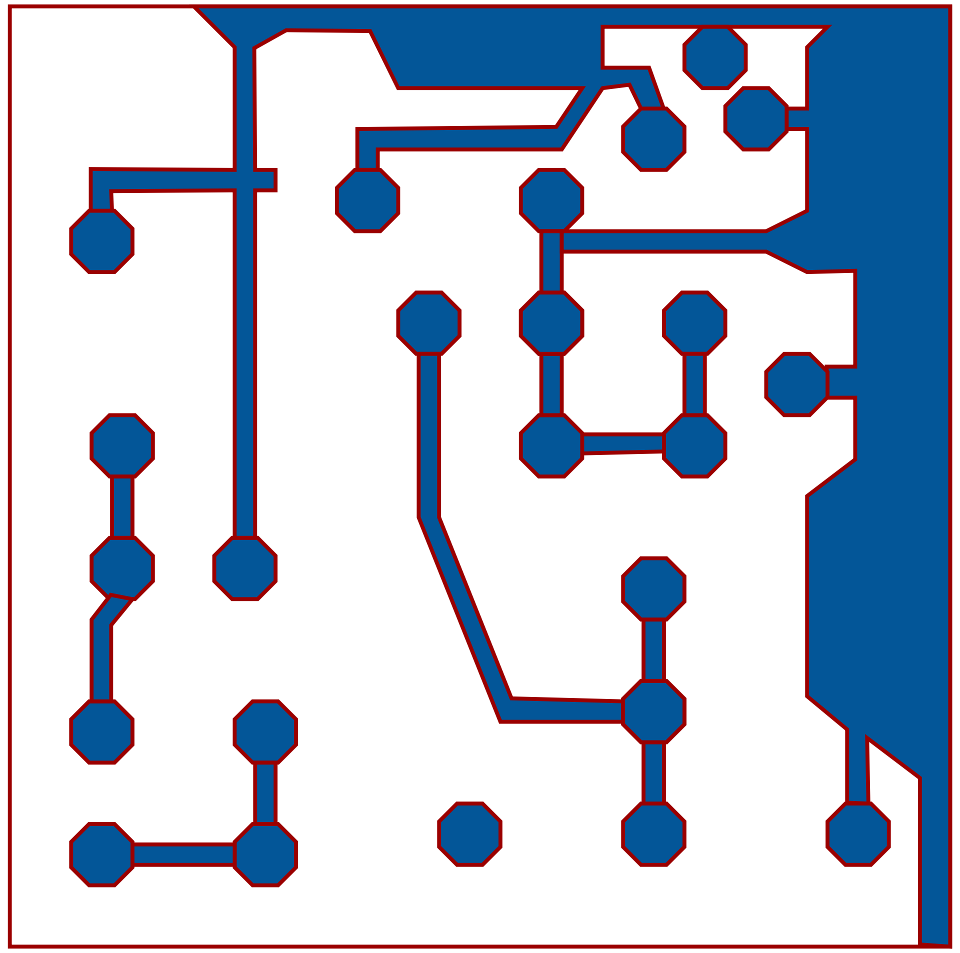
Bremen

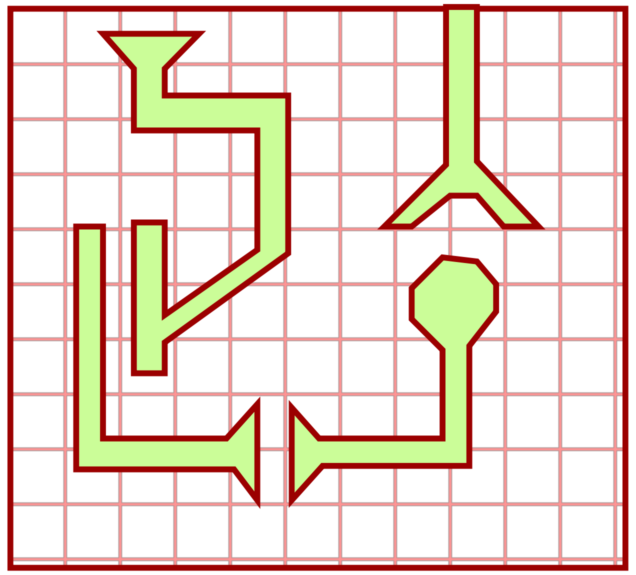
Computational Geometry with Applications

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

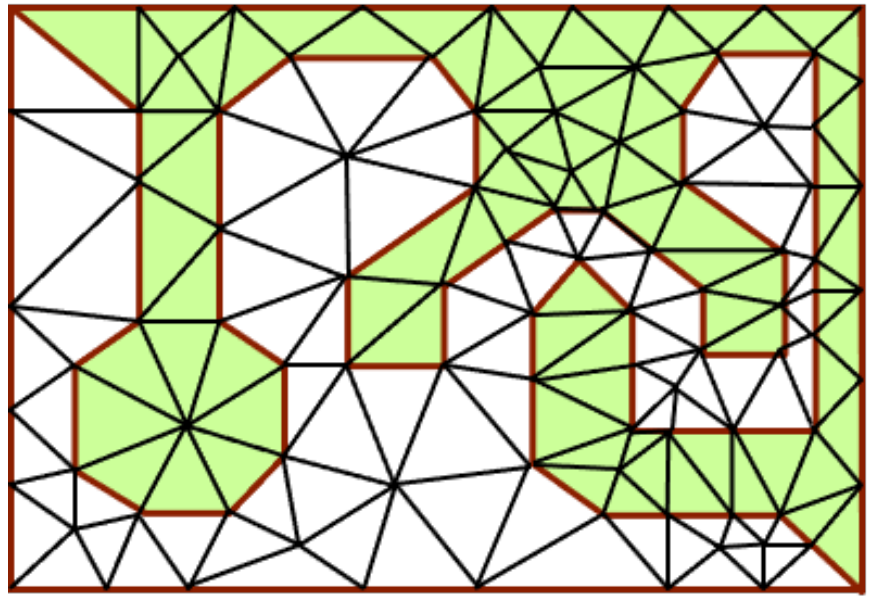
Meshing

- Important preprocessing step for many applications
  - "Domain discretization“ = a complex region (domain) in 2D or 3D is partitioned into a set of much simpler polytopes , e.g., tetrahedra or hexahedra
- Applications:
  - FEM = Finite Element Method (a.k.a. FEA)
  - CFD = Computational Fluid Dynamics
  - Simulation involves solving differential equations for „every“ point inside the domain -> solve only on the nodes

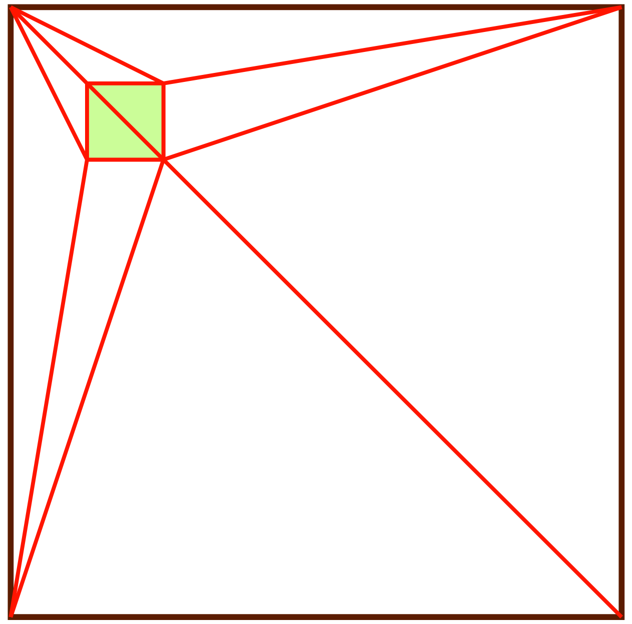




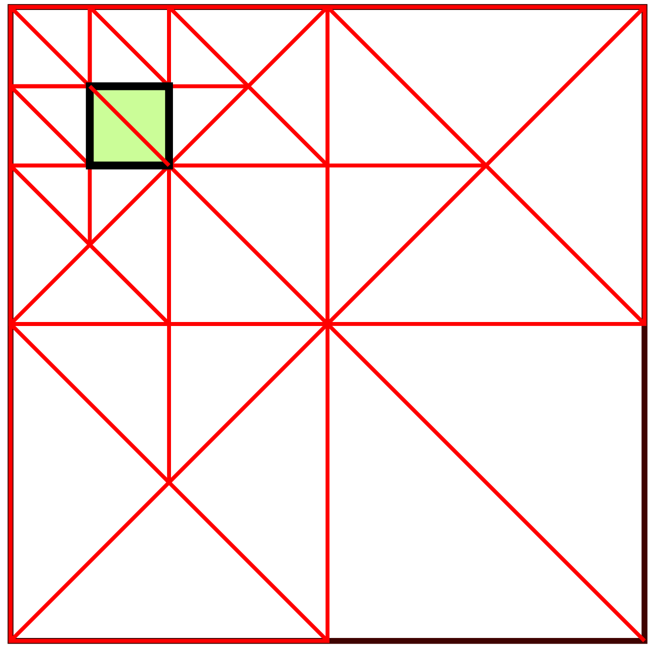
Uniform mesh, i.e. too many mesh elements.



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

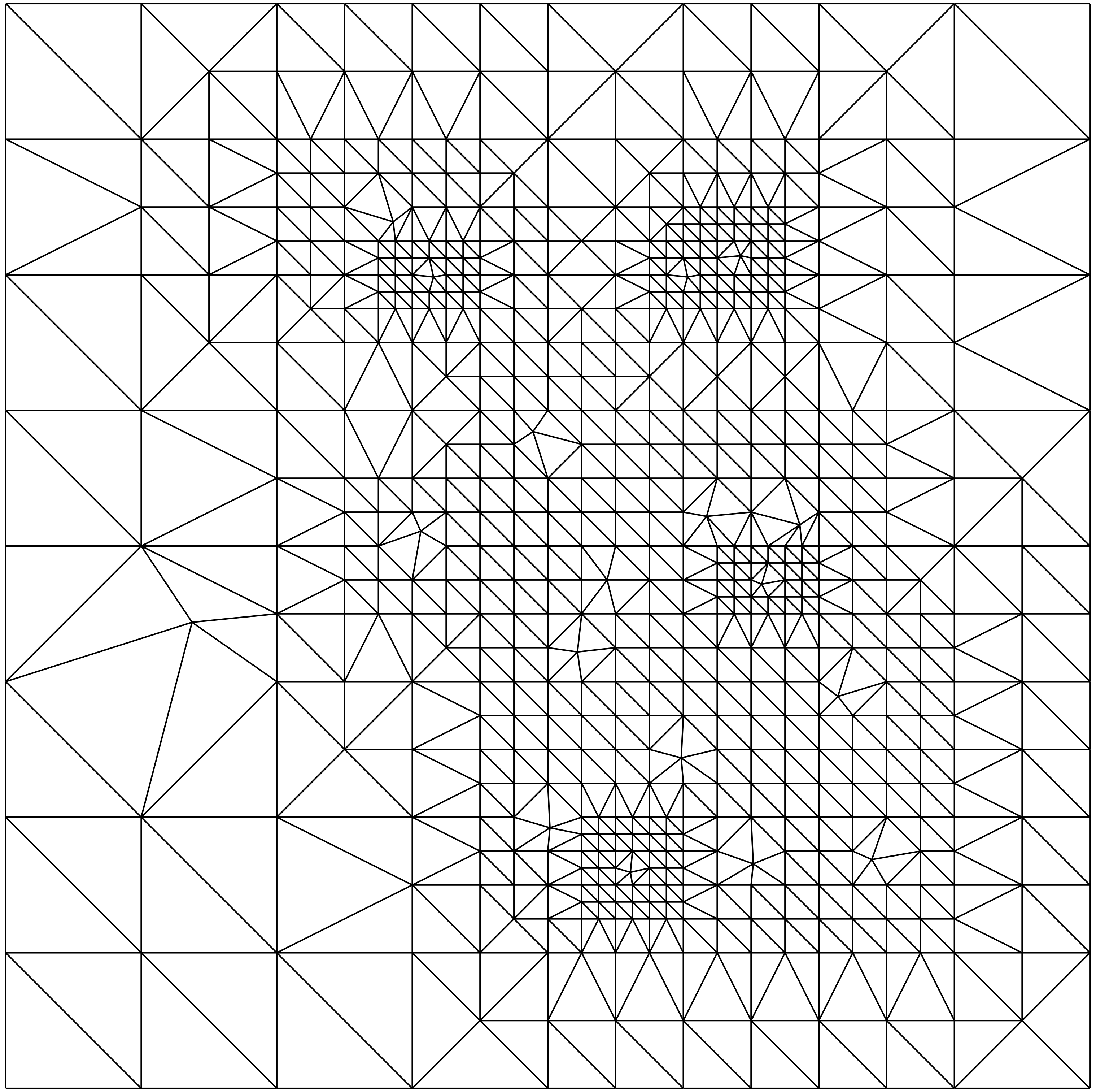


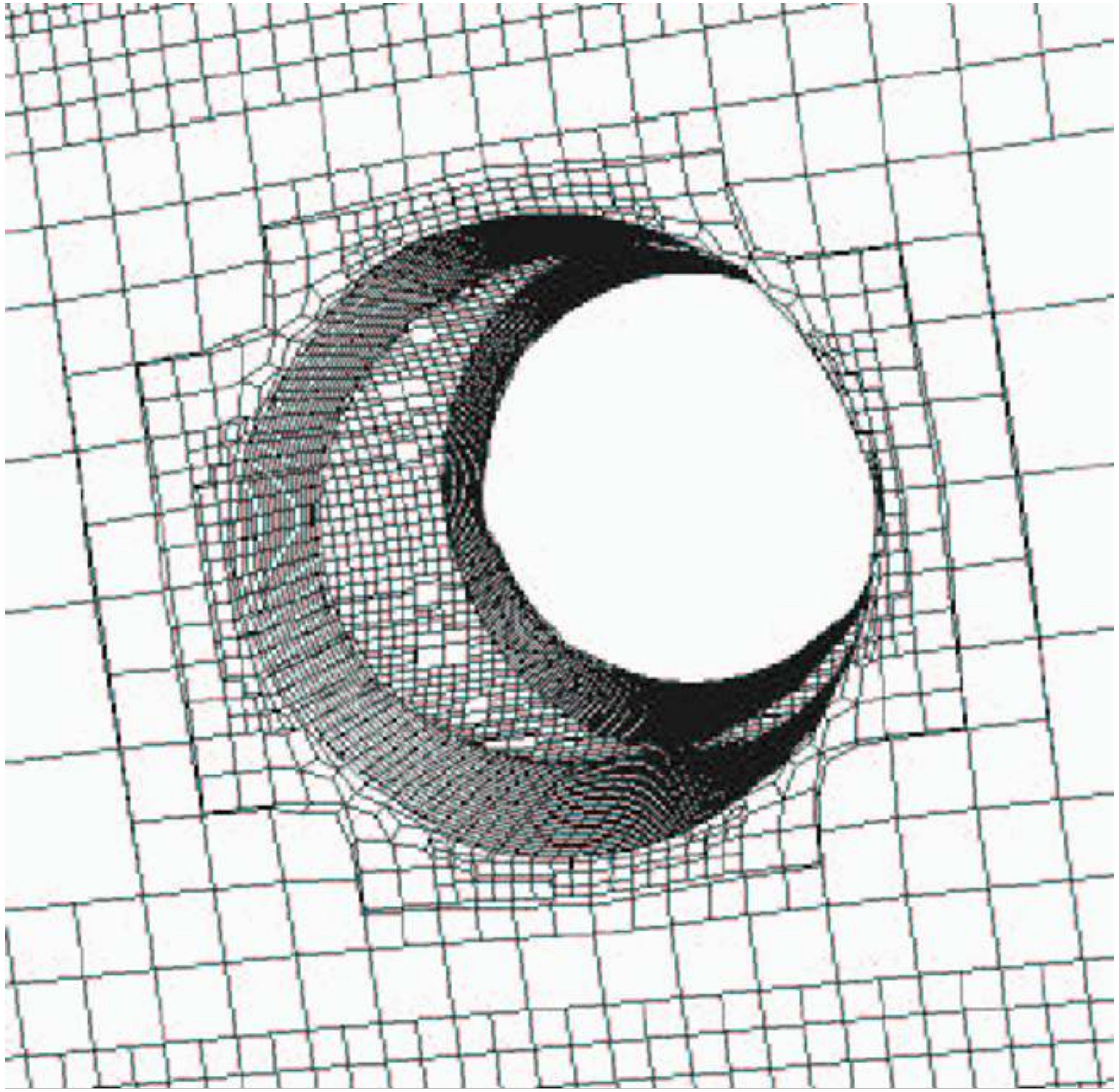
Non-uniform, conforming mesh that respects the input.
But acute triangles.



