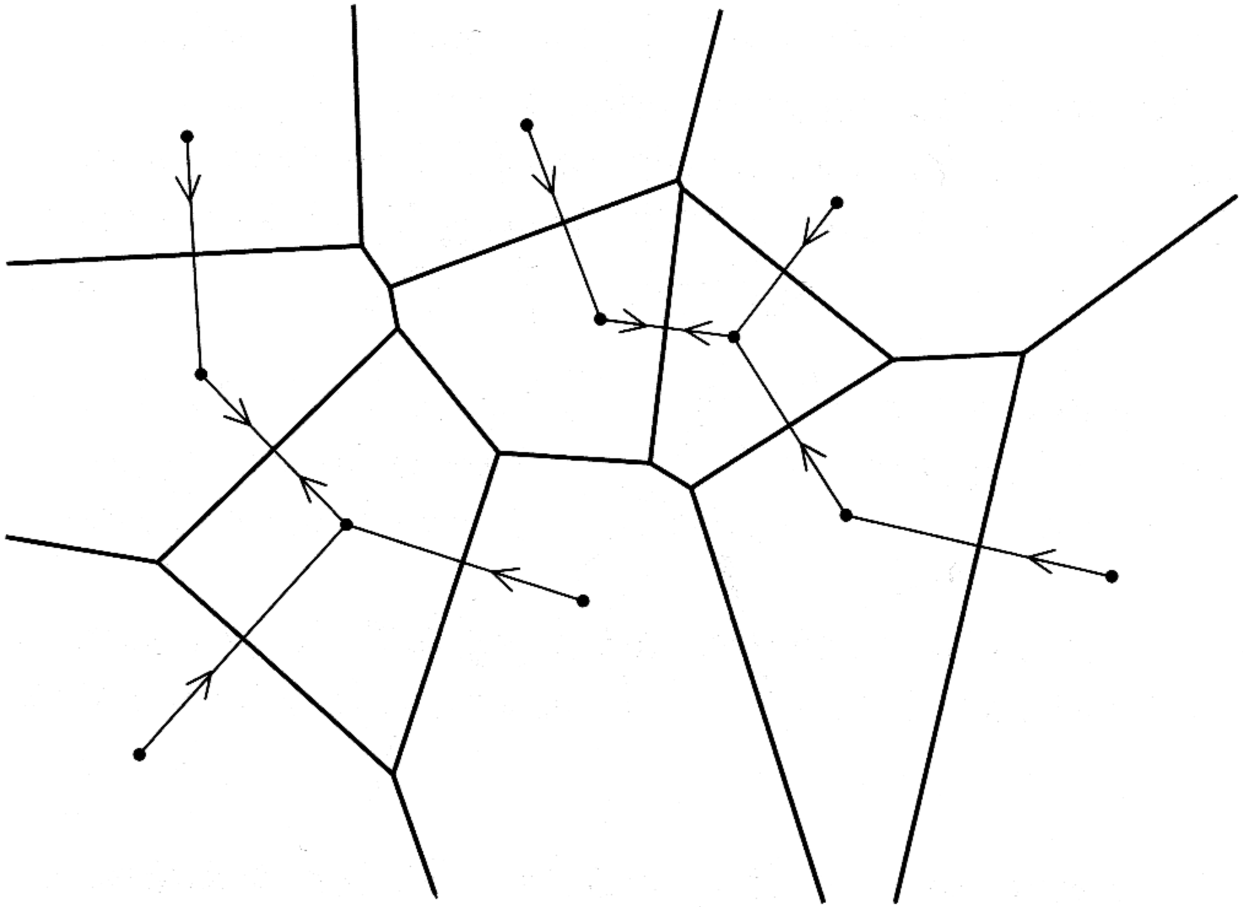
(a) faces

(b) edges

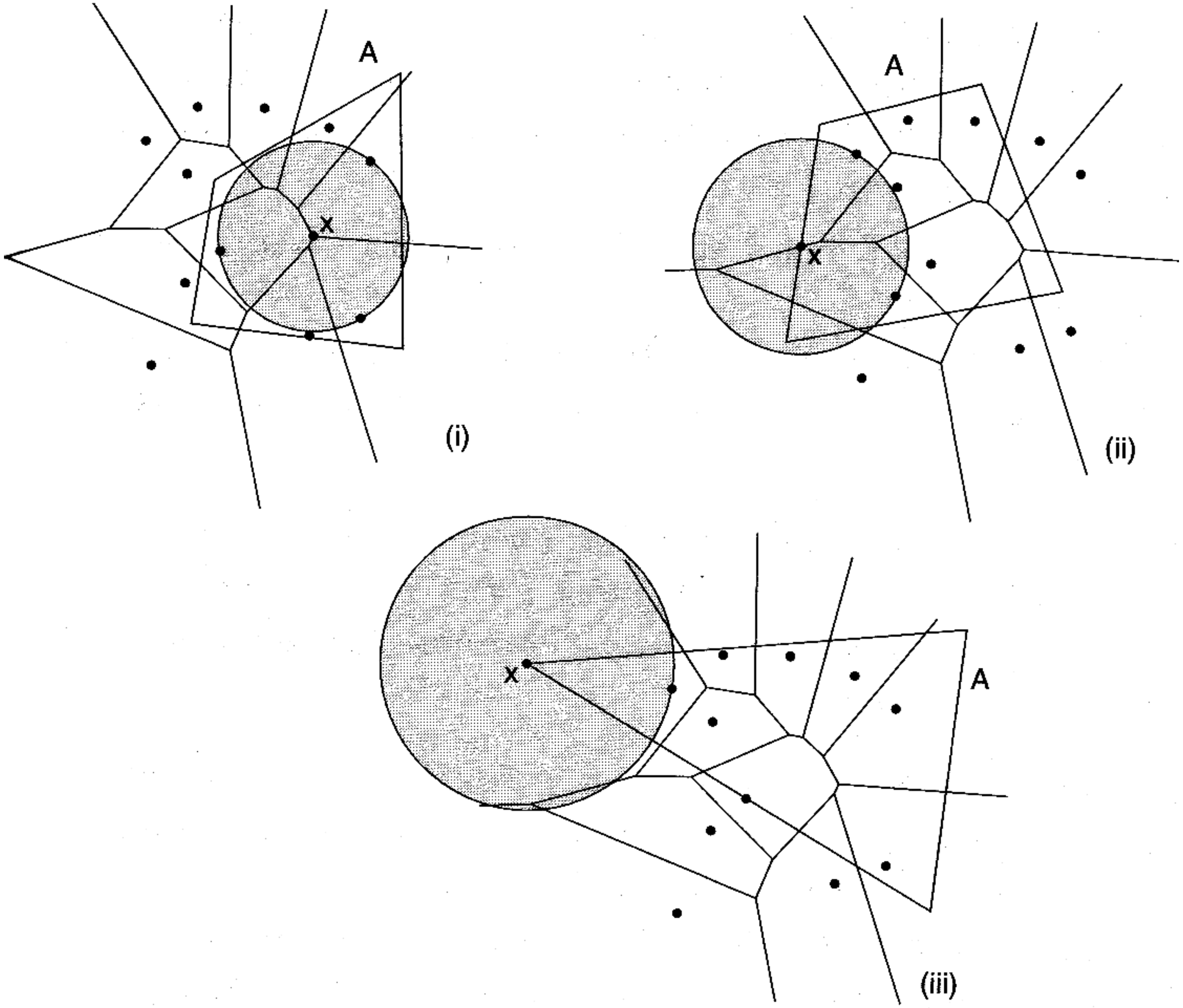
(c) a single edge

(d) vertices

Beispiel zu NNG(S) ⊆ D(S)