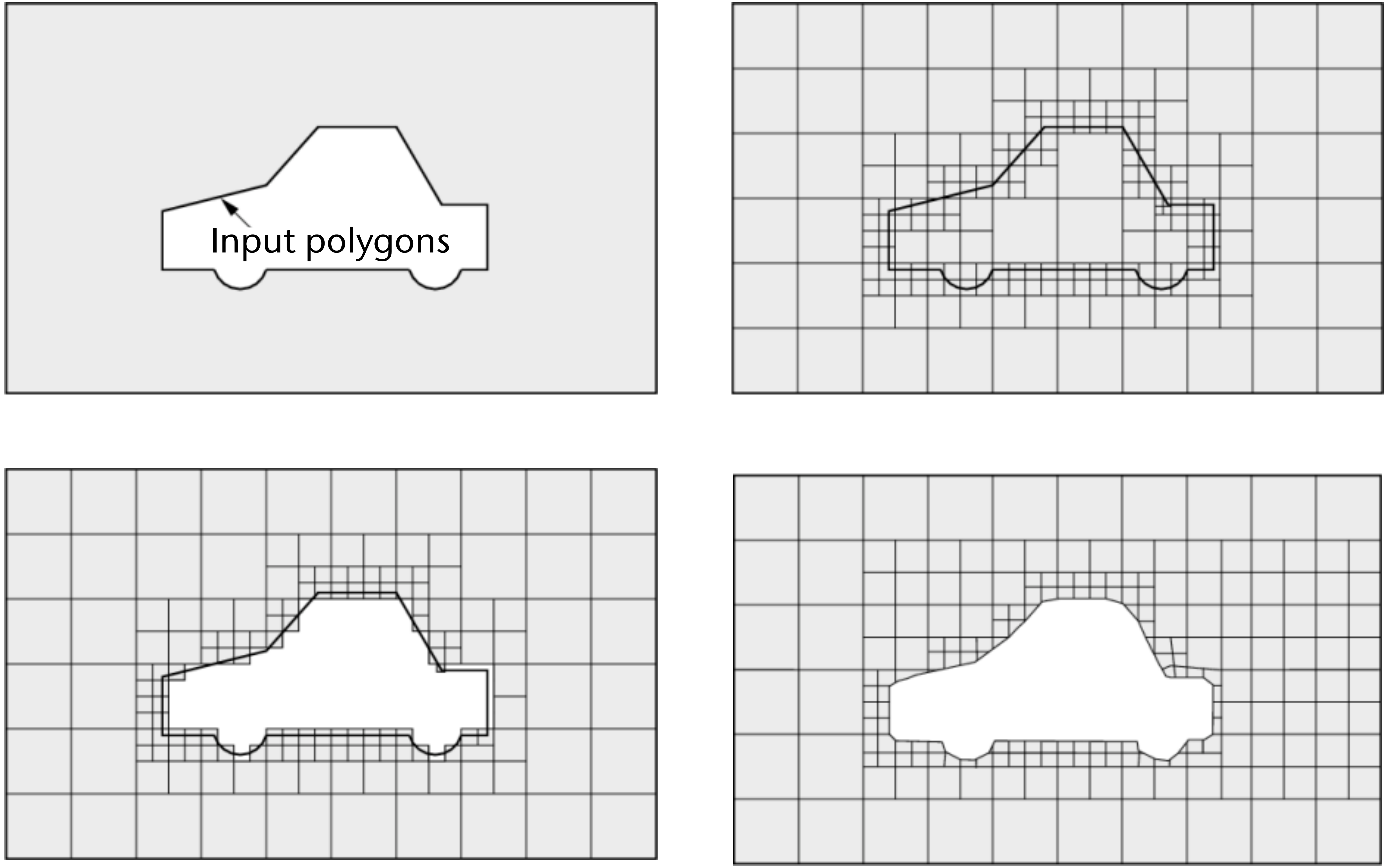
Mesh with all desired properties, based on quadtree.

Example Result of Our Meshing Algorithm

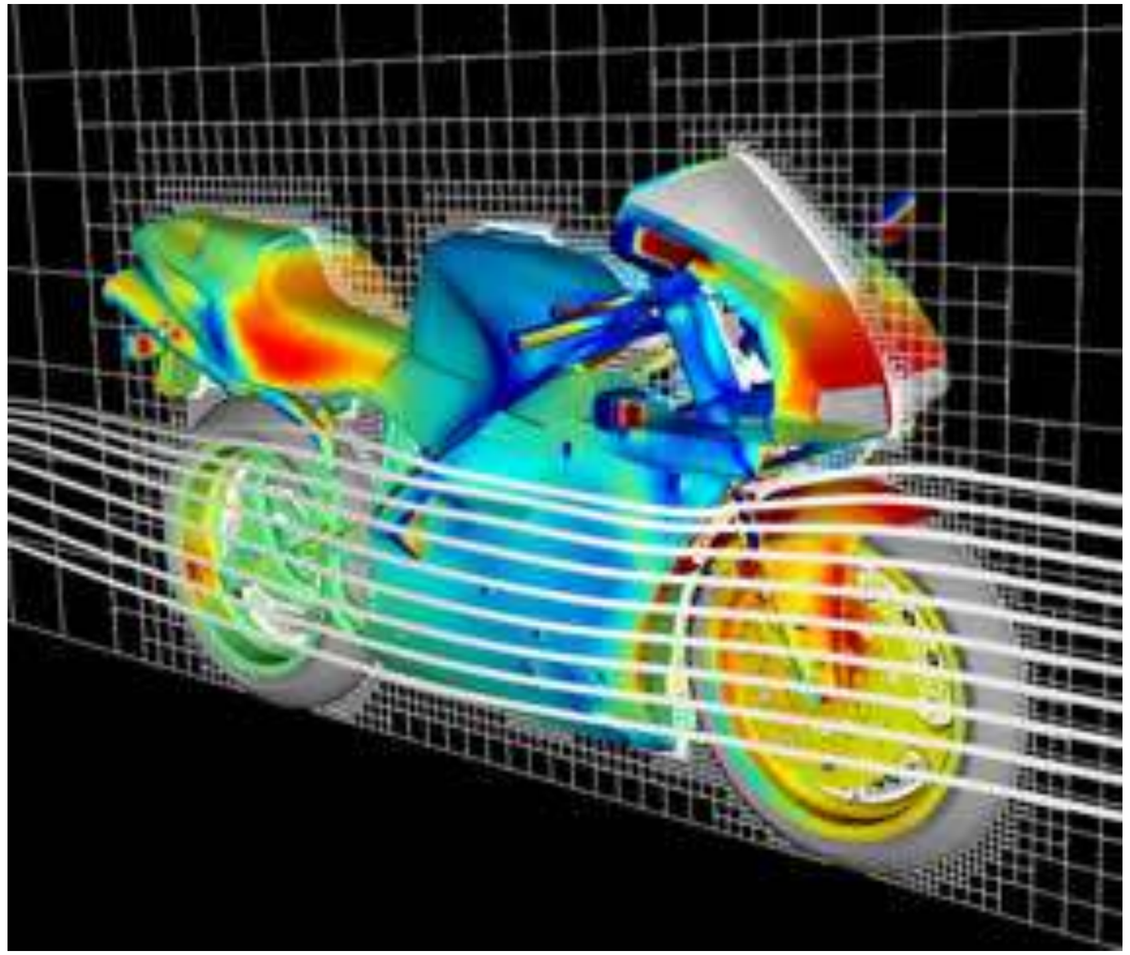


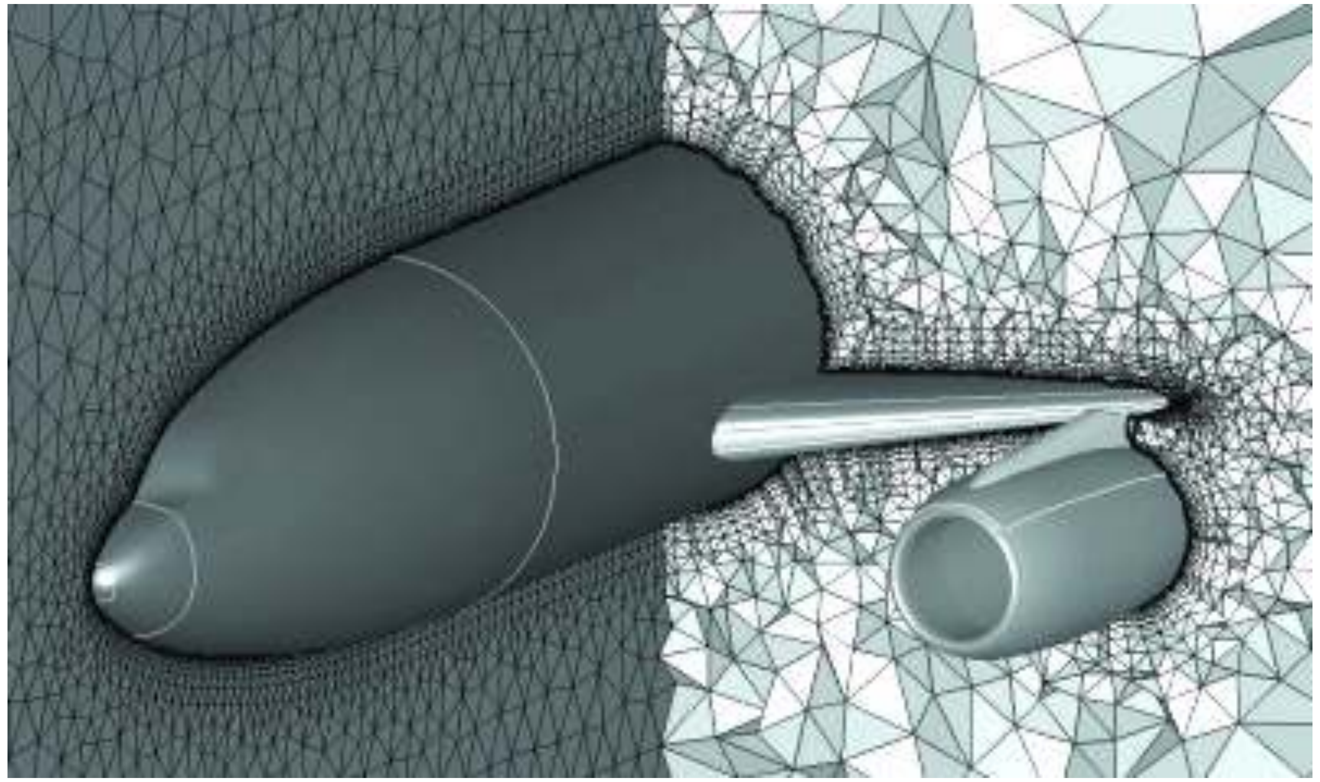


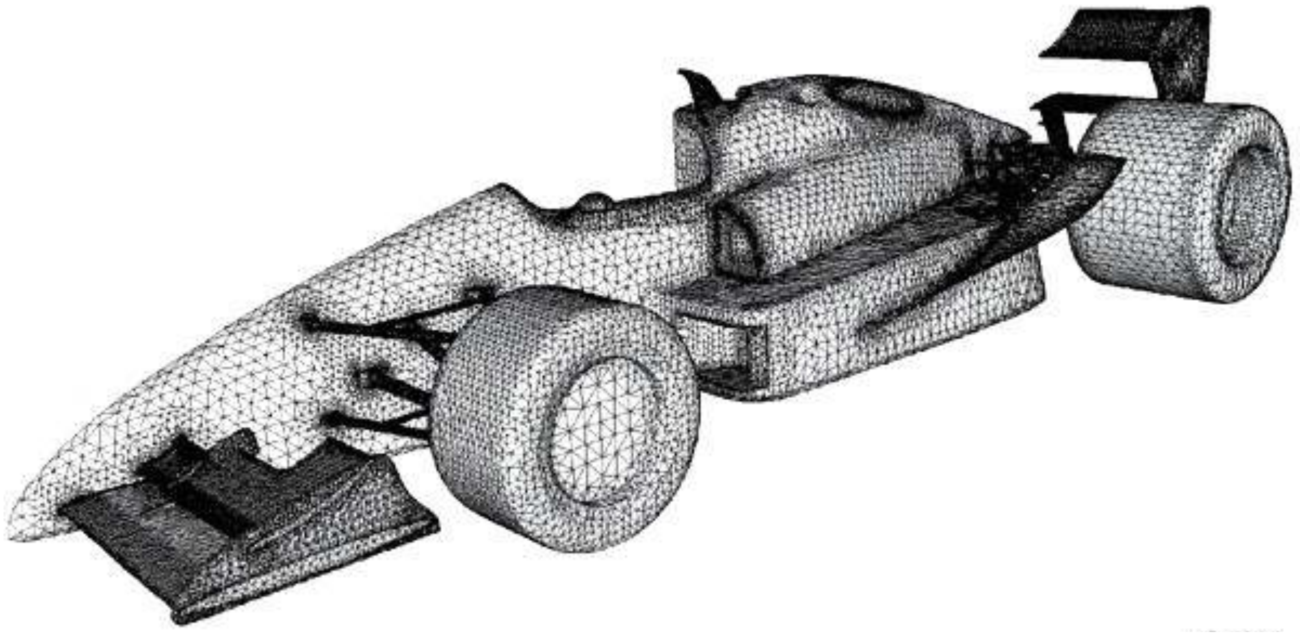
Example "snappyHexMesh"

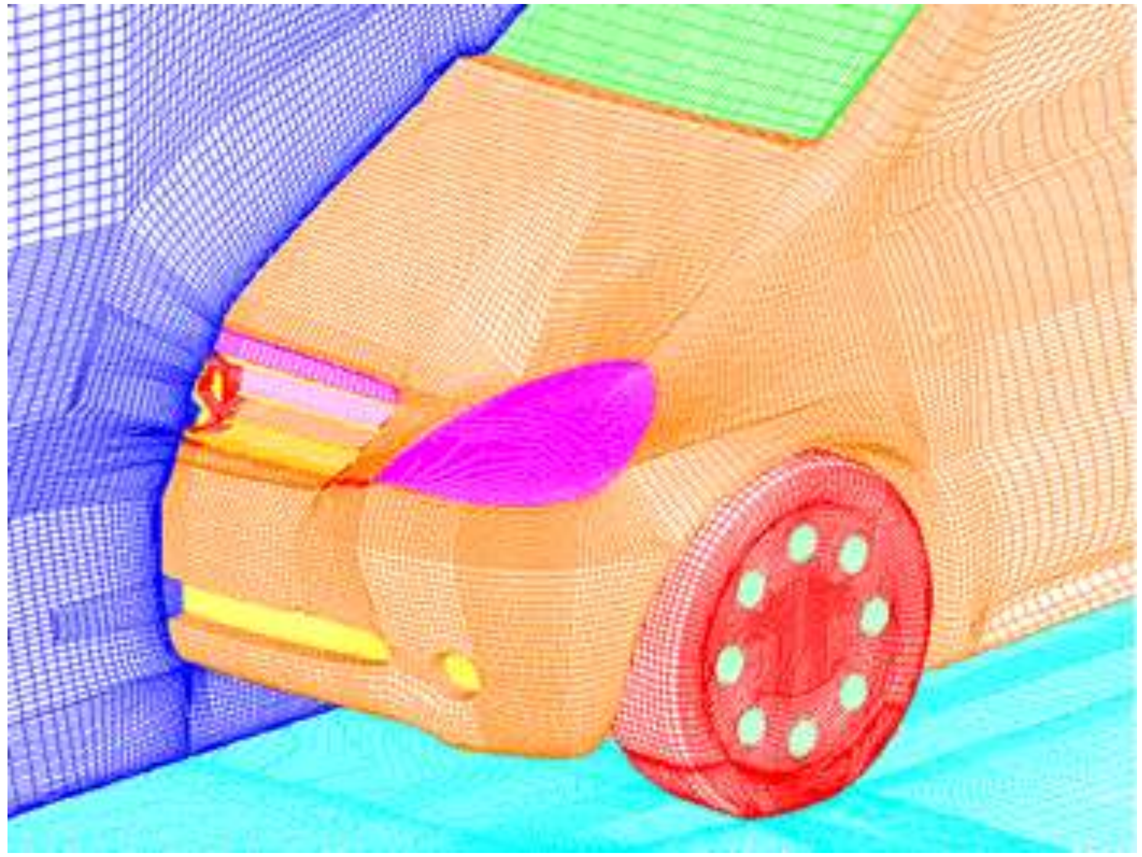


Other Kinds of Volume Meshes

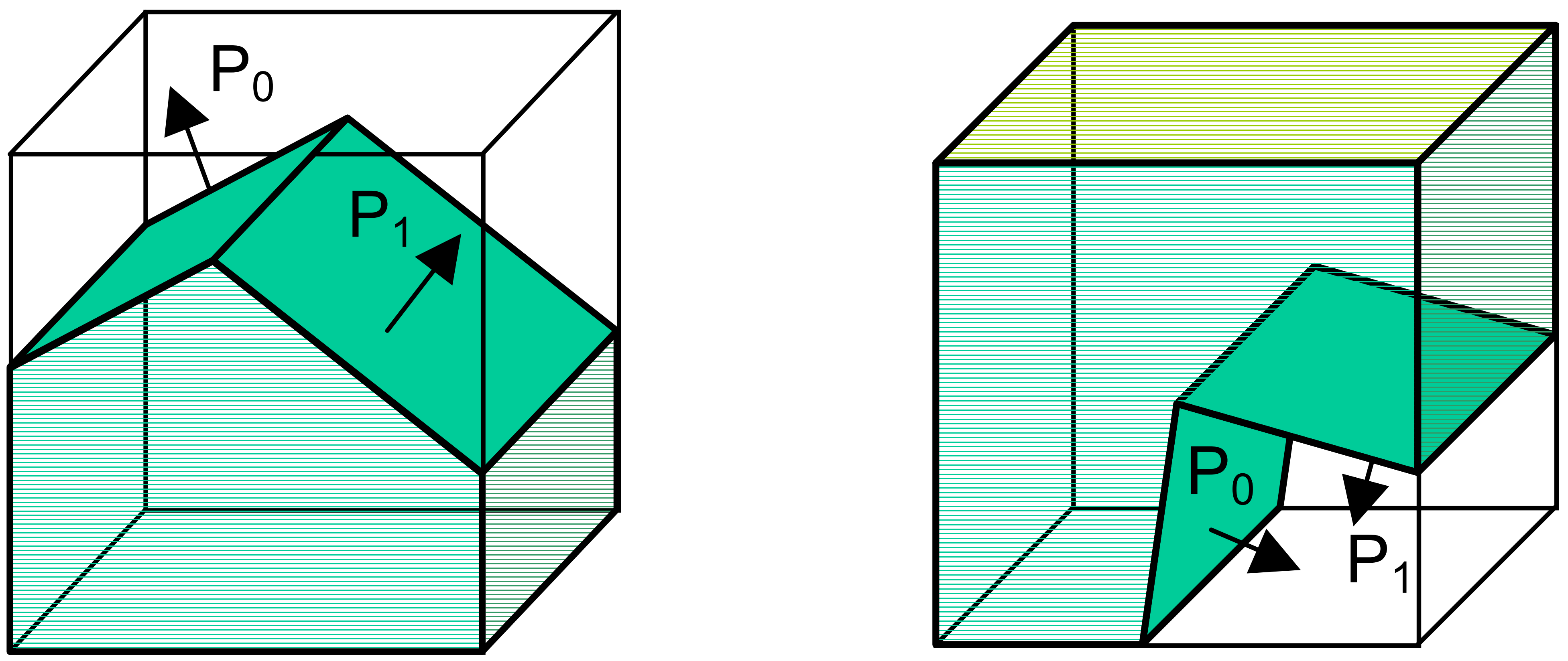








Variant: Exact Octrees (a.k.a. SP-Octrees)



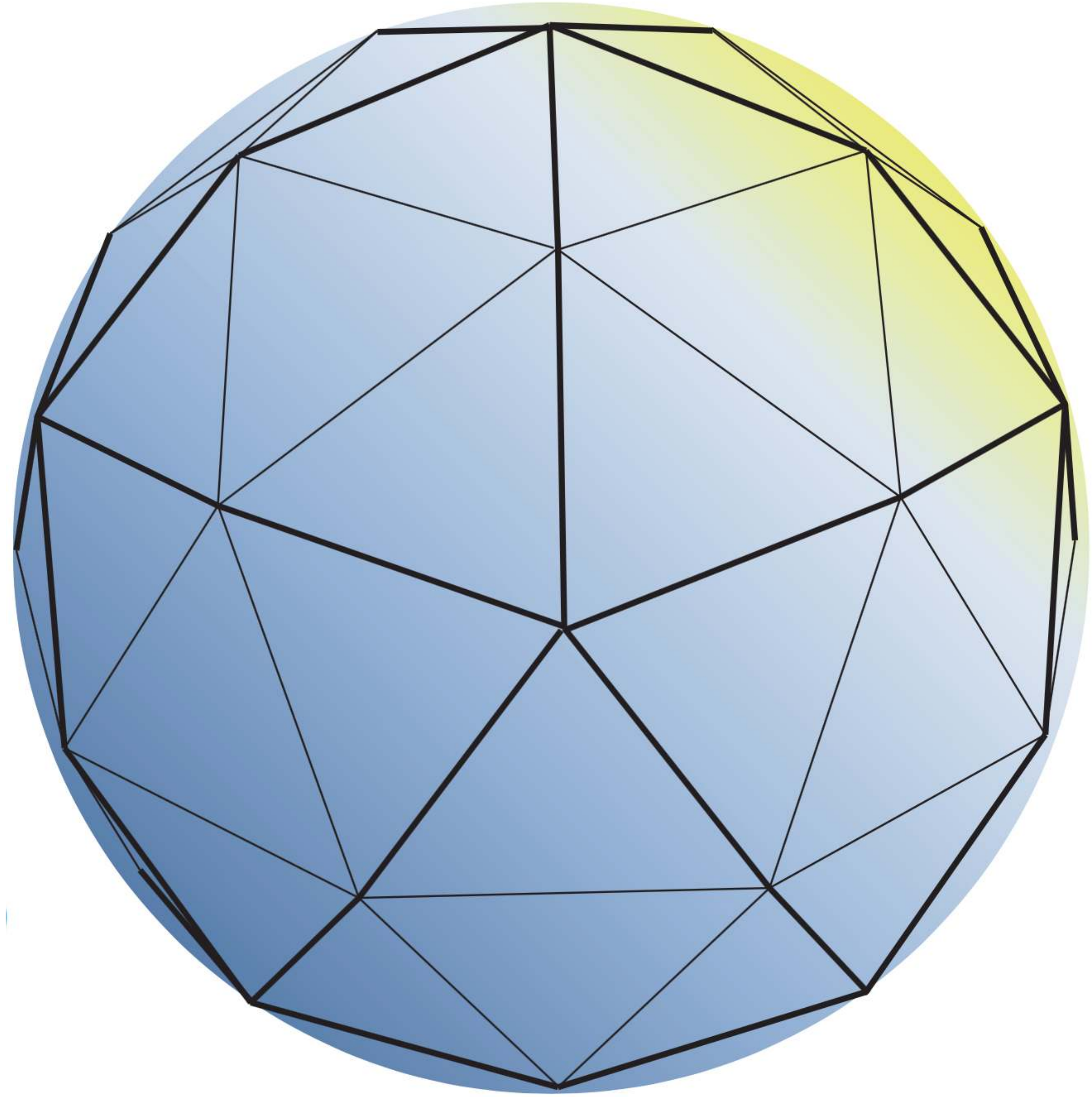
Boundary leaf nodes

Other leaf nodes are either completely outside, or completely inside

Geodesic Dome

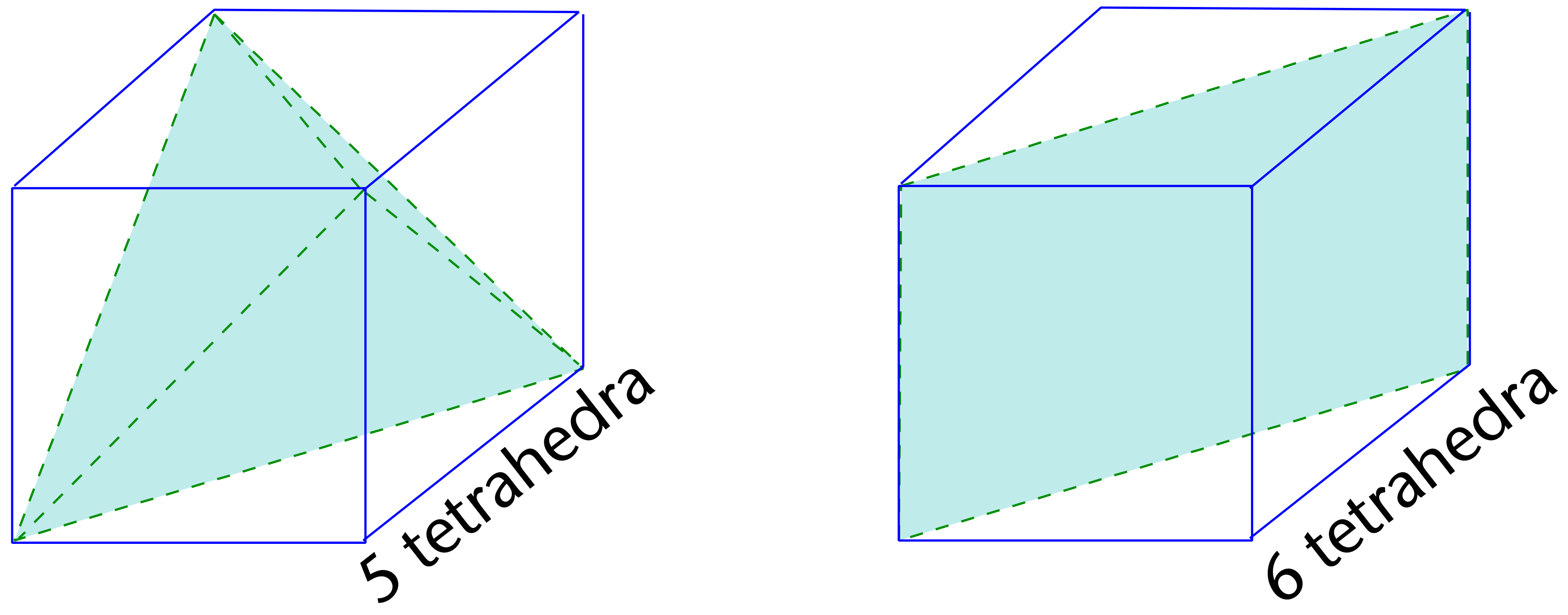
Start with icosahedron; subdivide each triangle by 4 smaller triangles (recursively) $\longrightarrow$ quadtree in each base triangle.

Navigation (finding neighbors of a node) in such an ensemble of quadtrees is a bit more complex

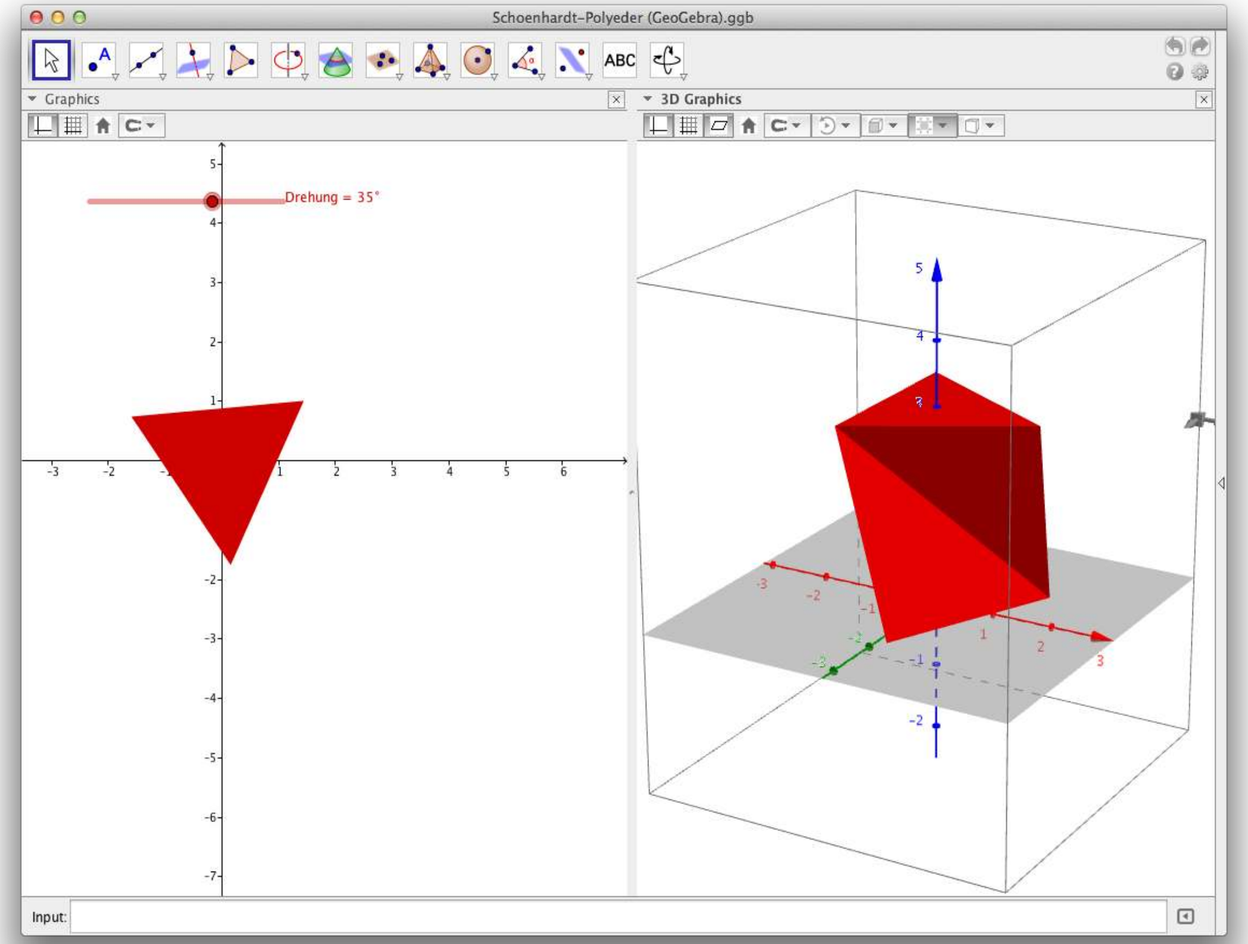


Digression: Triangulations in 3D (="Tetrahedralization")