Maximale, leere Kreise

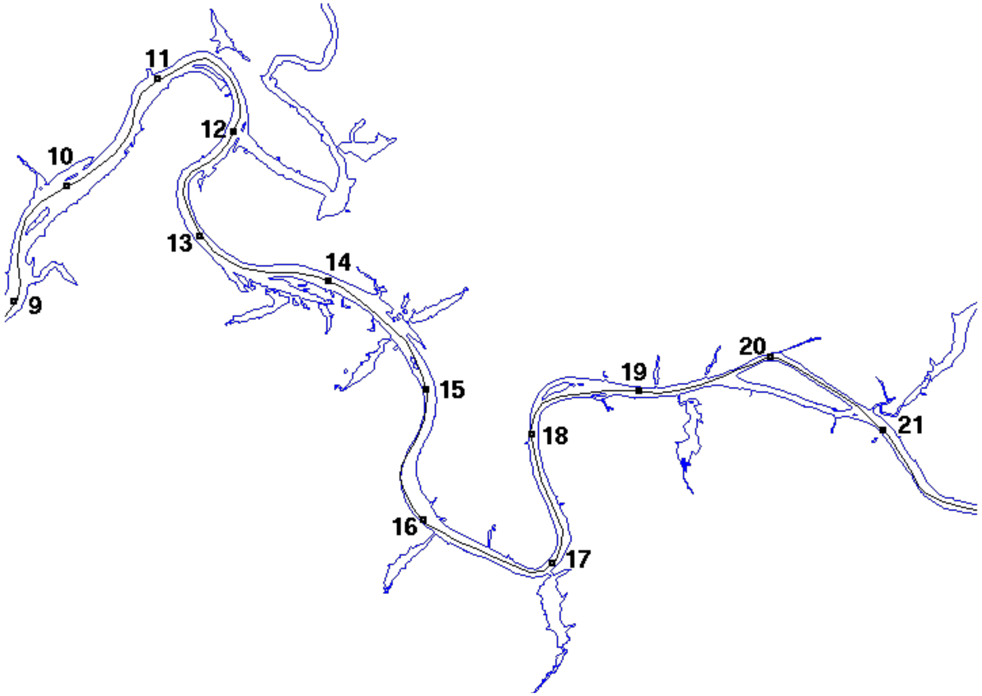


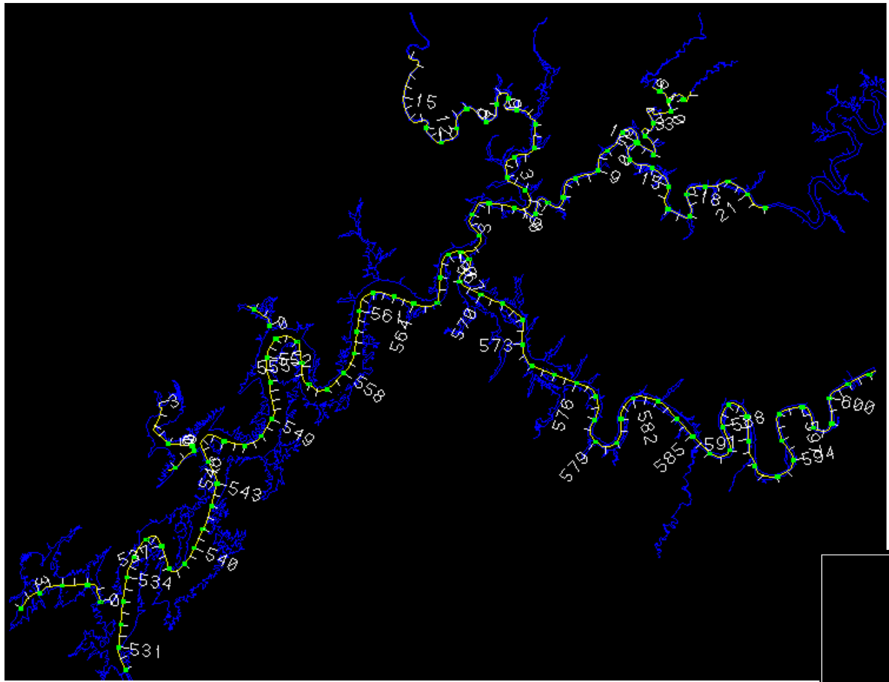
Anwendungsgebiete der Voronoi-Diagramme

- **Anthropology and Archeology** -- Identify the parts of a region under the influence of different Neolithic clans, chiefdoms, ceremonial centers, or hill forts.
- **Astronomy** -- Identify clusters of stars and clusters of galaxies (Here we saw what may be the earliest picture of a Voronoi diagram, drawn by Descartes in 1644, where the regions described the regions of gravitational influence of the sun and other stars.)
- **Biology**, **Ecology**, **Forestry** -- Model and analyze plant competition ("Area potentially available to a tree", "Plant polygons")
- **Cartography** -- Piece together satellite photographs into large "mosaic" maps
- **Crystallography and Chemistry** -- Study chemical properties of metallic sodium ("Wigner-Seitz regions"); Modelling alloy structures as sphere packings ("Domain of an atom")
- **Finite Element Analysis** -- Generating finite element meshes which avoid small angles
- **Geography** -- Analyzing patterns of urban settlements
- **Geology** -- Estimation of ore reserves in a deposit using information obtained from bore holes; modelling crack patterns in basalt due to contraction on cooling
- **Geometric Modeling** -- Finding "good" triangulations of 3D surfaces
- **Marketing** -- Model market of US metropolitan areas; market area extending down to individual retail stores
- **Mathematics** -- Study of positive definite quadratic forms ("Dirichlet tessellation", "Voronoi diagram")
- **Metallurgy** -- Modelling "grain growth" in metal films
- **Meteorology** -- Estimate regional rainfall averages, given data at discrete rain gauges ("Thiessen polygons")
- **Pattern Recognition** -- Find simple descriptors for shapes that extract 1D characterizations from 2D shapes ("Medial axis" or "skeleton" of a contour)
- **Physiology** -- Analysis of capillary distribution in cross-sections of muscle tissue to compute oxygen transport ("Capillary domains")
- **Robotics** -- Path planning in the presence of obstacles
- **Statistics and Data Analysis** -- Analyze statistical clustering ("Natural neighbors" interpolation)
- **Zoology** -- Model and analyze the territories of animals

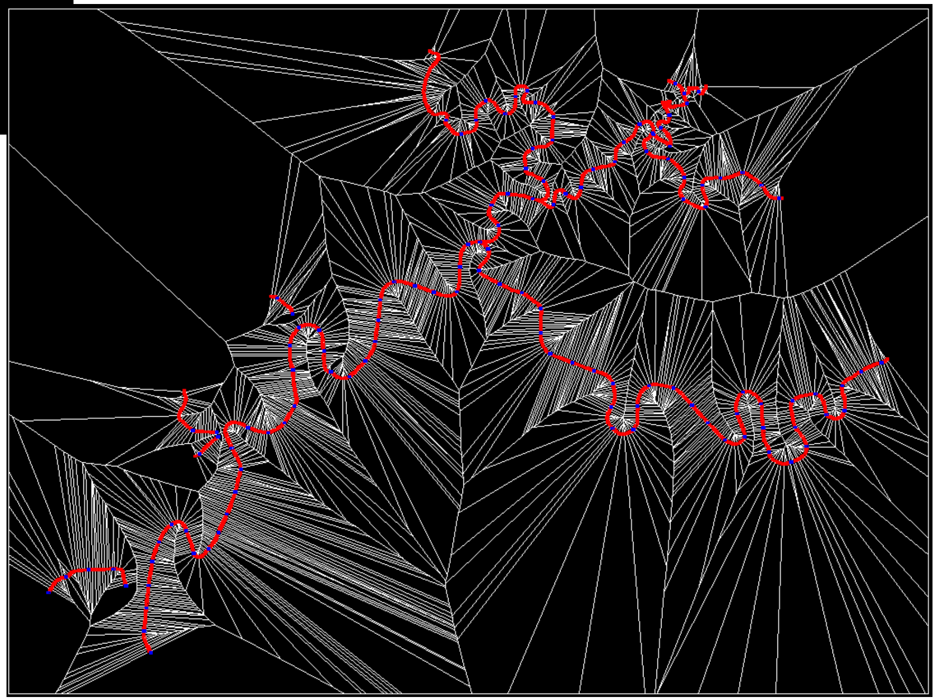
Anwendung: das River-Mile-Koordinatensystem

- Das River-Mile-Koordinatensystem:
  - Wird gerne in großen Wasserwegesystemen angewendet
  - Koordinaten eines Punktes in der Ebene = (l, q) wobei
    l = gemessen entlang der Mittellinie des Flusses,
    q = Entfernung von Punkt $(l, 0)$ senkrecht zur Tangente in $(l, 0)$
- Aufgabe:
  gegeben ein Punkt $(x,y) \longrightarrow$
  welche Koord. (l, q) hat er?





Zerlegung der Mittellinie in einen fein aufgelösten Polygonzug



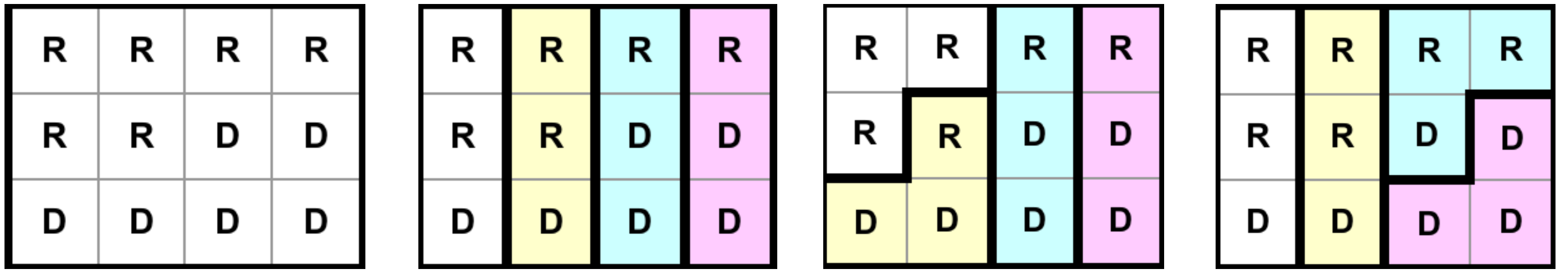
Voronoi-Diagramm dazu

Voronoi-Clustering

- Aufgabe:
  - Gegeben: Menge von Punkten
  - Gesucht: Partitionierung der Punktmenge in "Cluster"
  - Clustering = maximal intra-cluster similarity and
    minimal inter-cluster similarity
    = minimal intra-cluster distance and
    maximal inter-cluster distance

Gute Einteilung in Wahlbezirke (*Redistricting*)

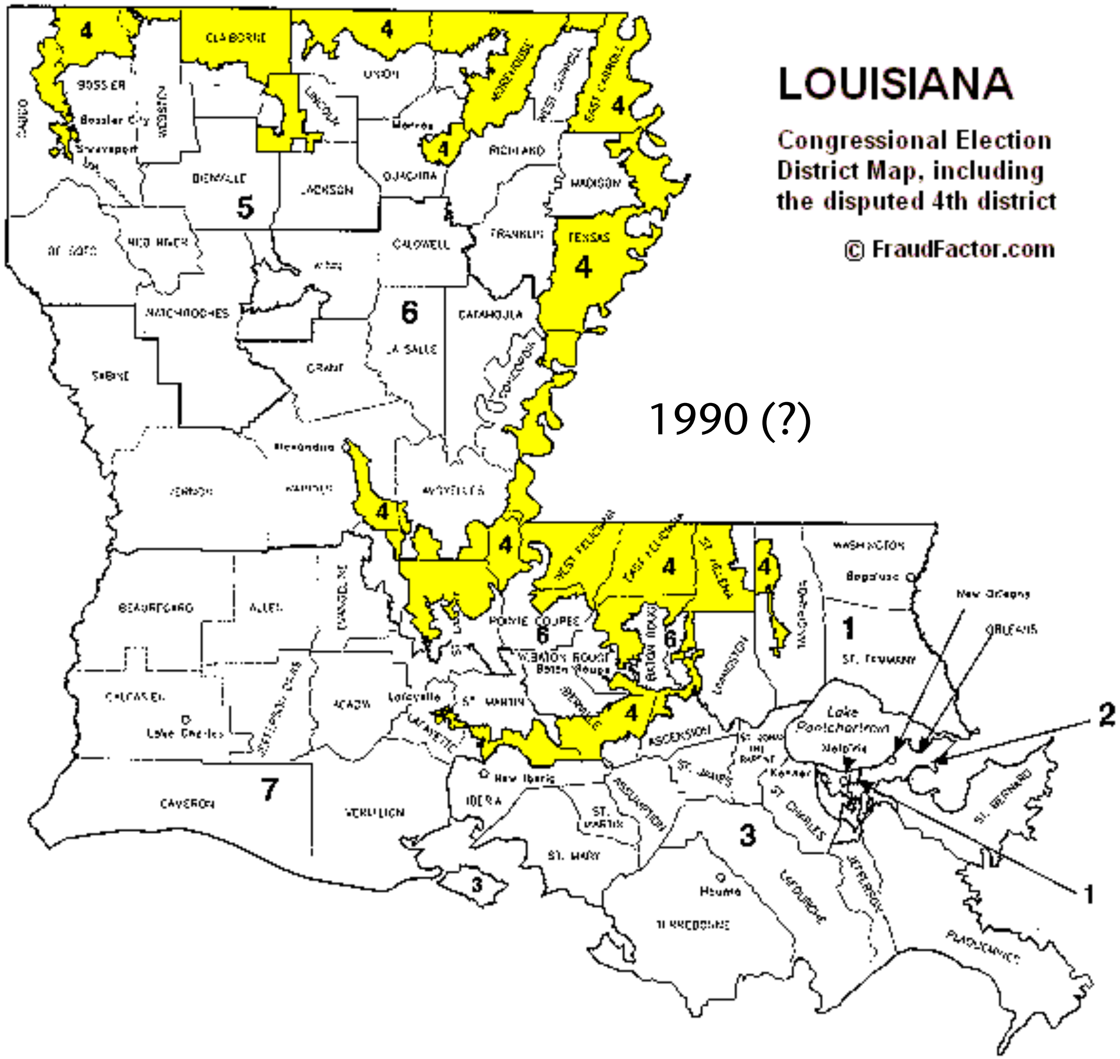
- Das Fairness-Prinzip: "one man, one vote"
  - Ganz einfach … oder?
- Einfaches Beispiel:



Wählerverteilung — Gleiche Anzahl Repräsentanten — Demokraten gewinnen — Republikaner gewinnen

- Gesetzliche Kriterien für Wahlbezirke in den USA:
  - Gleiche Anzahl Wähler
  - Jeder Bezirk soll zusammenhängend sein
  - "Kompaktheit" (ist im US-Gesetz aber nicht klar definiert)

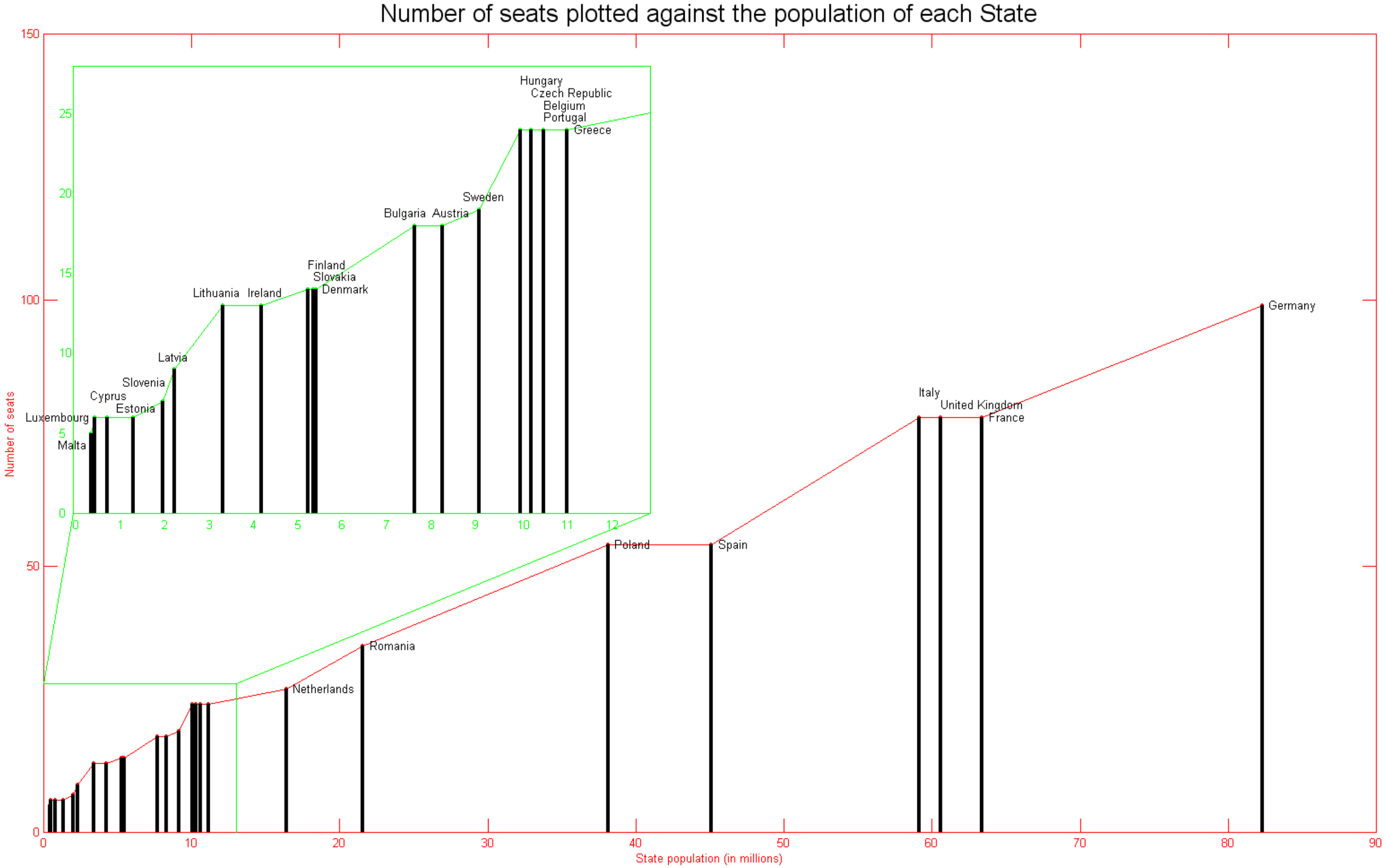
- Böses Beispiel:



1990 (?)

"In gerrymandered election districts, the voters don't choose their politicians - the politicians choose their voters!"

- Ähnlicher Effekt bei Europawahlen: die Stimme eines Wählers in Malta oder Luxembourg hat 10x mehr Gewicht als die eines deutschen Wählers!



- Eine mögliche Definition von Kompaktheit:

  Sei $\mathcal{D} = \{D_1, \ldots, D_k\}$

  eine Menge von Wahlbezirken (*districts*).

  Jeder Distrikt $D_i = \{p_j, \ldots, p_l\} \subset P = \{p_1, \ldots, p_n\}$

  enthält eine Menge von Wählern p_i .

  Die Kompaktheit eines Distrikts ist

$$c(D) = \sum_{i,j=1}^{|D|} d(p_i, p_j)$$

  Die Gesamtkompaktheit der Einteilung in Distrikte ist

$$c(\mathcal{D}) = \sum_{i=1}^{k} c(D_i)$$

- Theorem:

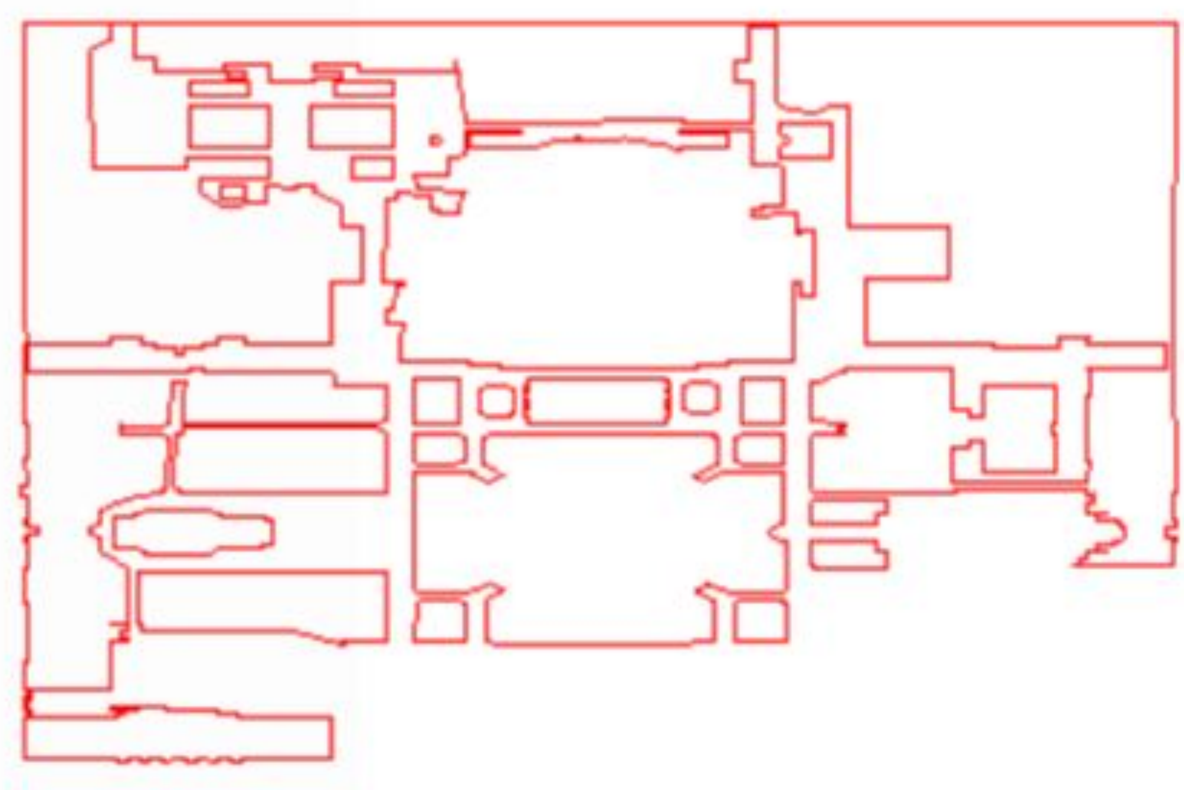
  Eine optimale Aufteilung in Wahlbezirke bzgl. Kompaktheit geht aus einem Power-Diagramm hervor.

- Aufgabe :
  - Konstruiere zu gegebener Menge Wähler $\{p_i\}$ eine Menge von Voronoi-Sites mit Gewichten, so daß
    - $\forall i : |D_i| = n$
    - Voronoi-Sites = "Wahllokale"
    - Gewicht = Maß für die Populationsdichte in dem zugehörigen Distrikt (kleines Gewicht = hohe Dichte)

- Ansatz :
  - Starte mit zufälligen Sites und Gewichten
  - Verschiebe Sites und Gewichte, bis $c(\mathcal{D})$ in lokalem Minimum