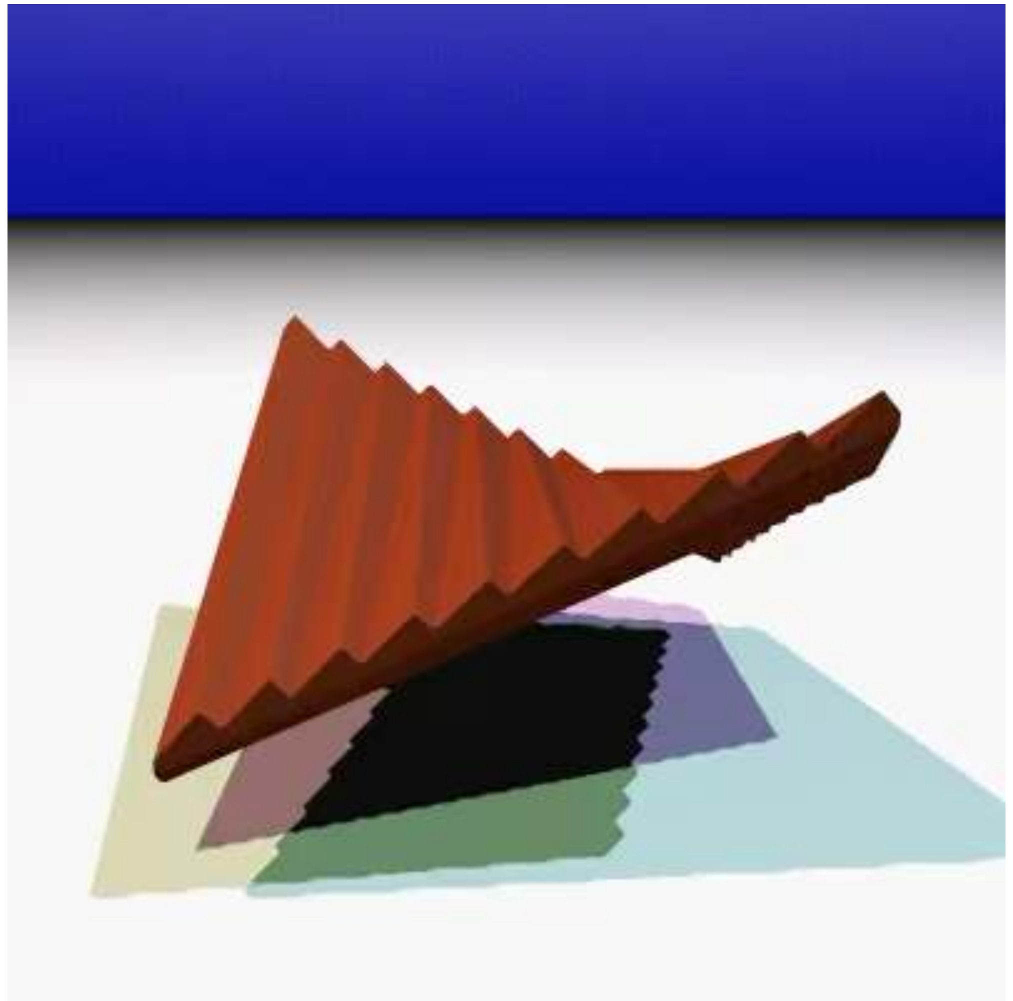
- Is it possible to triangulate a cube *without* additional points (Steiner points)?
- Different triangulations → different number of tetrahedra:



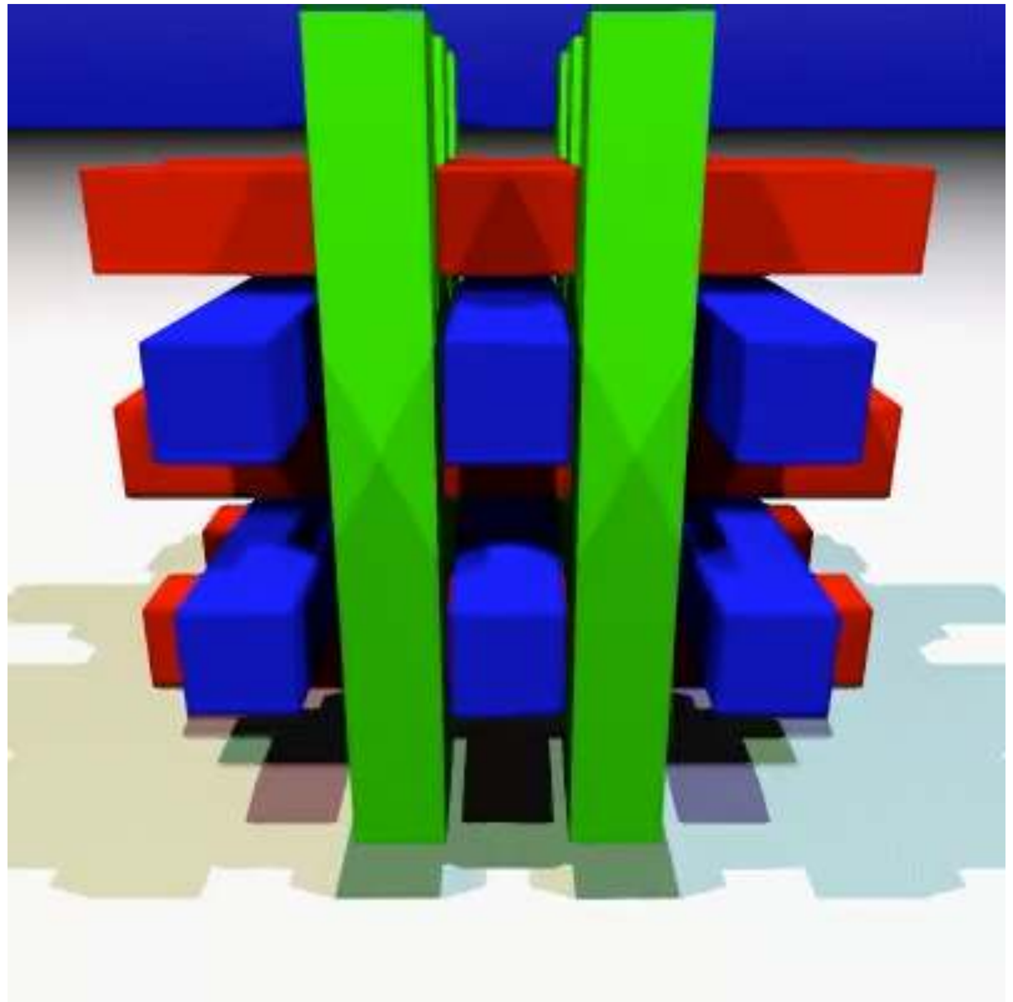
Untriangulable ("Un-Tetrahedralizable") Polyhedra



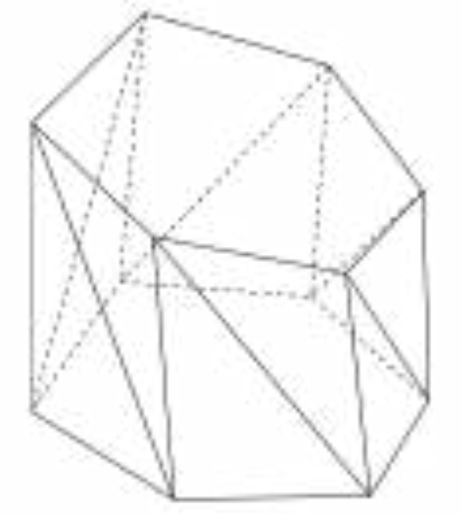
Schönhardt Polyhedron
(1928)



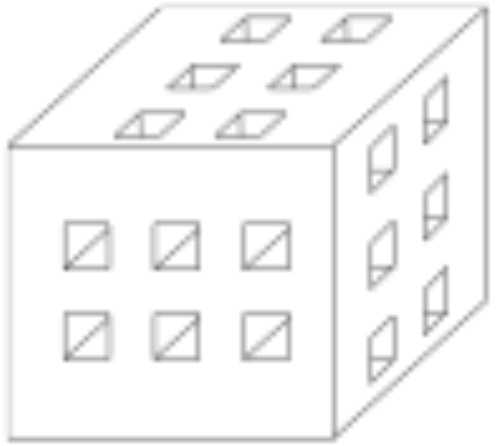
Chazelle Polyhedron
(1984)

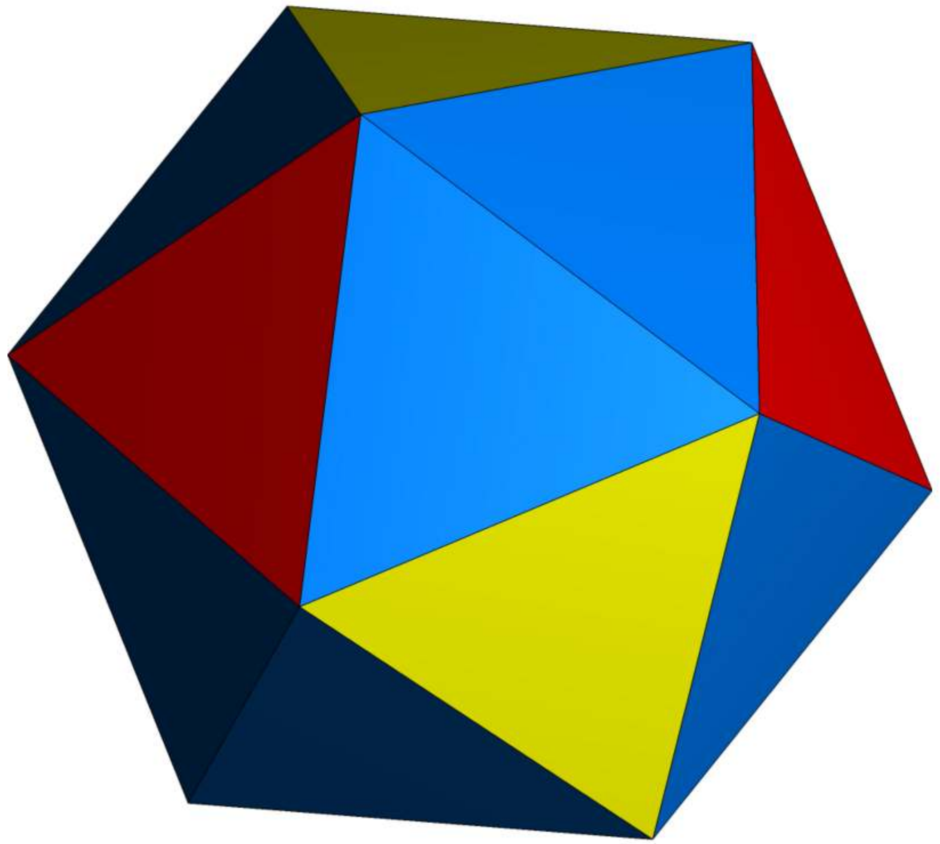


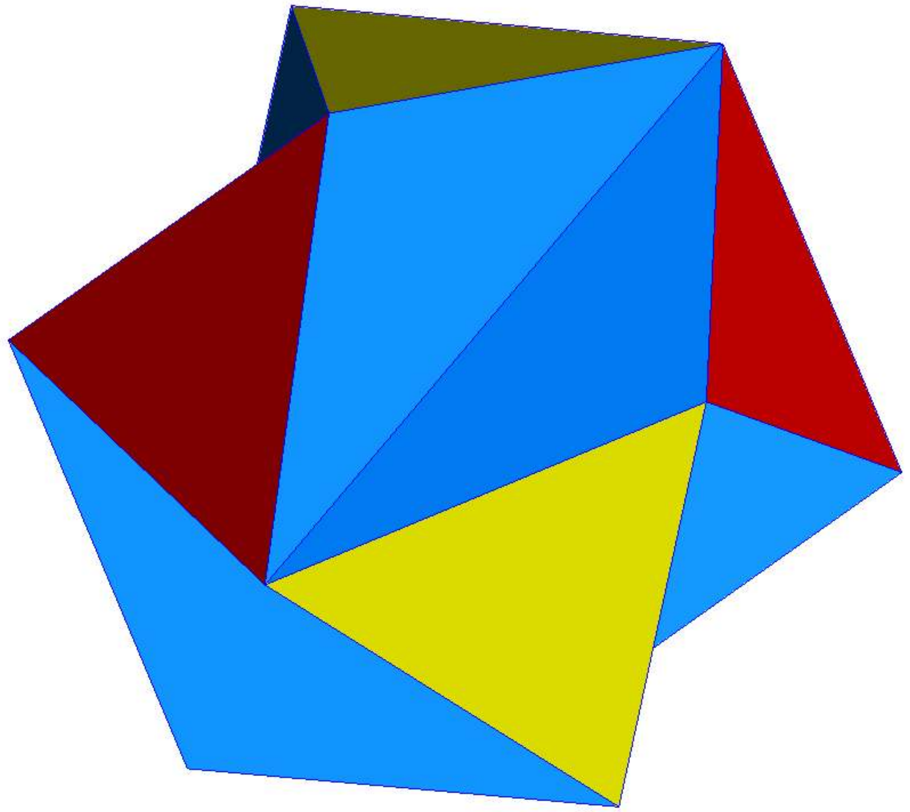
Thurston Polyhedron
(1977)

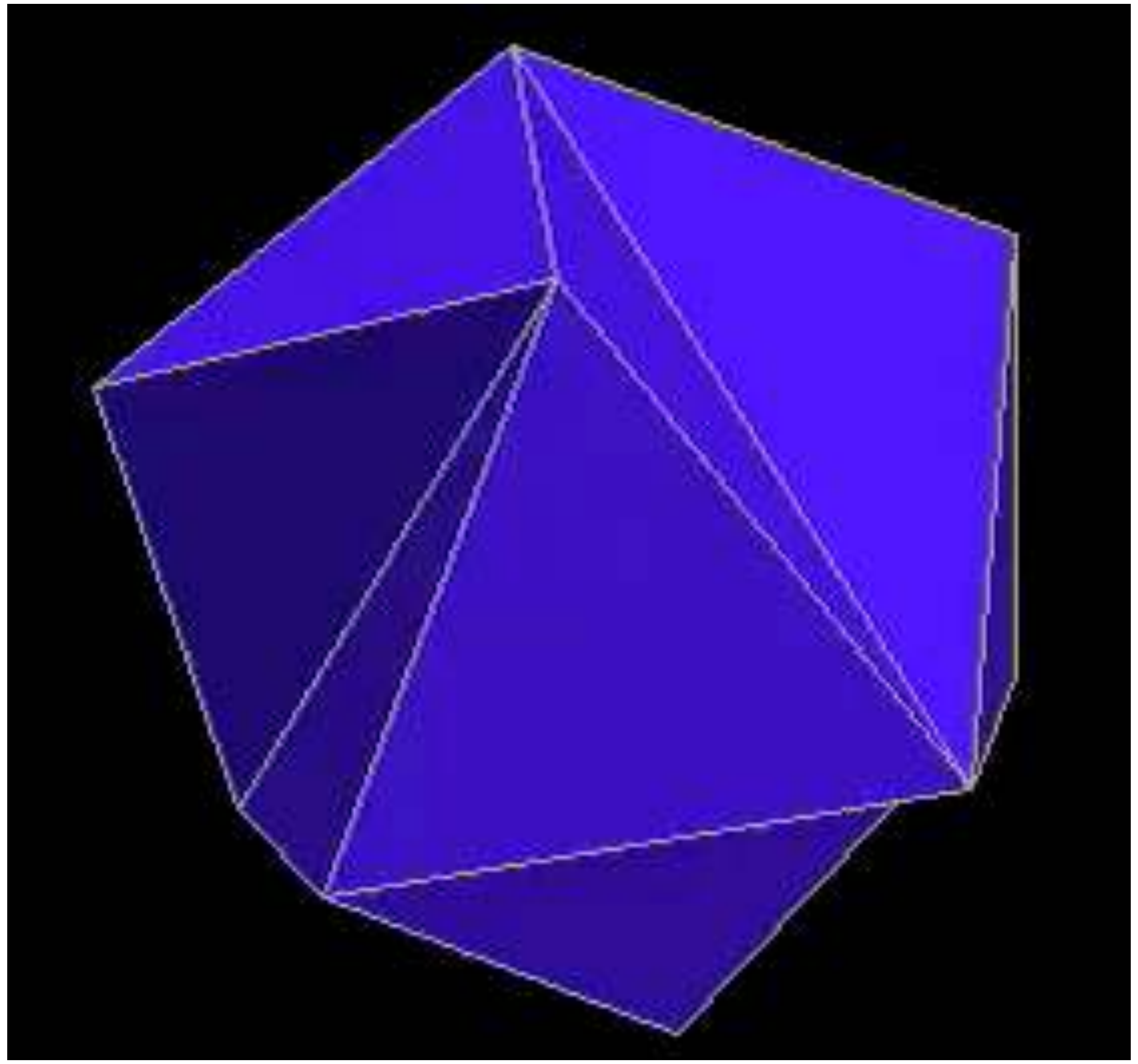


Generalization of Schönhardt
by Rambau



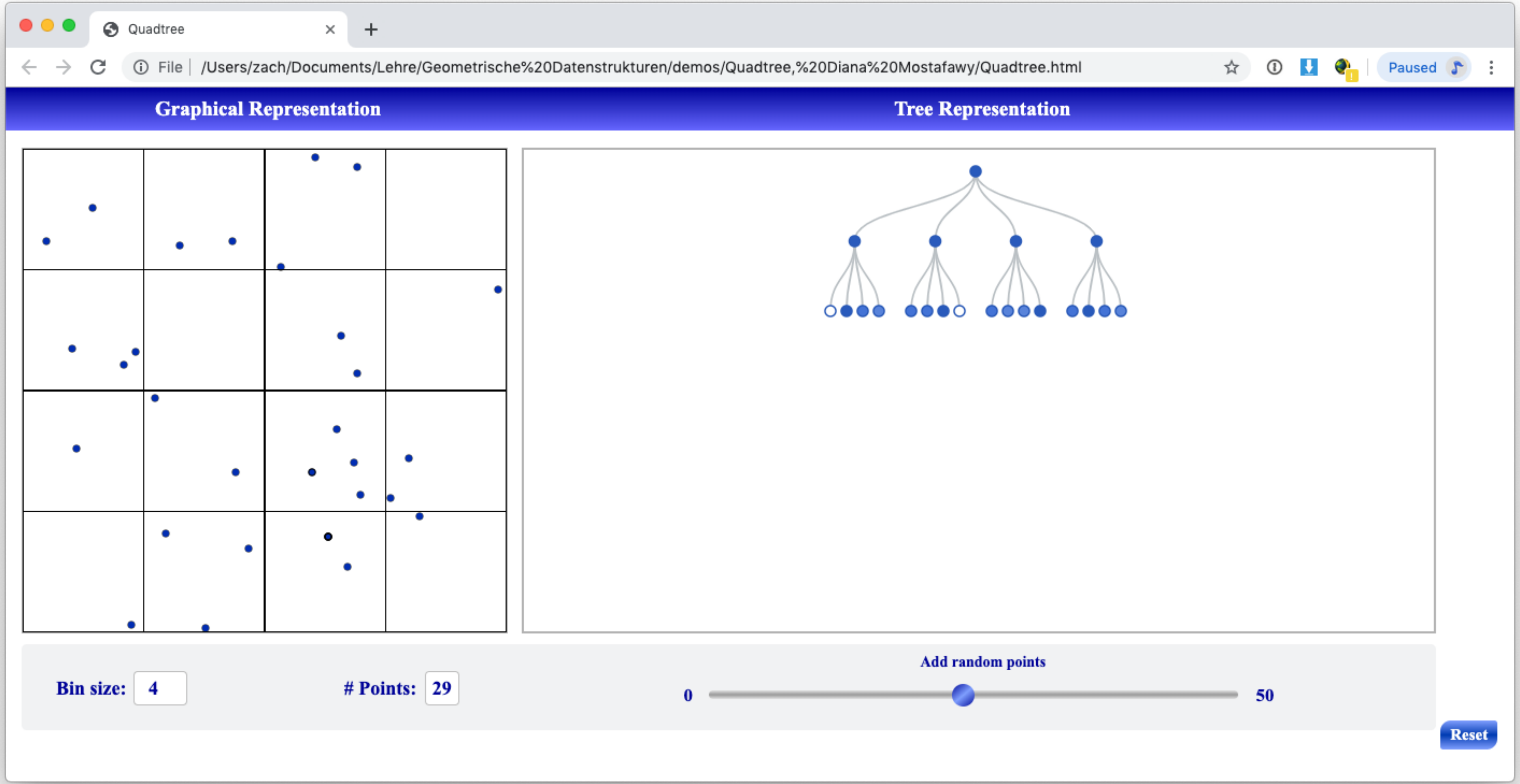




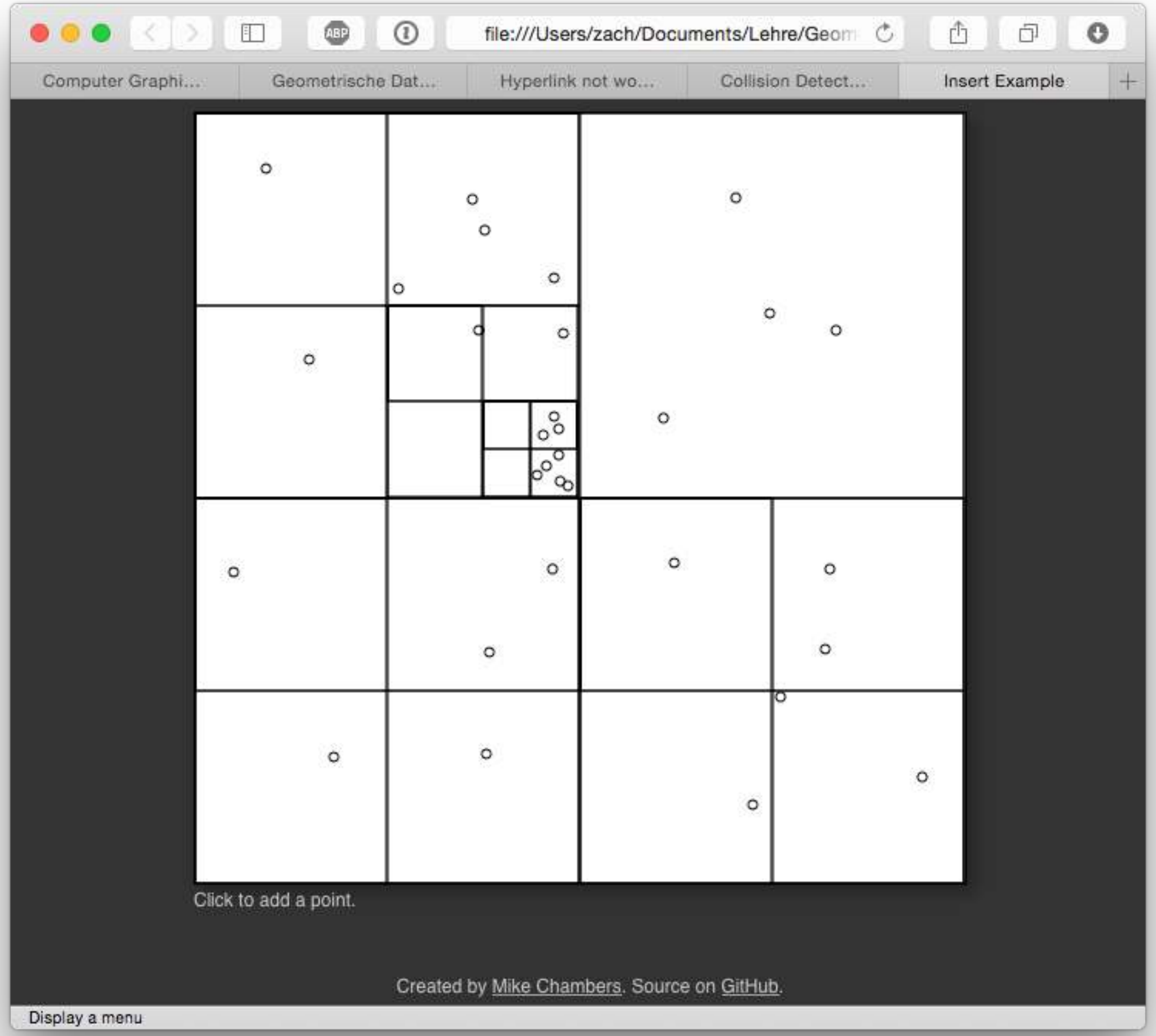


Jessen's Ikosahedron

Point Quadtree Demo



Quadtree Demo



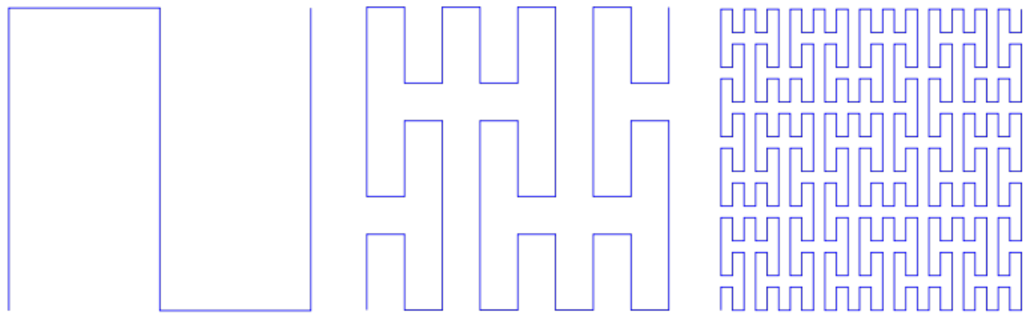
Recursion criterion here: more than 4 points in a node

Space-Filling Curves

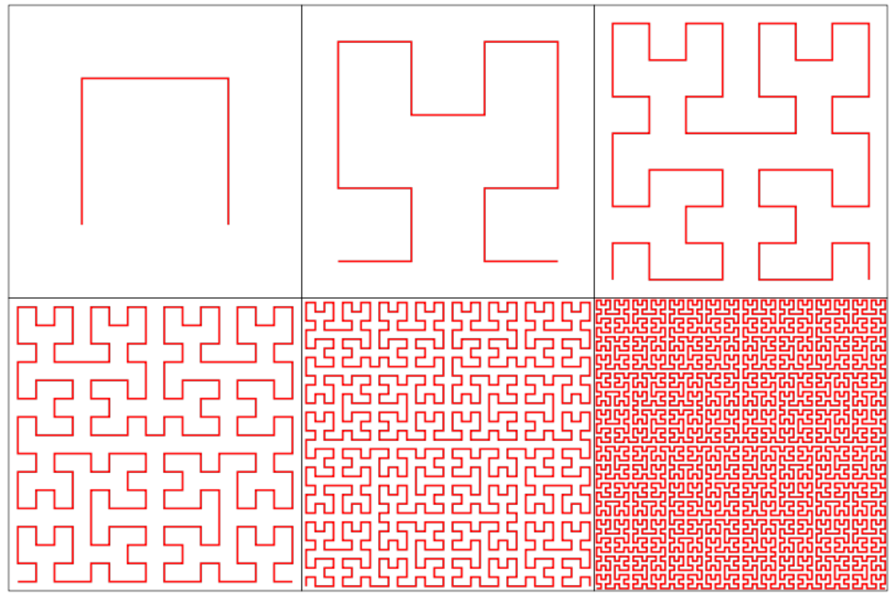
- Definition curve:
  A curve (with endpoints) is a continuous function with domain in the unit interval [0, 1] and range in some d-dimensional space.

- Definition space-filling curve:
  A space-filling curve is a curve with a range that covers the entire 2-dimensional unit square (or, more generally, an n-dimensional hypercube).

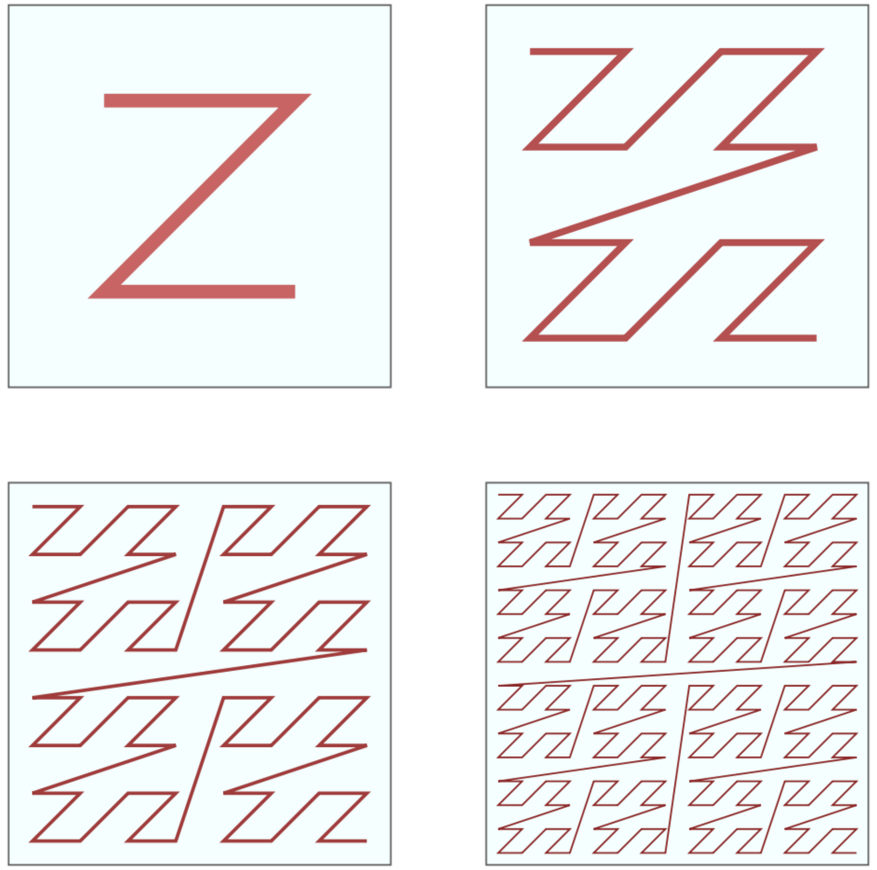
Examples of Space-Filling Curves



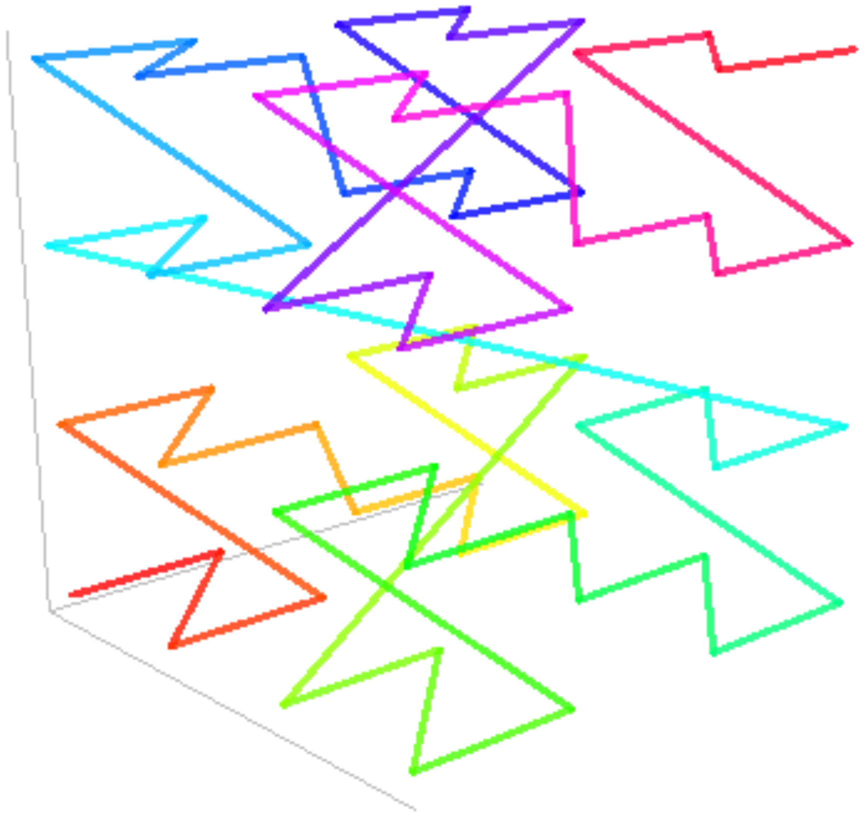
Peano curve



Hilbert curve

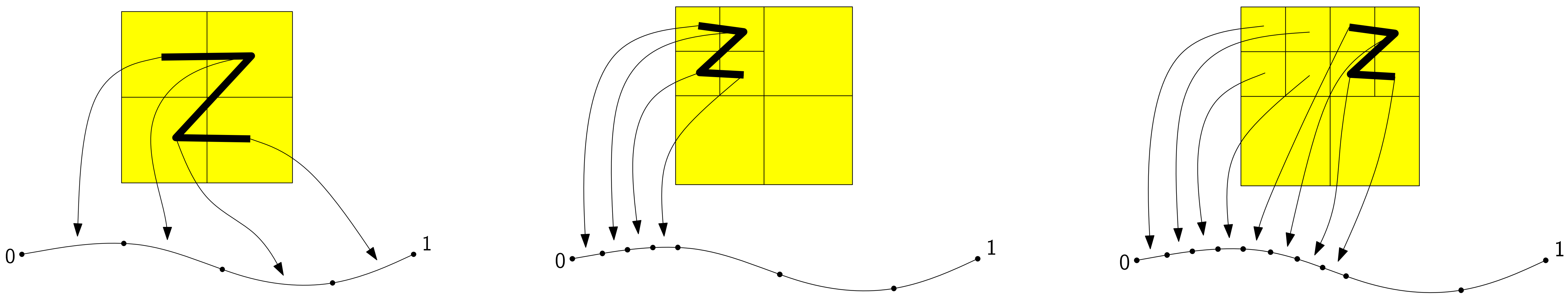


Z-order curve
(a.k.a. Morton curve)



Z-order curve in 3D

- Benefit: a space-filling curve gives a mapping from the unit square to the unit interval
  - At least, the limit curve does that ...