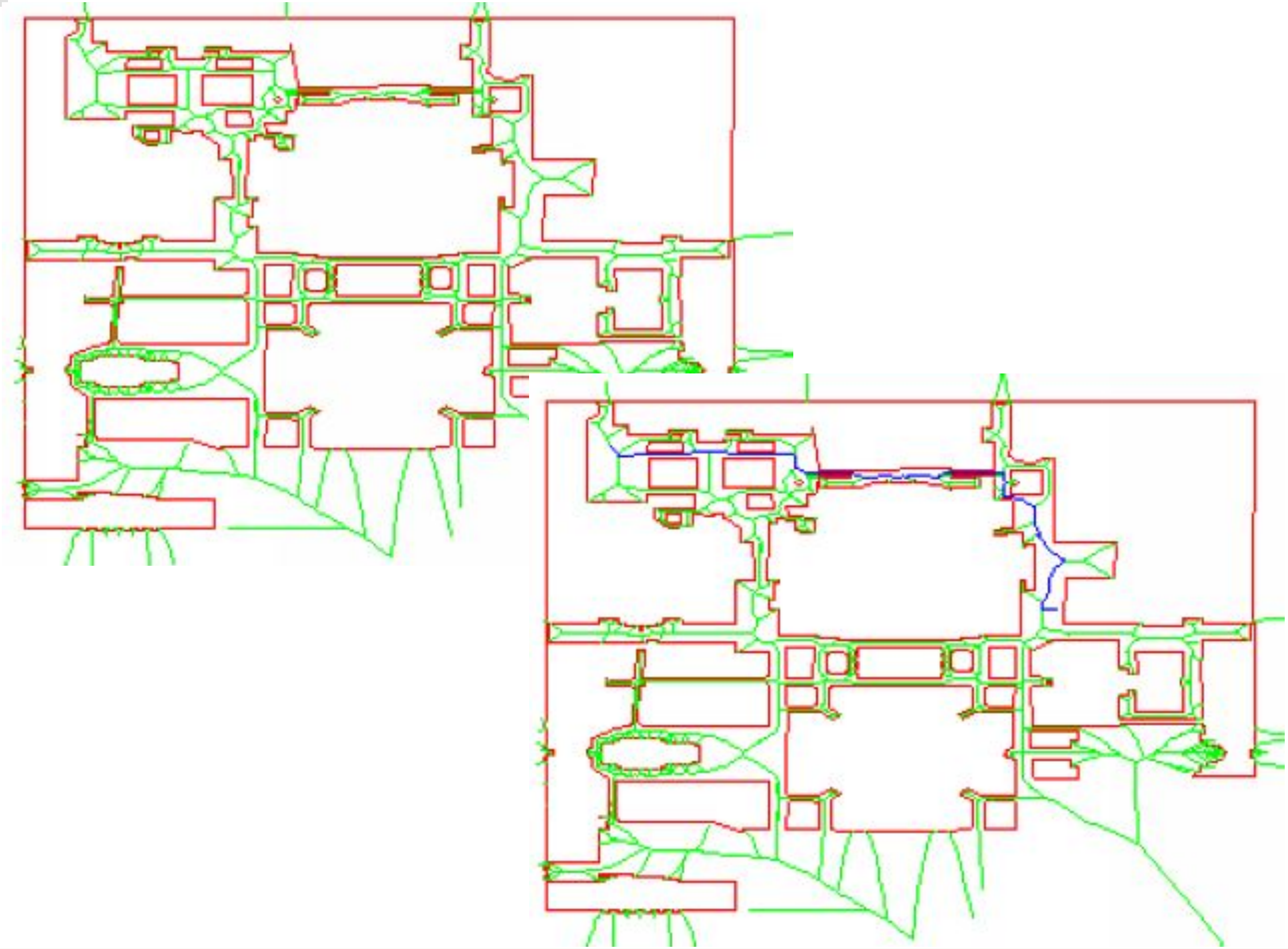
Pfadplanung

- Gegeben: Grundriß als Menge von Liniensegmenten
- Gesucht: Pfad (z.B. für autonomes Vehikel = Roboter) mit maximalem Abstand zu den Wänden



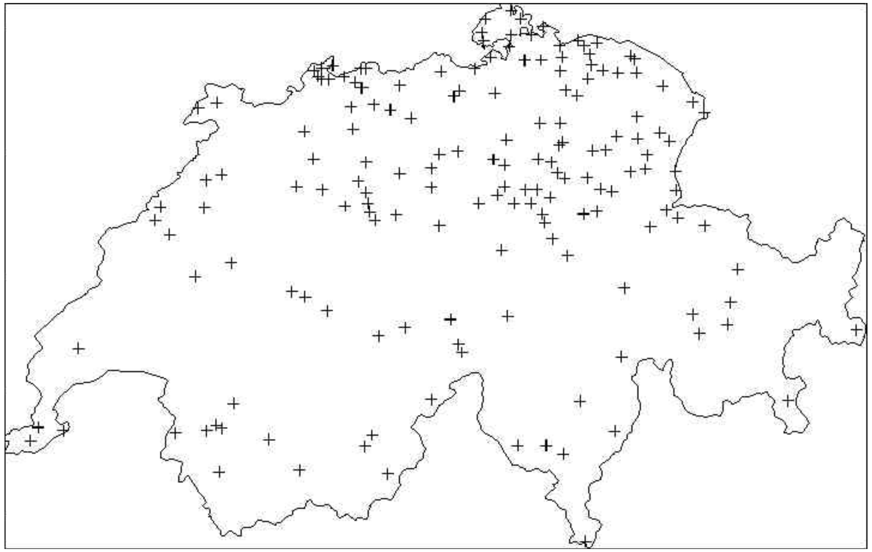
http://www.cs.columbia.edu/~pblaer/projects/path_planner/

- Lösung:
  - (Verallgemeinertes) Voronoi-Diagramm dazu konstruieren
  - Näheste Voronoi-Knoten zu Start- und Endpunkt suchen
  - Mit Dijkstra-Algo kürzesten Pfad von Start- zu End-Knoten durch Voronoi-Diagramm suchen

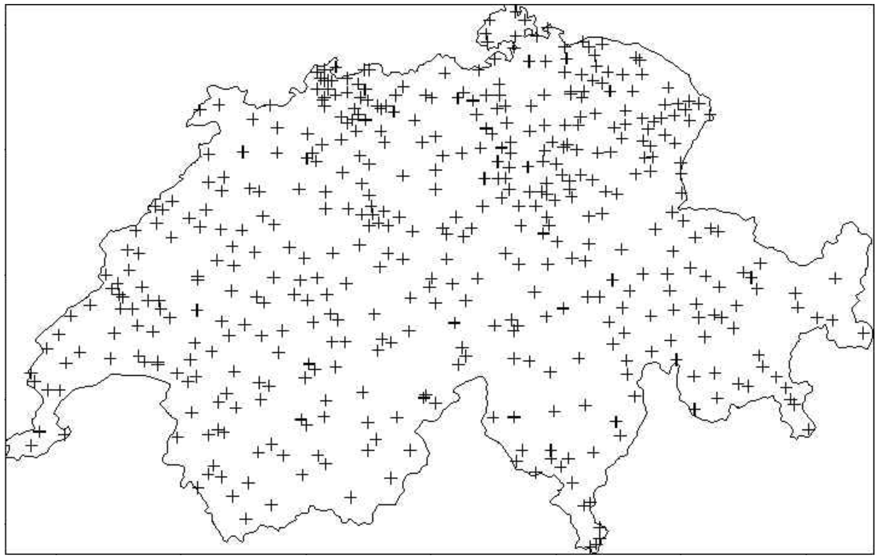


Bewertung von Samplings

- Beispiel: Wetterstationen
- Frage: wo ist die geringste Dichte?



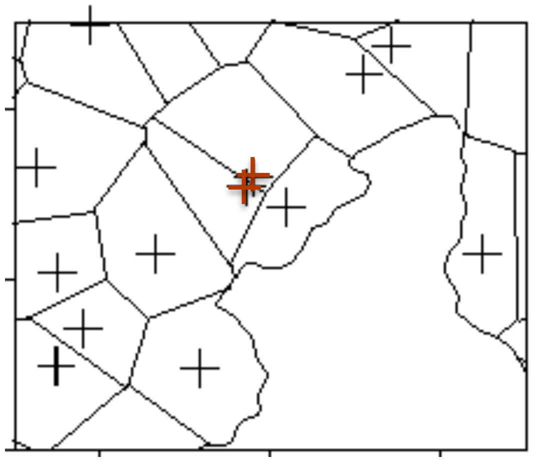
- Ideales Sampling → jeder Punkt würde eine Fläche von
$$\bar{A} = \frac{A}{n}$$
abdecken (A = Gesamt-fläche)

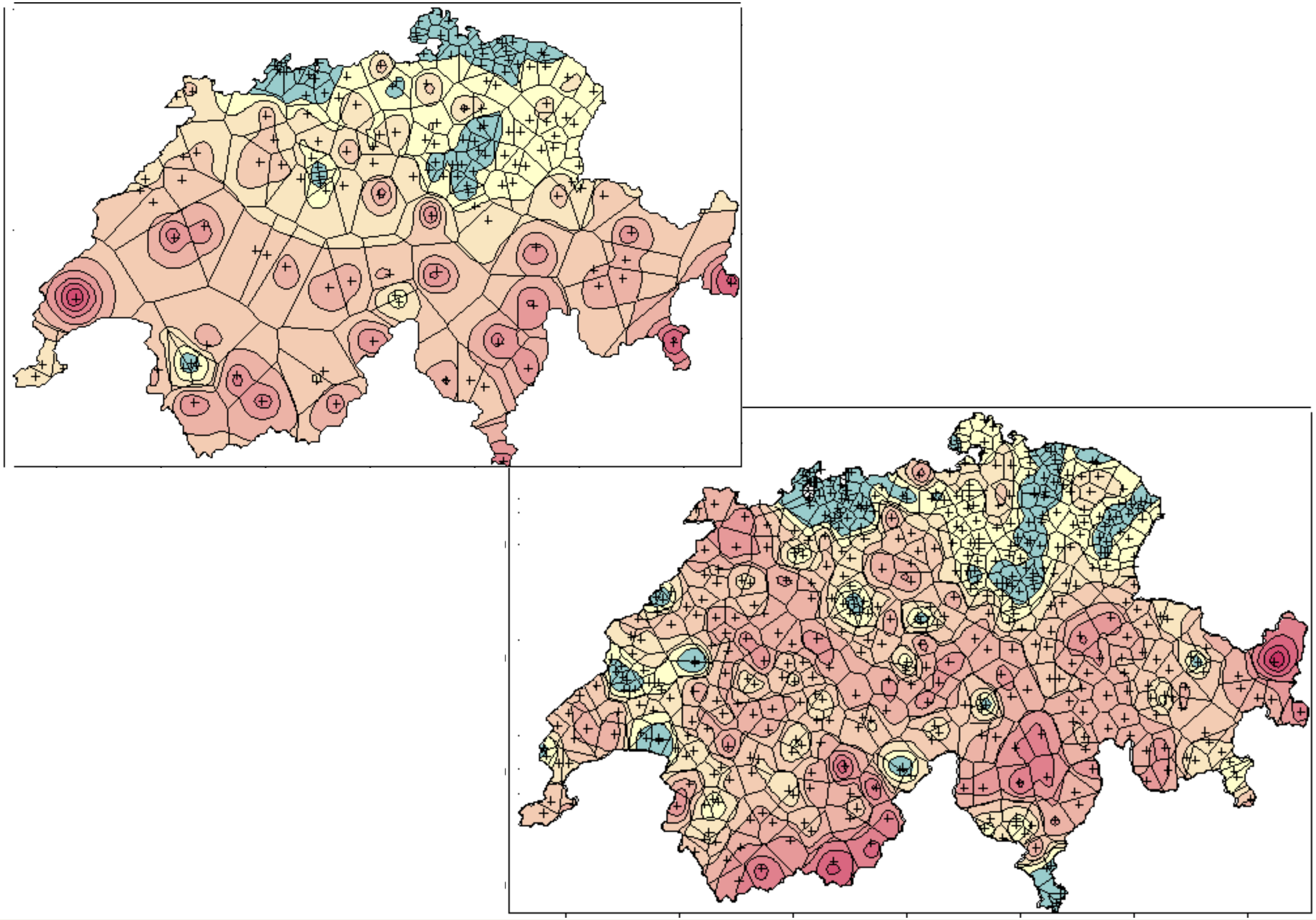


- Lösung:
  - Voronoi- und Delaunay-Diagramm berechnen
  - Relative Größe pro Zelle ist