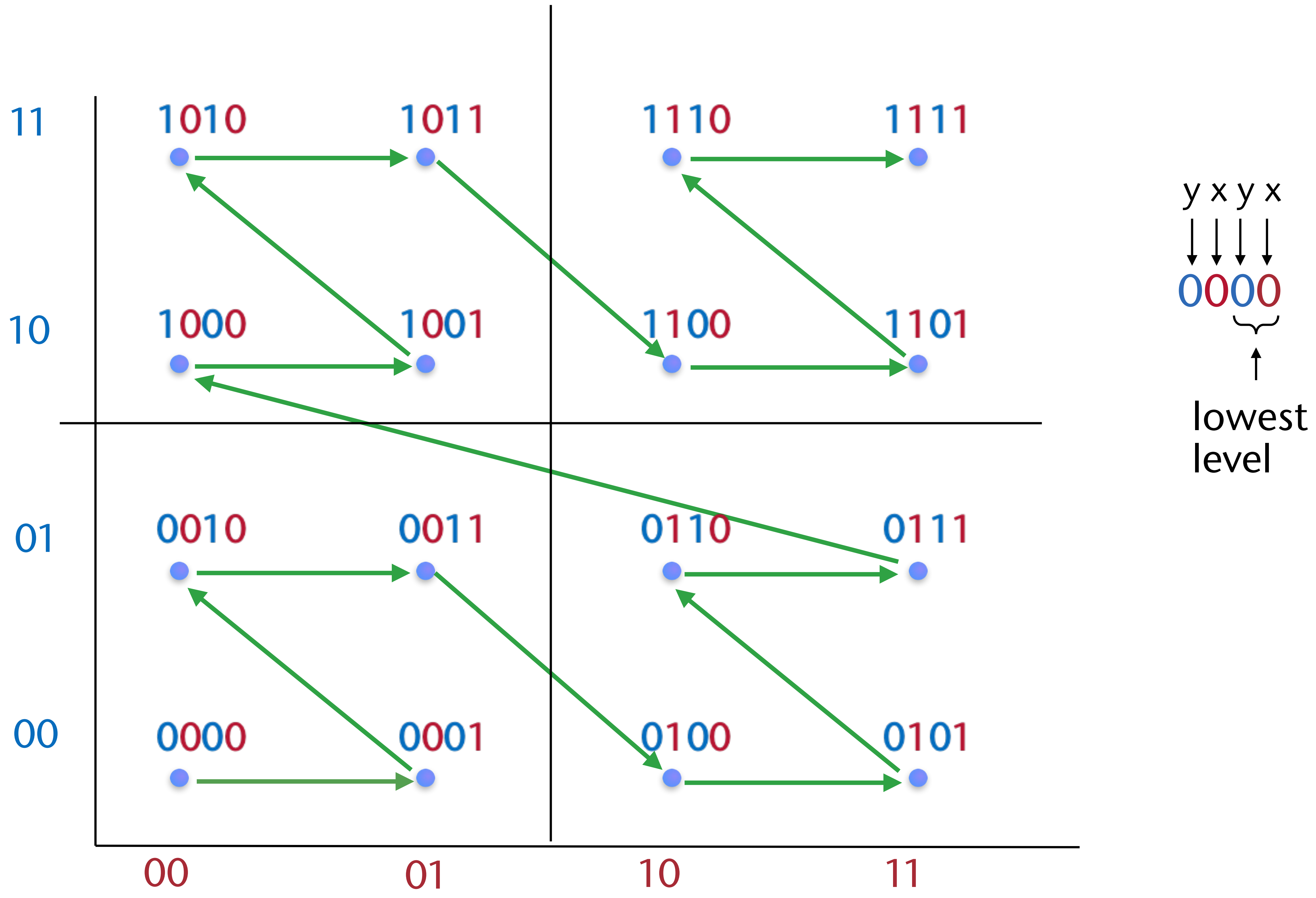


  - We can construct a "space-filling" curve only up to some specific (recursion) level, i.e., in practice space-filling curves are never really *space-filling*

Construction of the Z-Order Curve (here, in 3D)

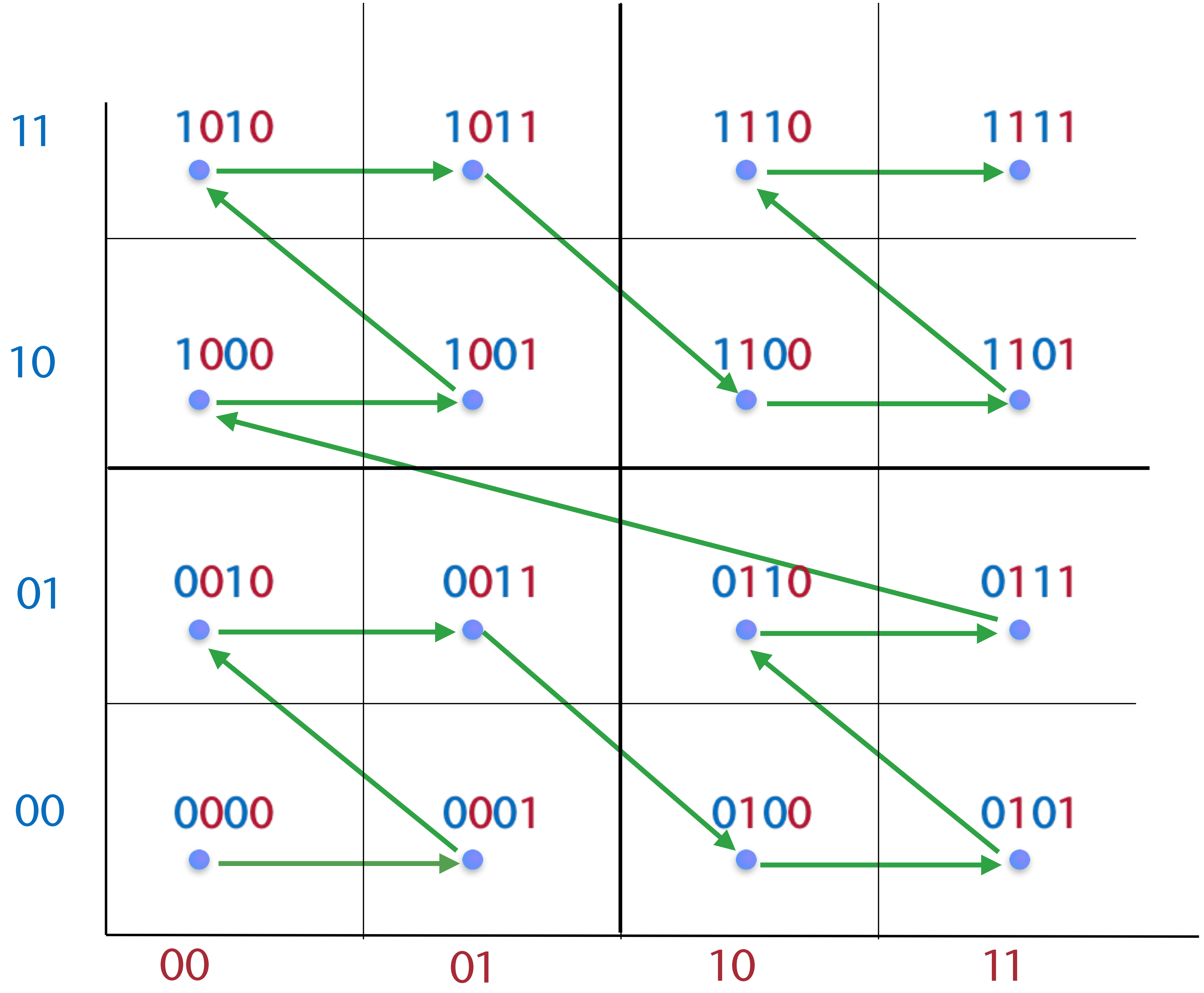
1. Choose a level k
2. Construct a regular lattice of points in the unit cube, 2^k points along each dimension
3. Represent the coordinates of a lattice point p by integer/binary number, i.e., k bits for each coordinate, e.g. $p_x = b_{x,k} \ldots b_{x,1}$
4. Define the Morton code of p as the interleaved bits of the coordinates, i.e., $m(p) = b_{z,k} b_{y,k} b_{x,k} \ldots b_{z,1} b_{y,1} b_{x,1}$
5. Connect the points in the order of their Morton codes $\longrightarrow$ z-order curve at level k

Example



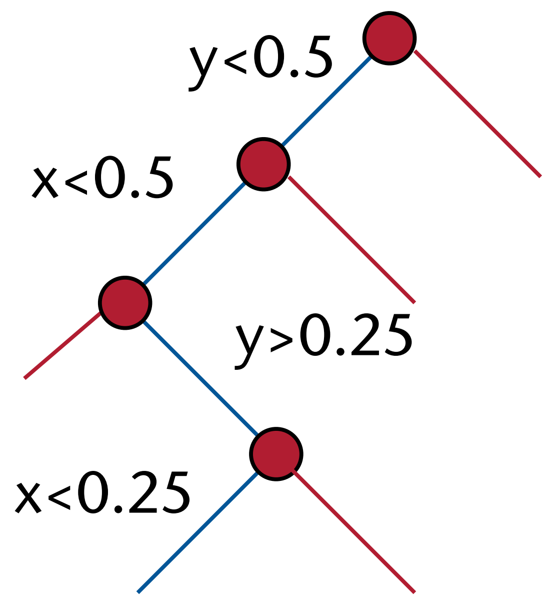
11
10
01
00
1010
1011
1110
1111
1000
1001
1100
1101
0010
0011
0110
0111
0000
0001
0100
0101
00
01
10
11
y x y x
0000
lowest level

Note: the Z-curve induces a grid (actually, a complete quadtree)



Properties of Morton Codes (here, in 2D)

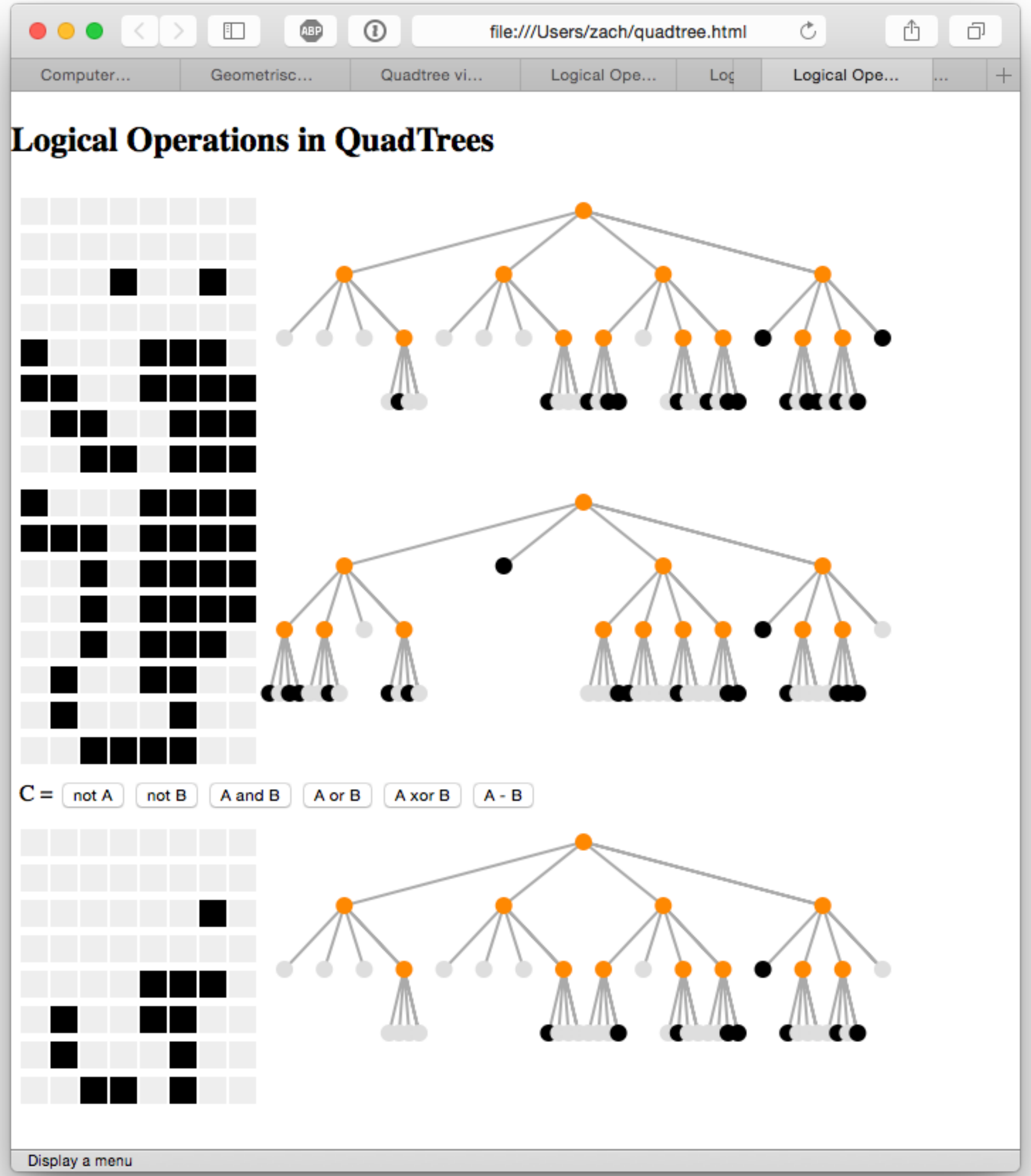
- The Morton code of each point is *2k* bits long
- All points *p* with Morton code *m(p)* = 0xxx lie below the plane *y* = 0.5
- All points with m(p) = 11xx lie in the upper right quadrant of the square
- If we build a quadtree/octree on top of the grid, then the Morton code encodes the *path* of a point, from the root to the leaf that contains the point ("0" = left, "1" = right)



- The Morton codes of two points differ for the first time – when read from left to right – at bit position *h* ⇔