$$A_i = \frac{V_i}{\bar{A}}$$

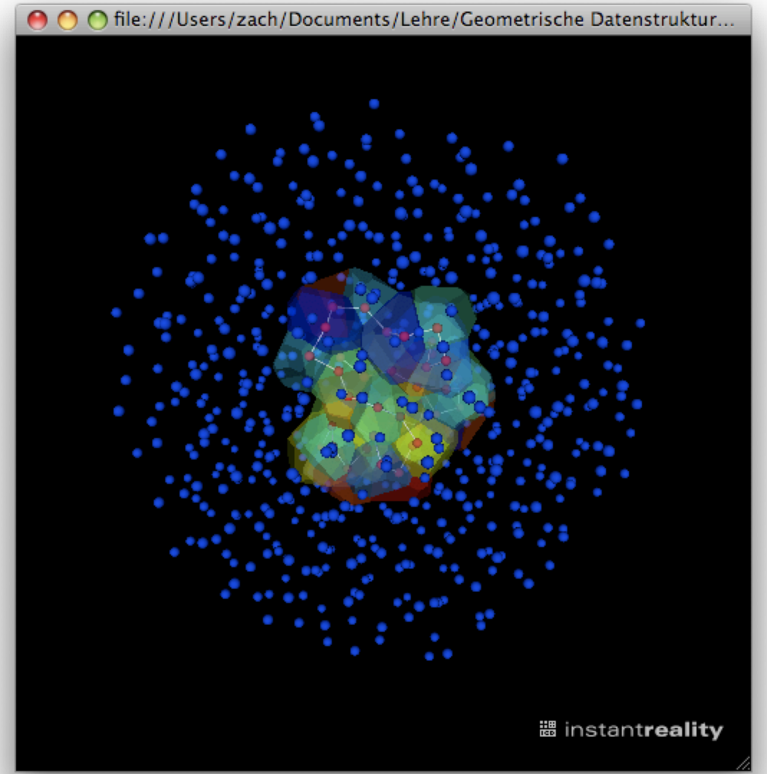
  - $A_i > 1 \rightarrow$ zu geringe Dichte
  - Sample-Punkte "bestrafen", falls sie dicht beieinander liegen relativ zur Größe der Zelle → Distanz zum nearest neighbor





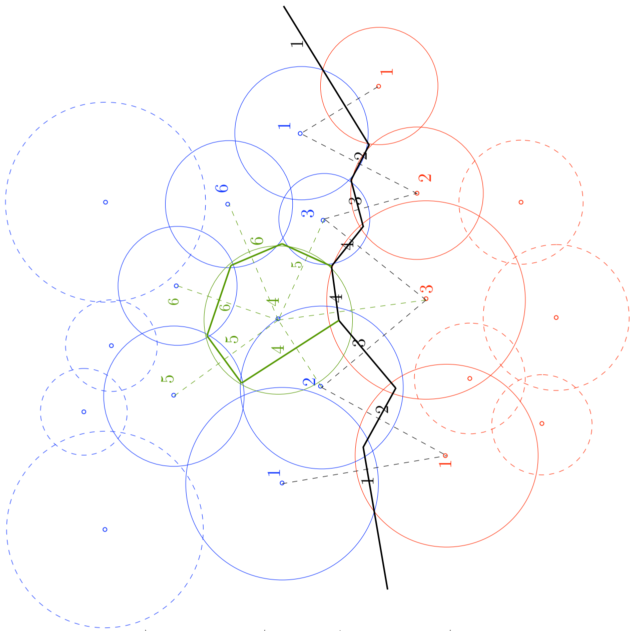
Protein-Struktur-Analyse

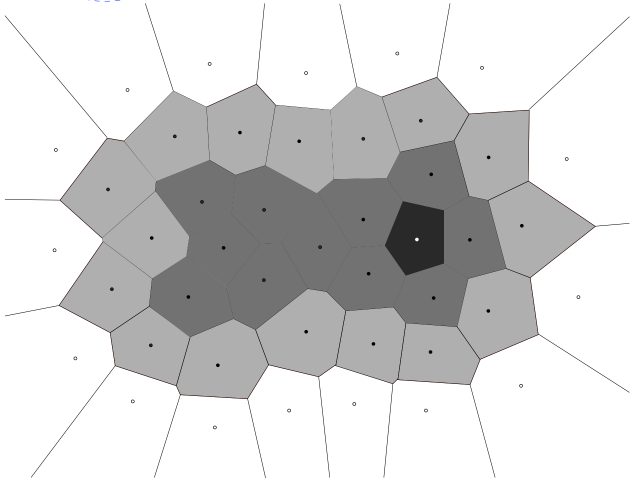
- Frage:
  - Wie sieht die aktive Oberfläche (= Interface) eines Moleküls aus?
  - Welche Atome interagieren mit Atomen aus der Umgebung
- Eine Lösung:
  - Plaziere zufällig Atome um das geg. Molekül herum
  - Berechne das Voronoi-Diagramm alle Punkte
  - Interface = Voronoi-Facetten zwischen Molekül und Umgebungsatomen



Verbesserungen

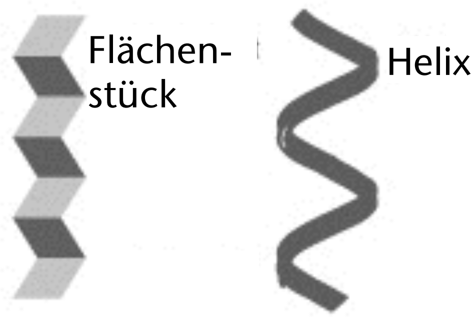
- Verwende Power-Diagramm oder Voronoi-Diagramm mit additiven Gewichten
  - Gewicht = Atomradius
- Berechne "Tiefe" pro Atom:
  - Atome mit einer Voronoi-Facette nach außen = Tiefe 1
  - Traversiere Delaunay-Graph breadth-first von außen nach innen
  - Je tiefer ein Atom, desto geringer sein Beitrag zu Wechselwirkungen



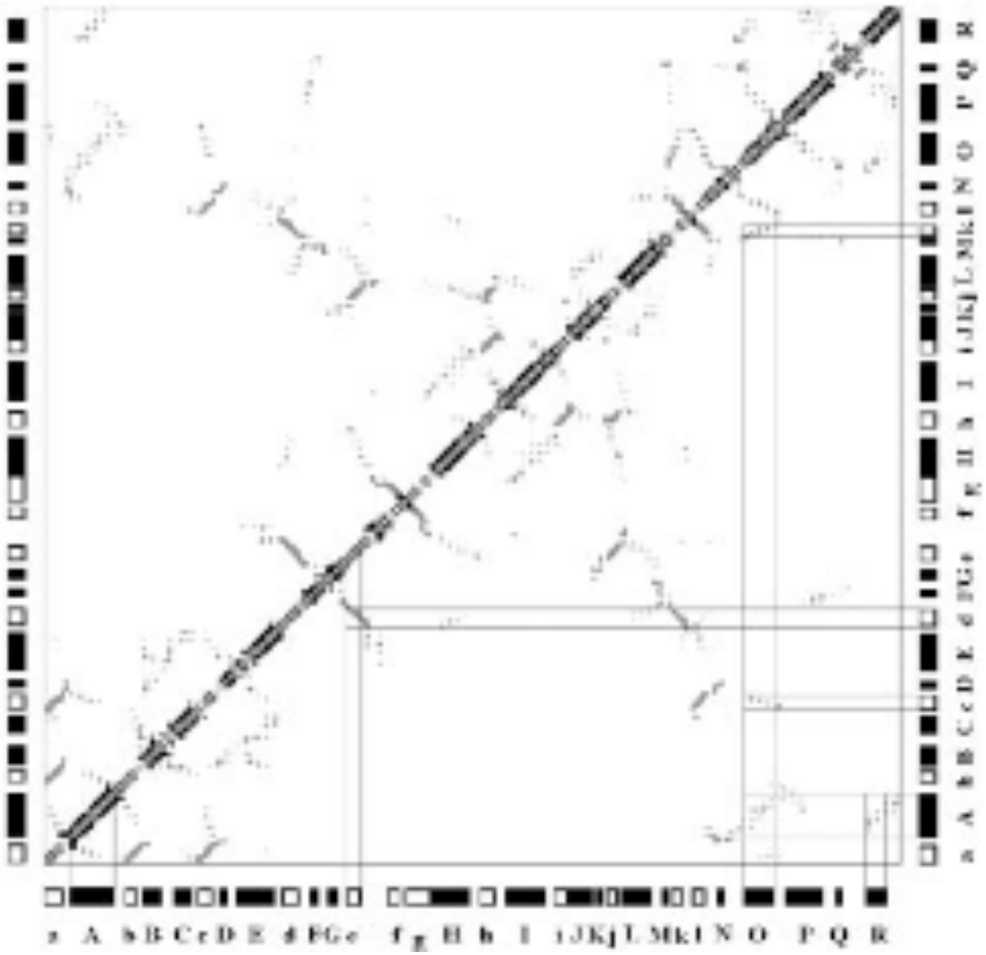


Secondary Structure of Proteins

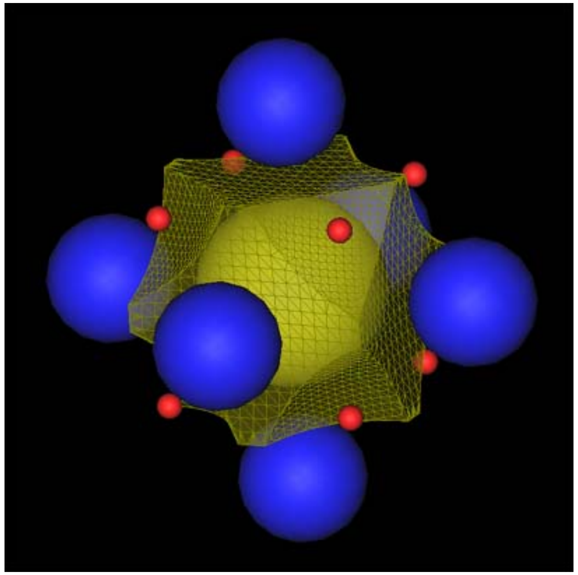
- Lange Proteine falten sich zu Helices, Knäueln, und Flächenstücken
- Ergibt Wechselwirkungen zwischen Atomen (Bindungen), die nicht in der chemischen Formel zu sehen sind
- Frage: gegeben die Positionen der Atome, wie sieht die sekundäre Struktur aus?
  - Welche Atome sind "benachbart", welche nicht
  - Wie stark sind sie benachbart?
- Lösung: Voronoi-Diagramm
  - Benachbart = gemeinsame Voronoi-Facette
  - Stärke der Nachbarschaft = Größe der Facette