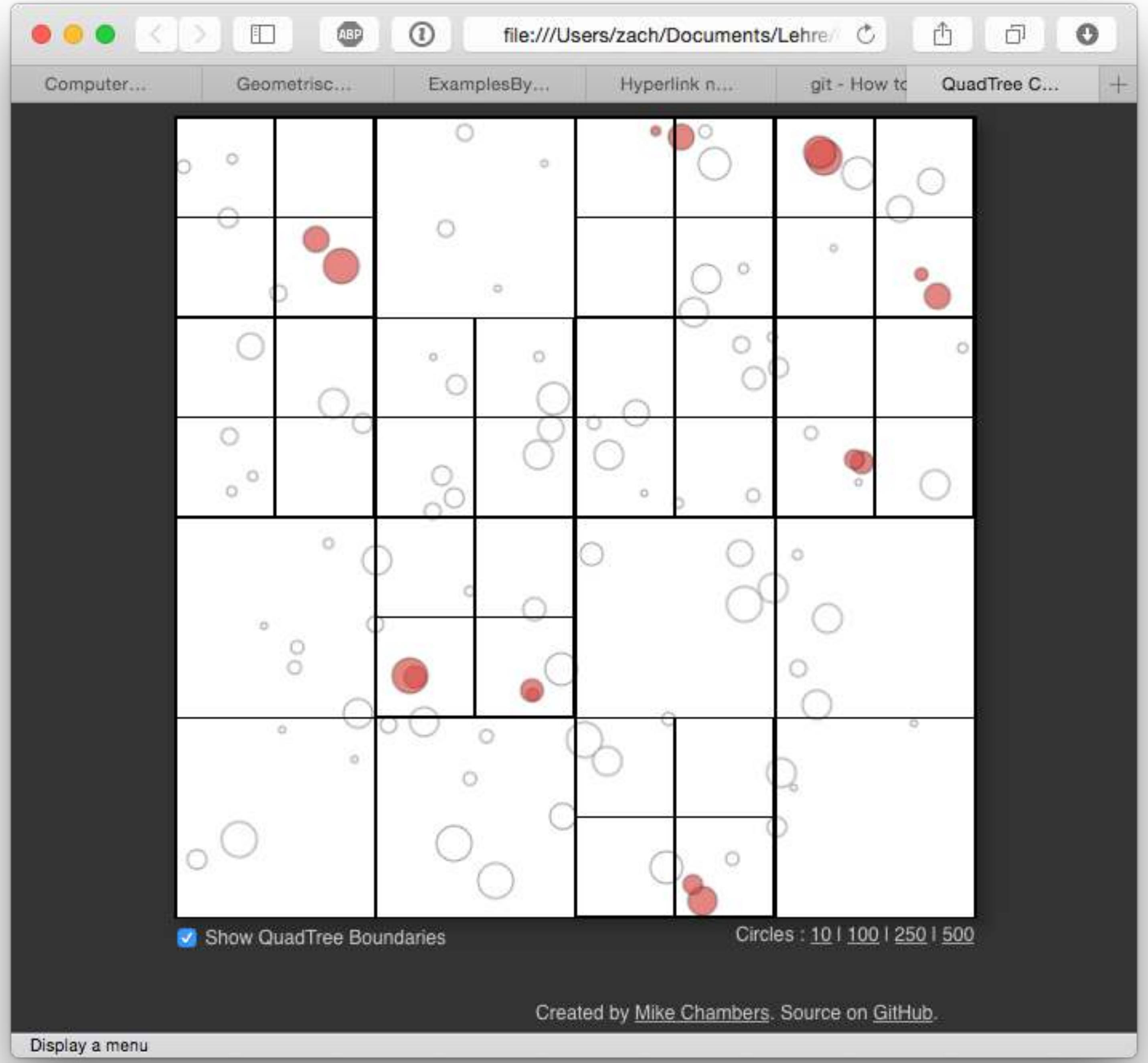
  the paths in the binary tree over the grid split at level *h*

Logic Operations with Quadtrees



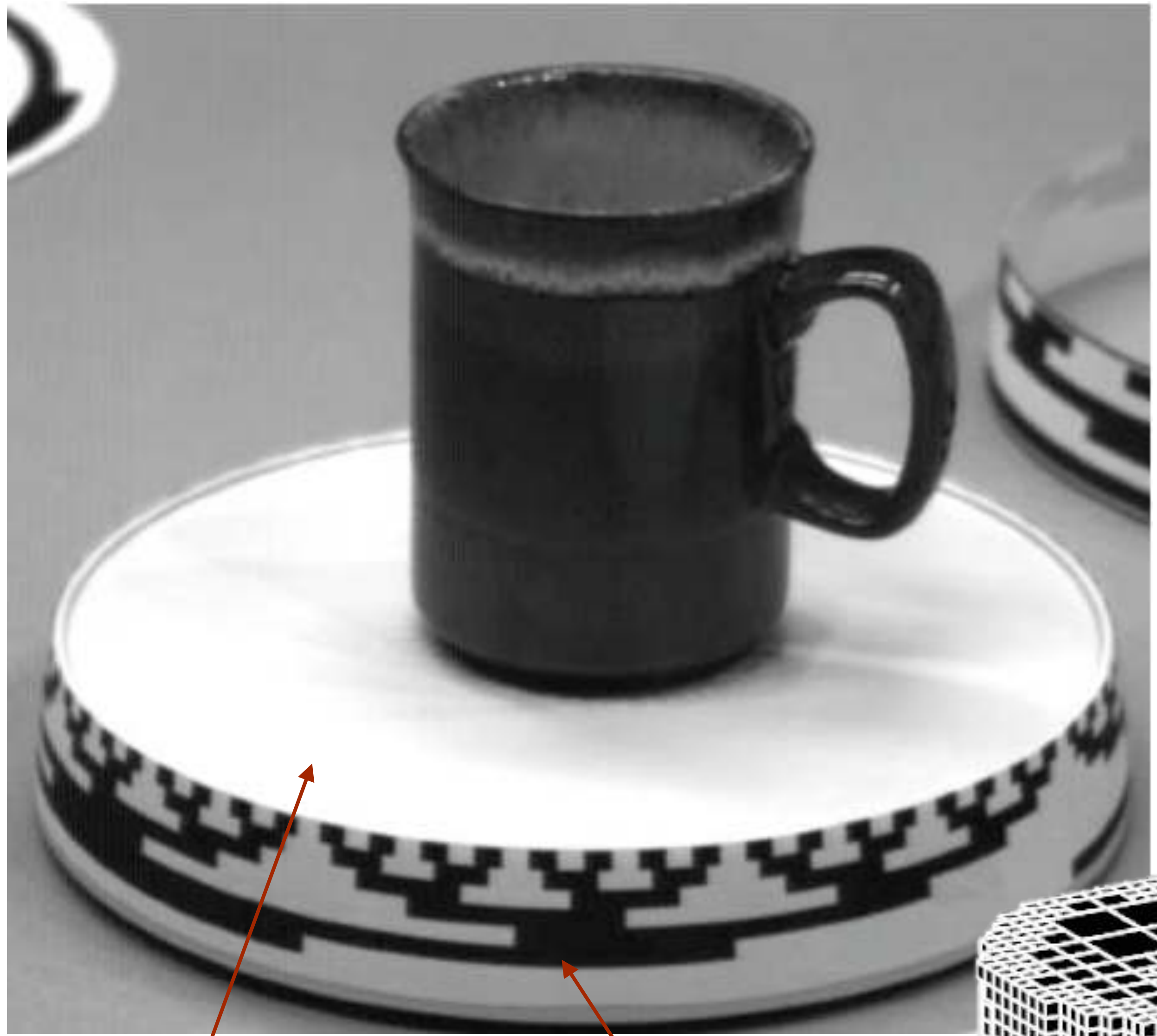
http://blog.ivank.net/quadtree-visualization.html

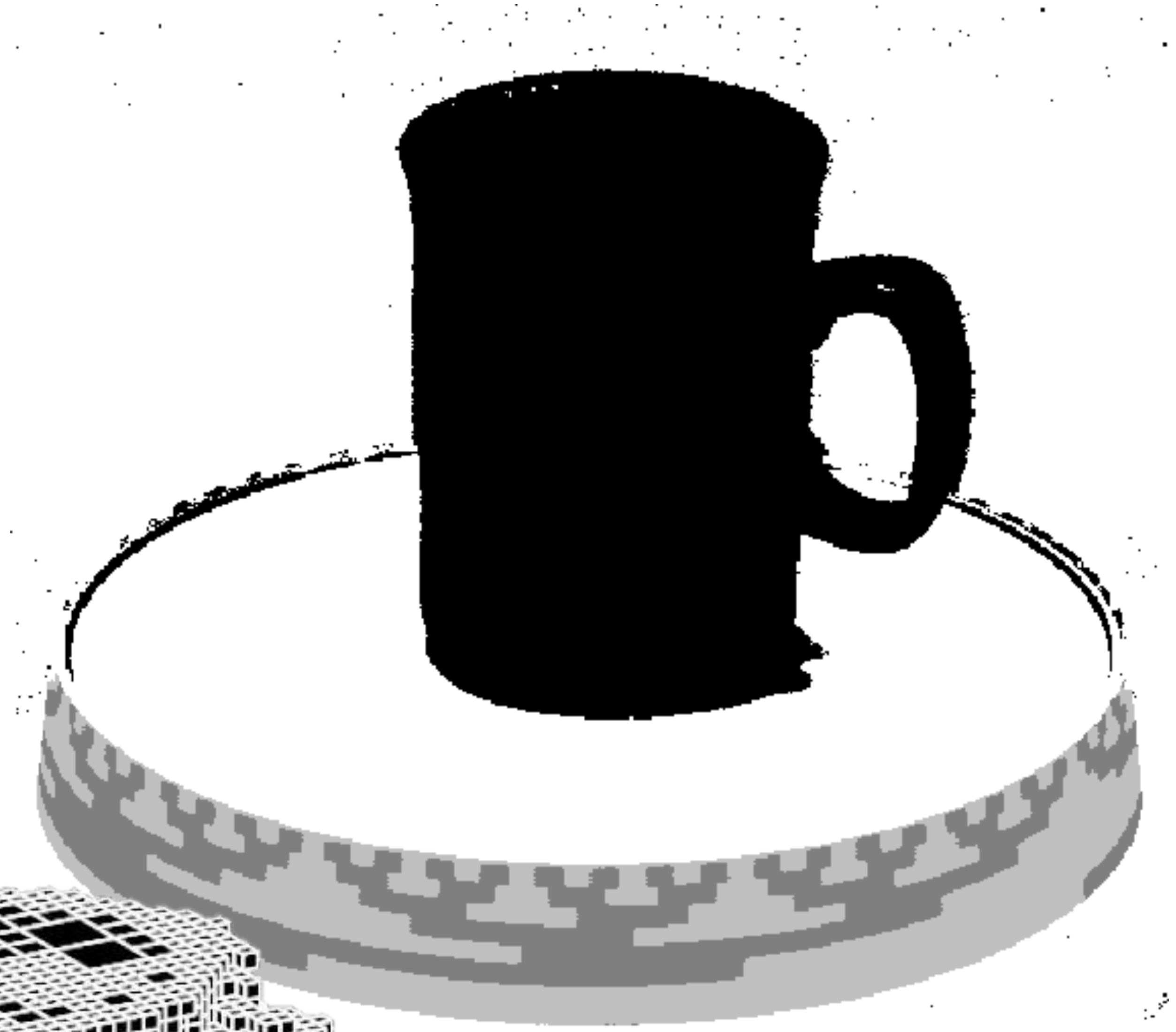
Acceleration of "Collision Detection" by Quadtrees

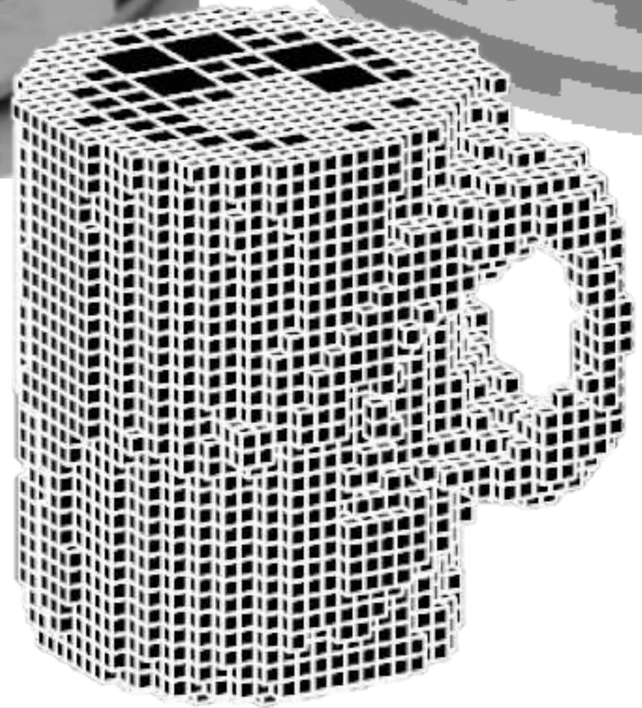


http://www.mikechambers.com/blog/2011/03/21/javascript-quadtree-implementation/

Octree Models from Images







Drehteller

Gray Code
(zur Erkennung der
Orientierung des
Drehtellers)