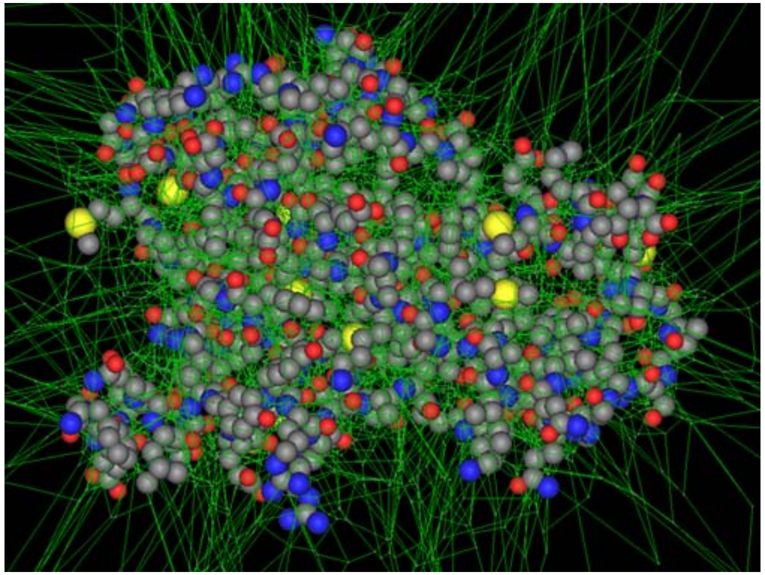


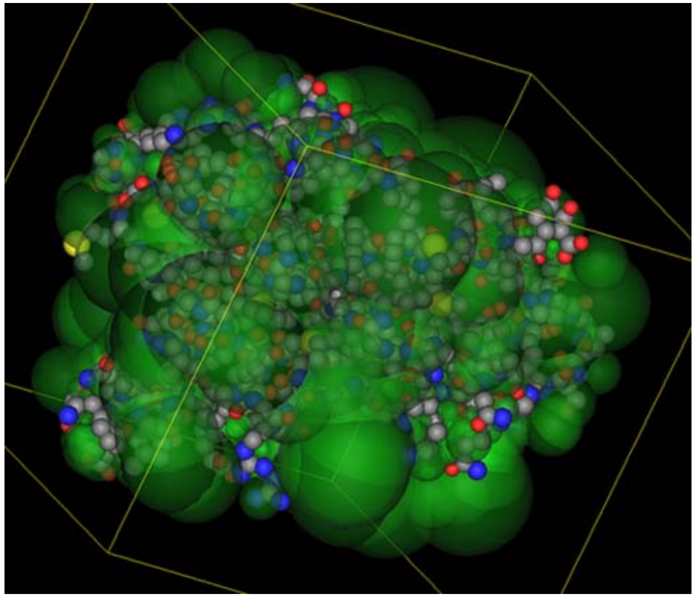
- Resultat: Adjazenz-Matrix (grau/schwarz = schwach/stark benachbart)



Appolonius-Diagramme in 3D







Z.B. um die Leerstellen in einem Molekül zu bestimmen

Visibility Sorting Using Voronoi Diagrams

- Erinnerung: BSPs für Visibility-Sortierung
- Methode:
  - Definiere eine Visibility-Relation auf Voronoi-Regionen

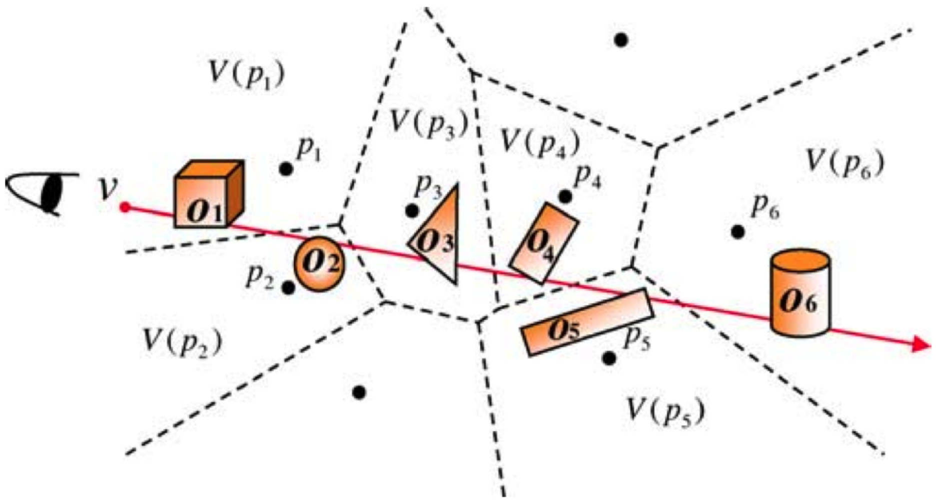
    $$R_1 \prec_v R_2$$

    jeder Punkt der Voronoi-Zelle R_2 wird durch einen Punkt der Zelle R_1 bzgl. des Viewpoints v verdeckt
  - Nun gilt:

    $$R_1 \prec_v R_2 \Leftrightarrow \forall p_1 \in R_1 \forall p_2 \in R_2 : \|v - p_1\| < \|v - p_2\|$$

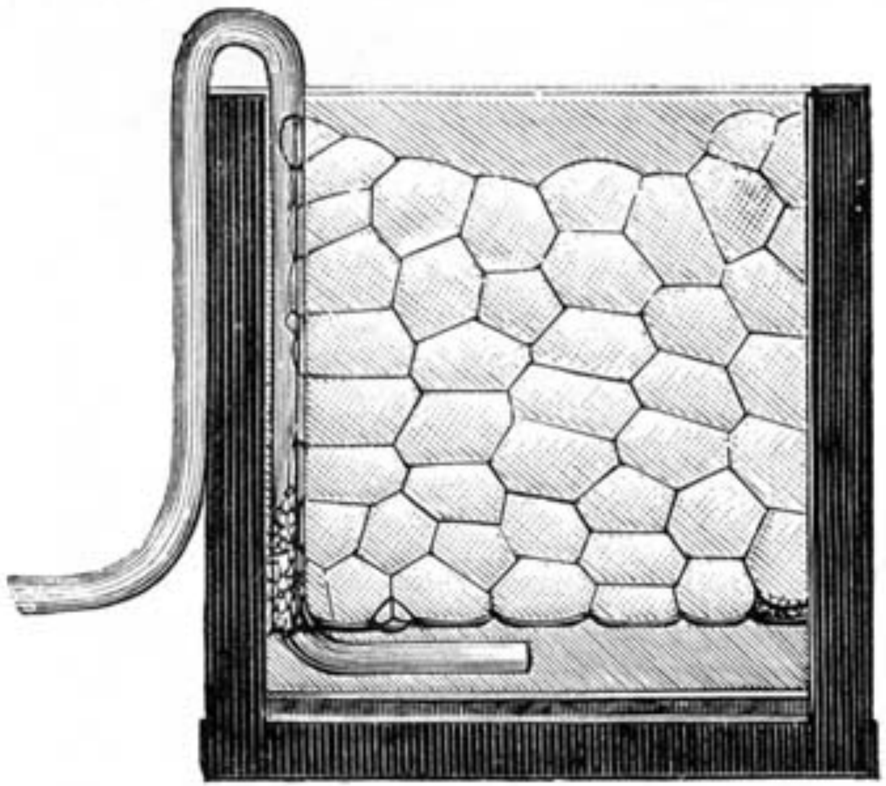
    - Bew.: klar weil R_1 und R_2 komplett auf verschiedenen Seiten des Bisektors zwischen R_1 und R_2 liegen.

- Idee:
  - Zunächst alle Pgone in Voronoi-Zellen clustern
  - Zur Laufzeit nur noch die Voronoi-Sites sortieren (inkrementell)

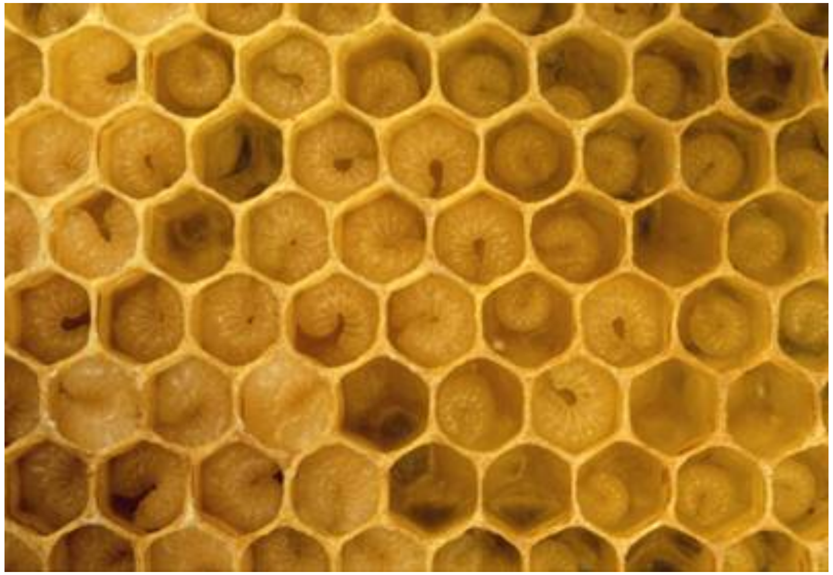


- Ansatz zum Voronoi-Clustering:
  - Initialisierung: eine Zelle pro Polygon mit Schwerpunkt als Site
  - Die kleinste Zelle löschen:
    - Voronoi-Diagramm lokal neu berechnen
    - Polygone der kleinsten Nachbarzelle zuordnen
  - Abbruch falls keine Zelle mehr aufgelöst werden kann, ohne daß eine zyklische Visibility-Ordnung in einer Zelle entsteht

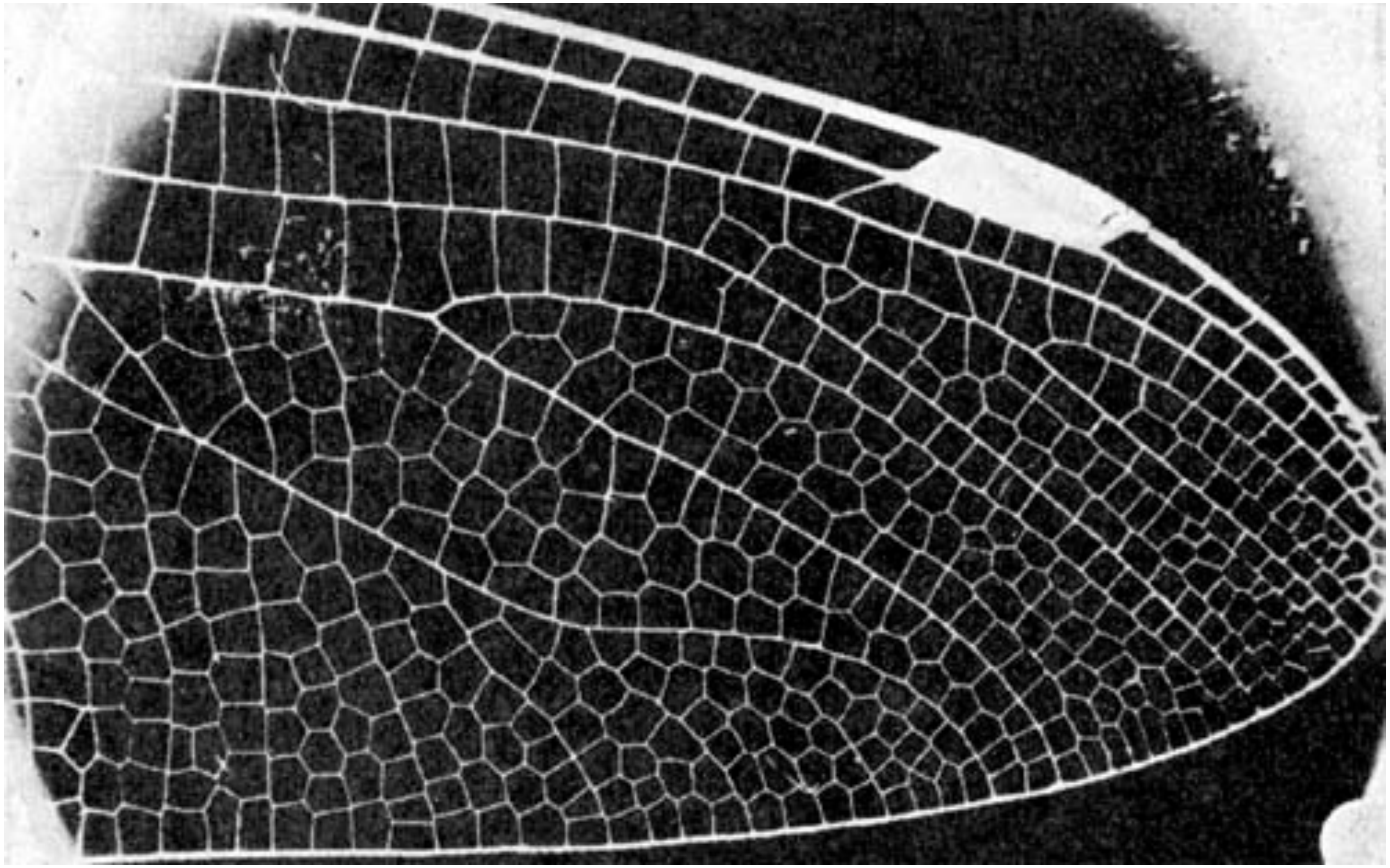
Voronoi-Diagramme in der Natur



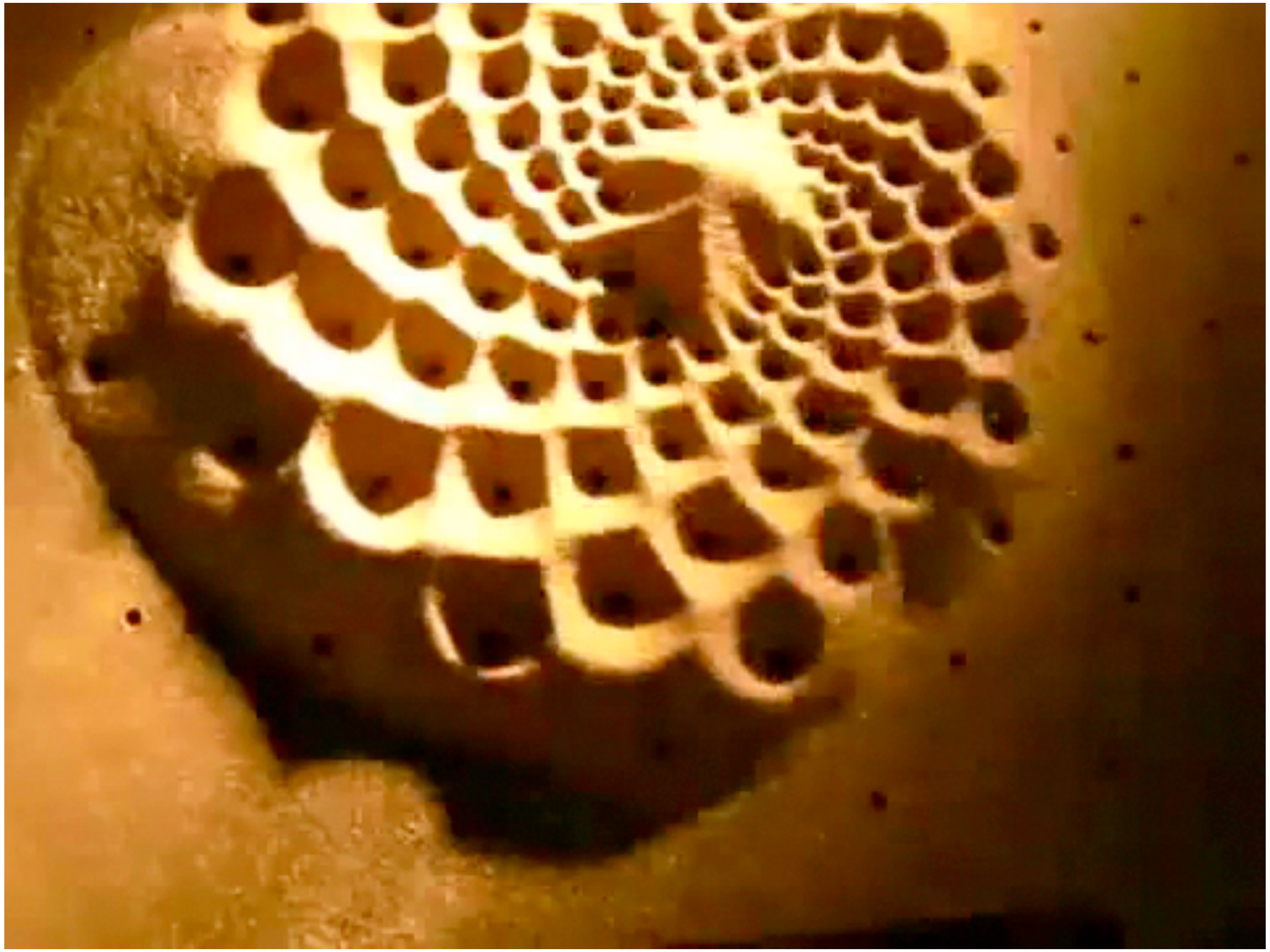
Seifenblasen in einem Glasrahmen



Bienenwaben
(centroidal Voronoi tesselation)



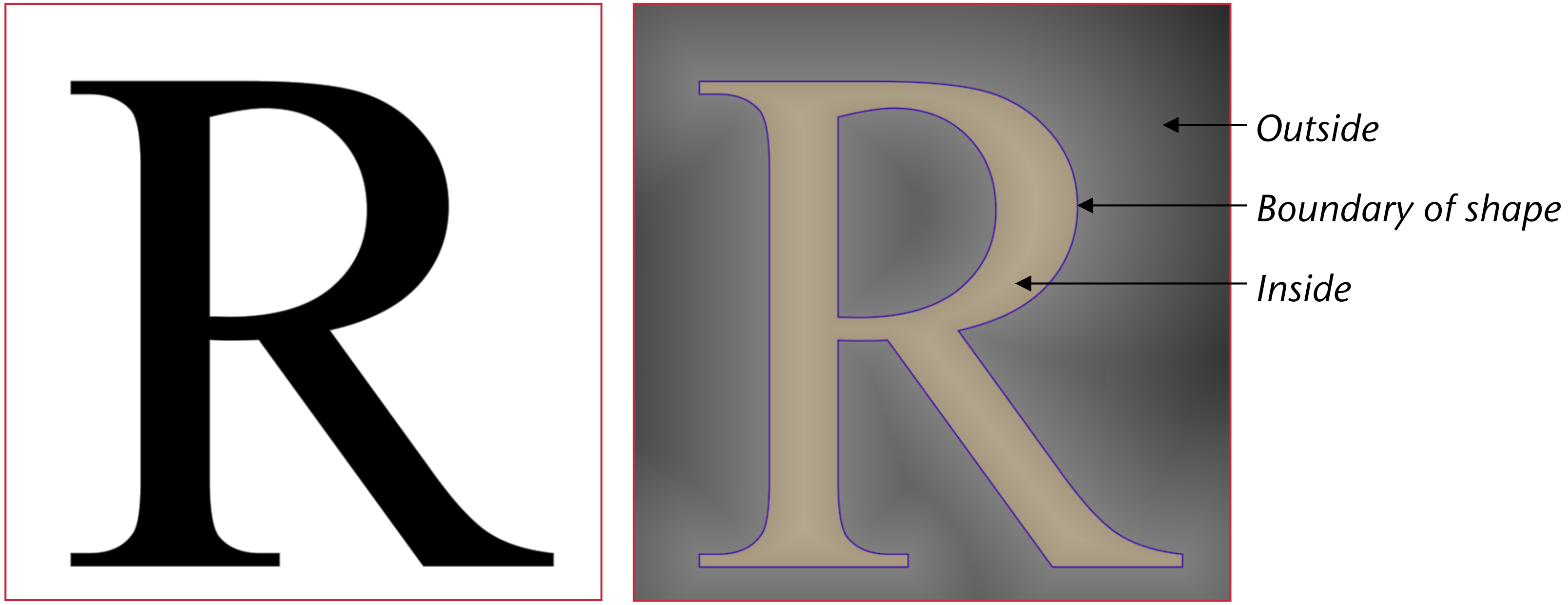
Libellenflügel



Voronoi-Diagramm in der interaktiven Kunst

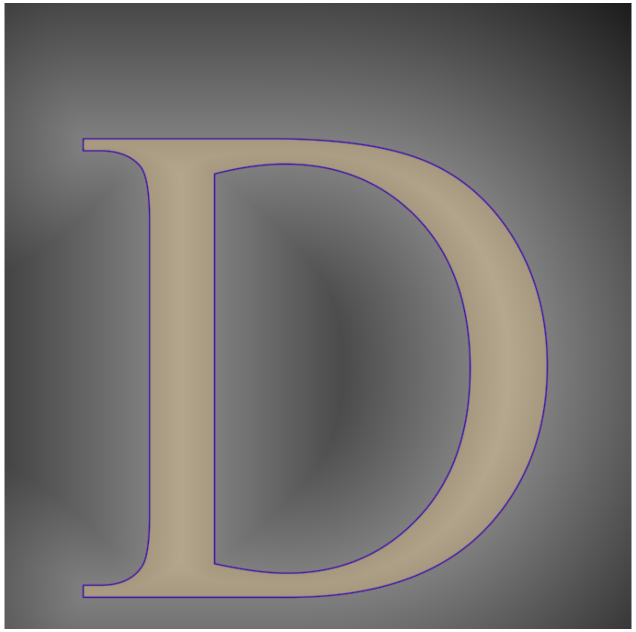
Scott Snibbe, phaeno, Wolfsburg

Distance Fields

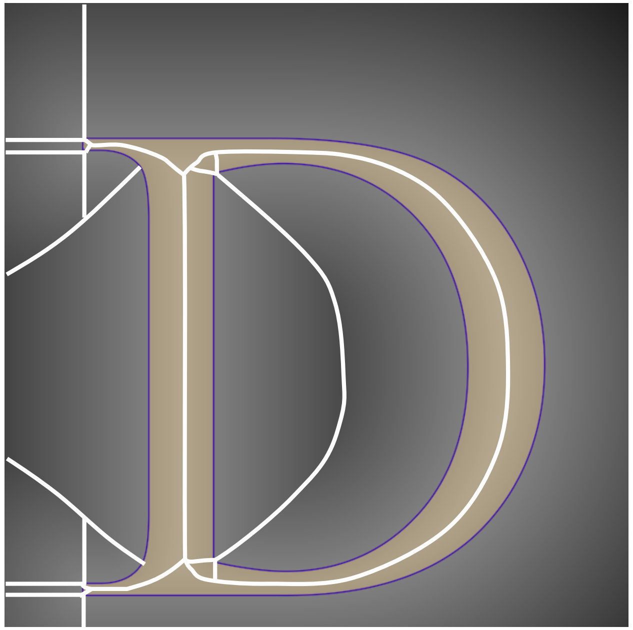


2D Shape *Shape's distance field*

- Distance fields are C^0-continuous everywhere
- Distance fields are C^1-continuous except at boundaries of Voronoi regions

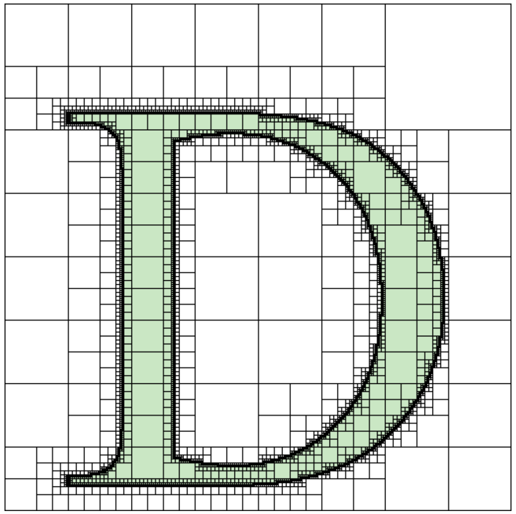


Distance field is C^0 continuous

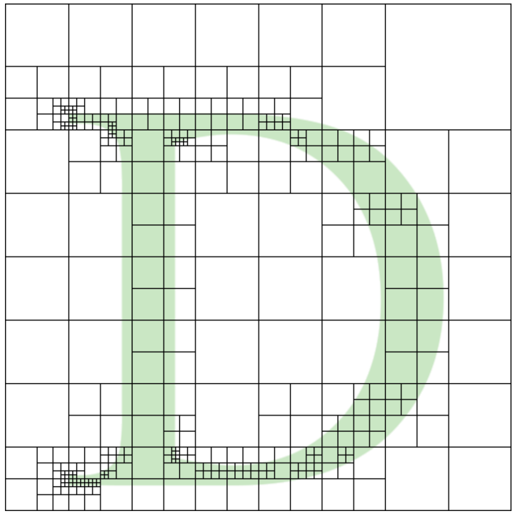


C^1 continuous except at Voronoi boundaries

- Adaptively Sampled Distance Fields (ADFs): sample at low rates where the distance field is smooth; sample at higher rates only where necessary (e.g., near corners)

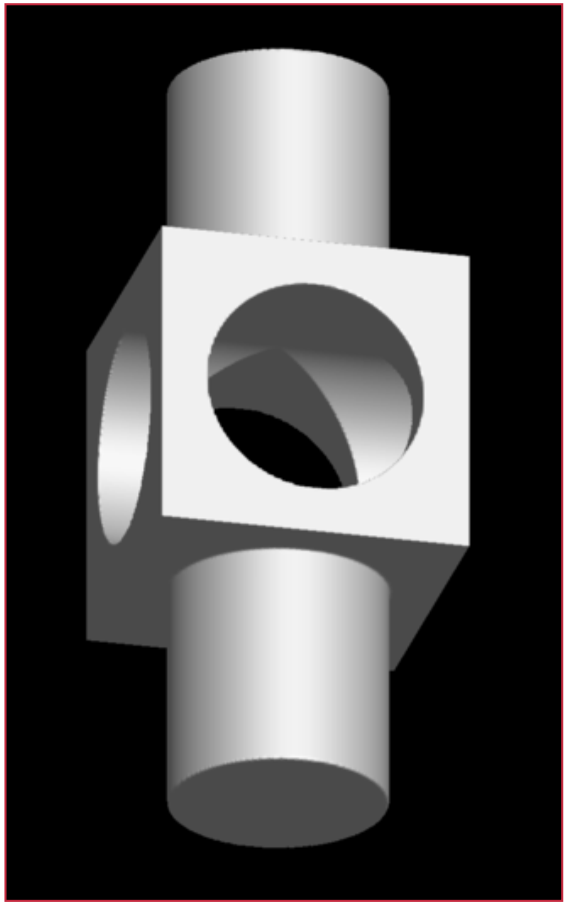


Boundary-limited quadtree

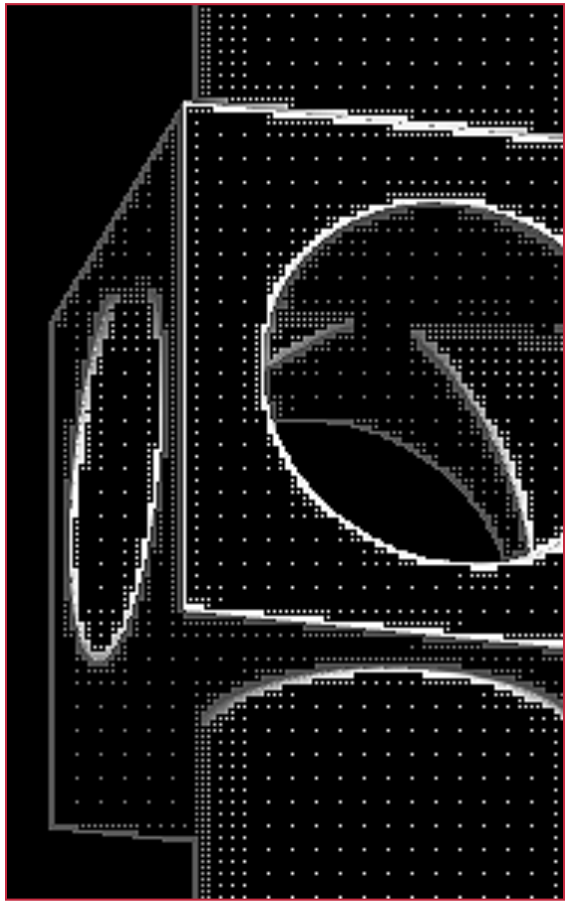


Detail-directed ADF

- Rendering ADF's using adaptive ray-casting:

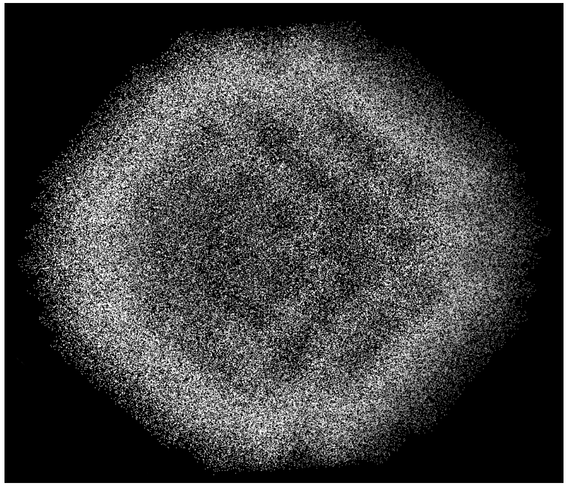


Rendered via adaptively ray casting

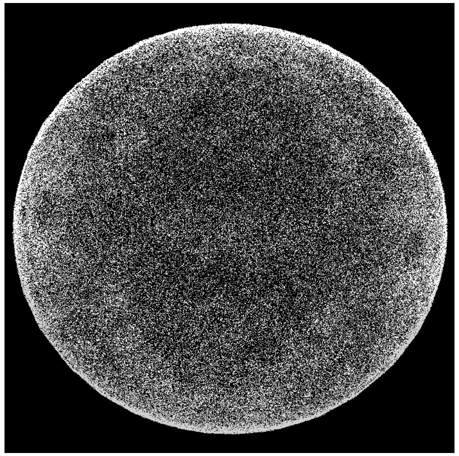


Rays cast to render part of the image on the left

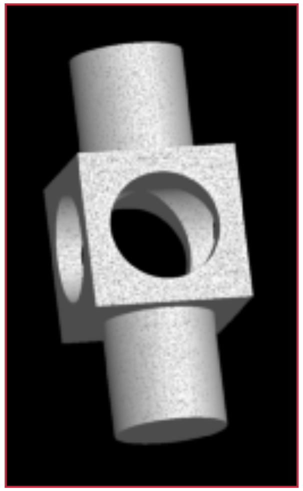
- Point-based rendering of ADF's:
  - Seed each boundary leaf cell with randomly placed points, number of points proportional to cell size
  - Relax the points onto the ADF surface using the distance field and gradient
  - Optionally shade each point using the field's gradient



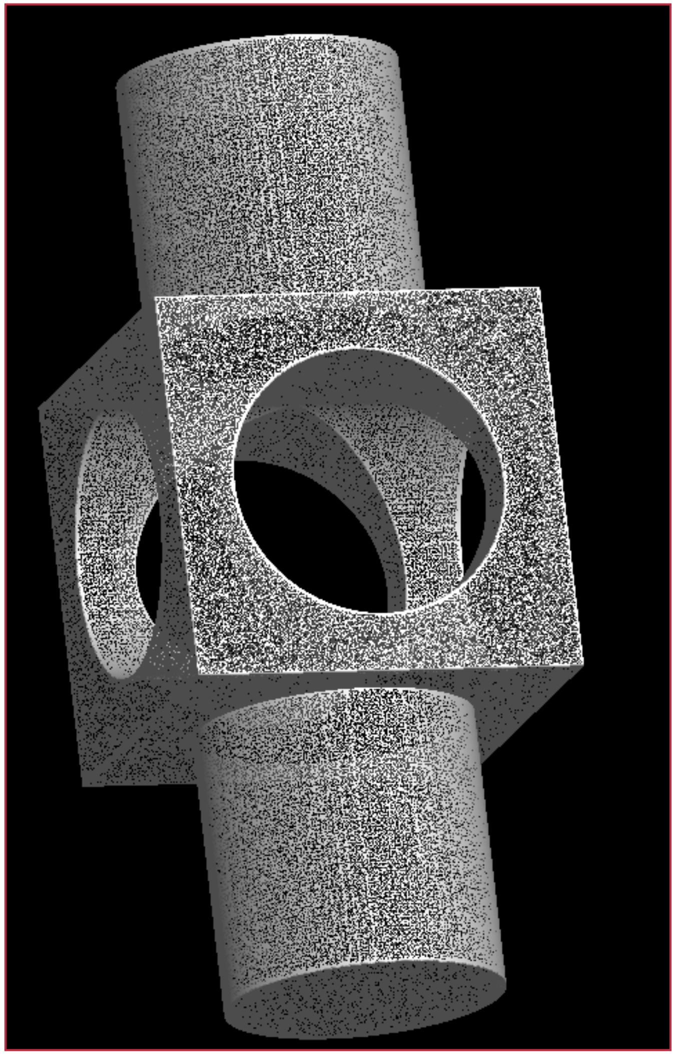
Original points seeded in boundary leaf cells



Points after relaxation onto the surface



An ADF rendered as points at two different scales



Thanks Folks