Example Models

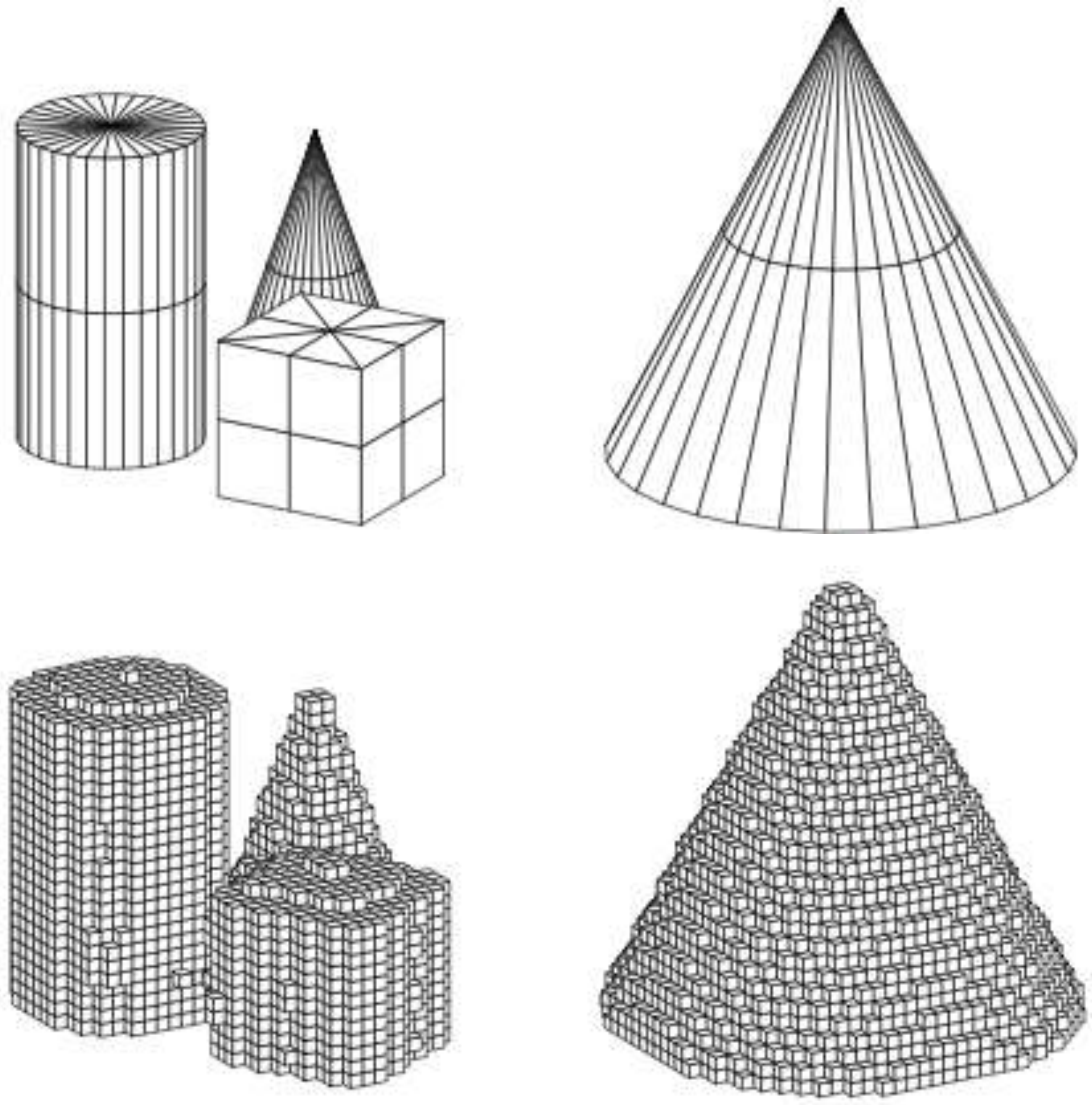
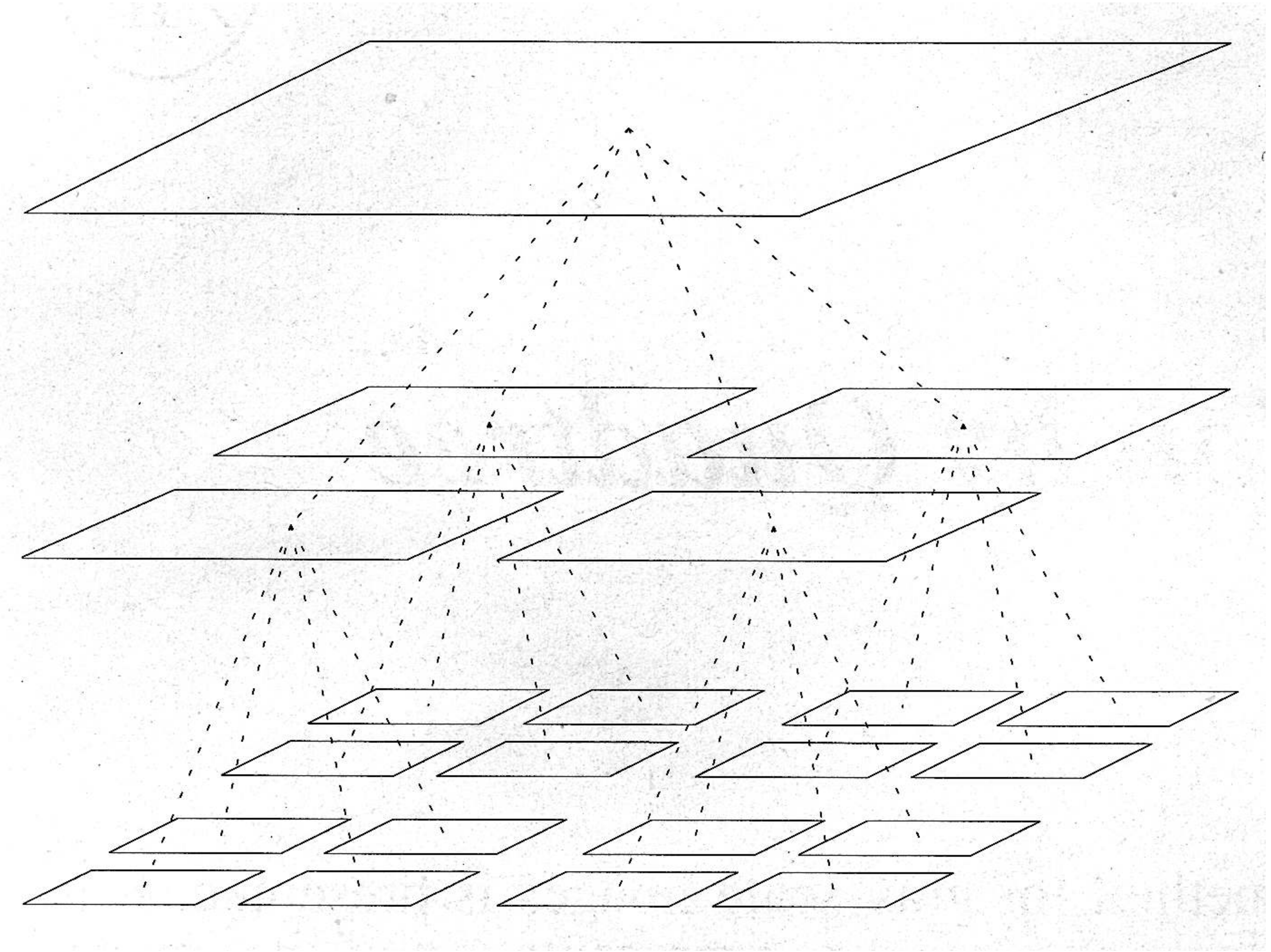
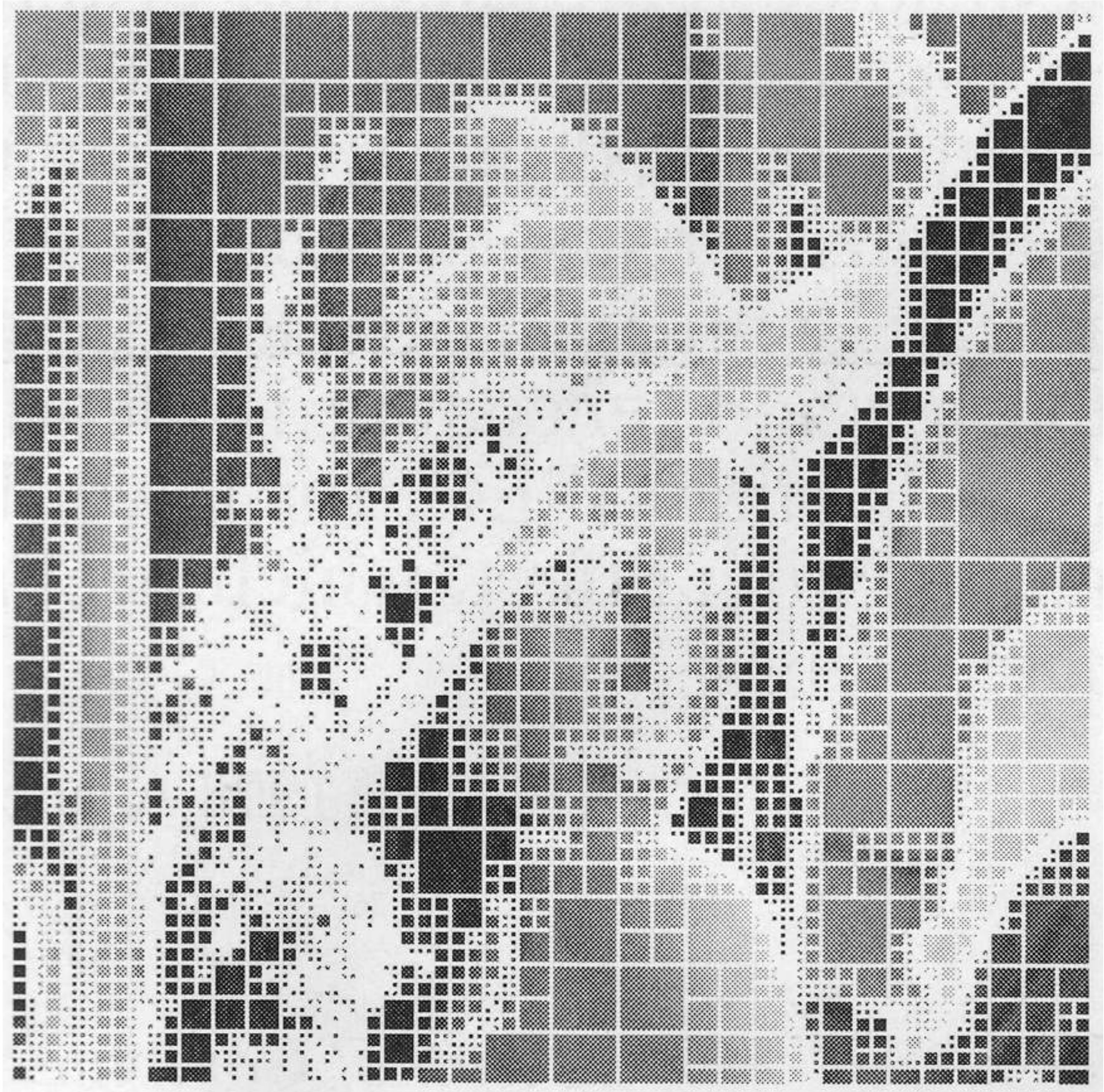


Image Compression using Quadtrees



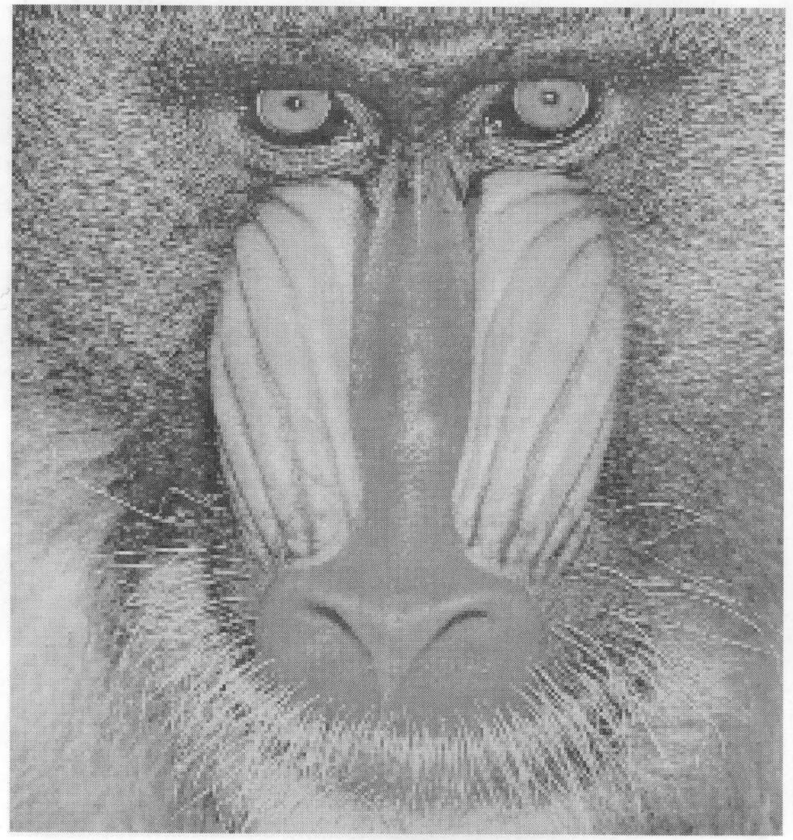


The two test images par excellence

Results

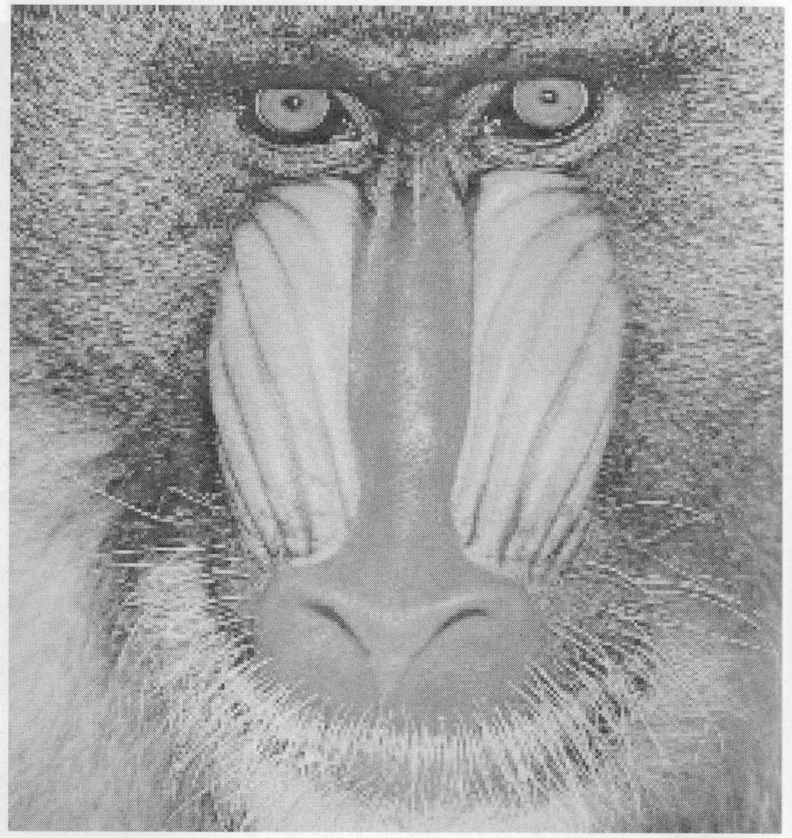
QP: 1.03 bits per pixel

QP: 1.95 bits per pixel

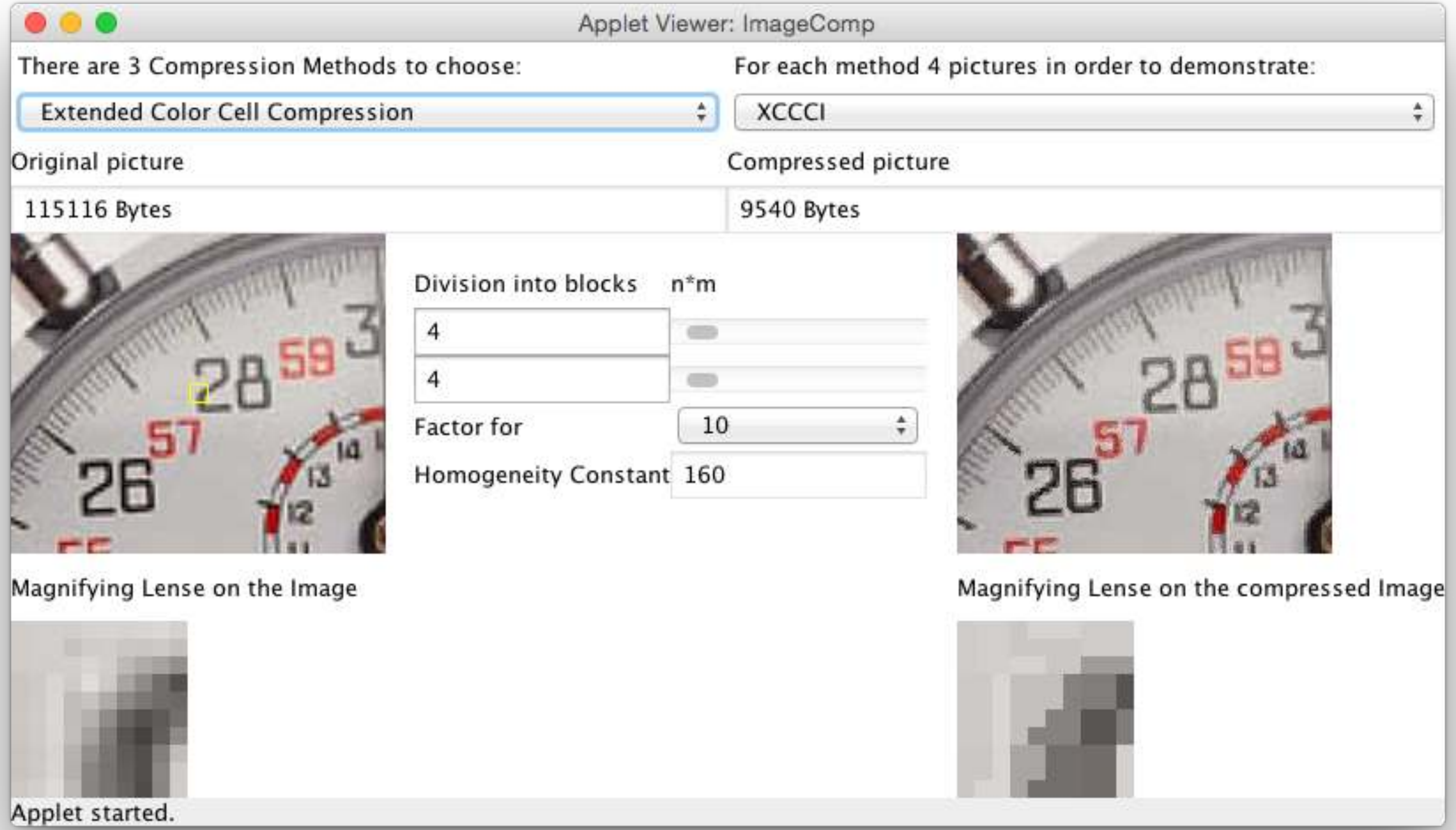


JPEG: 1.00 bits per pixel

JPEG: 1.99 bits per pixel



Demo for BTC and CCC Compression

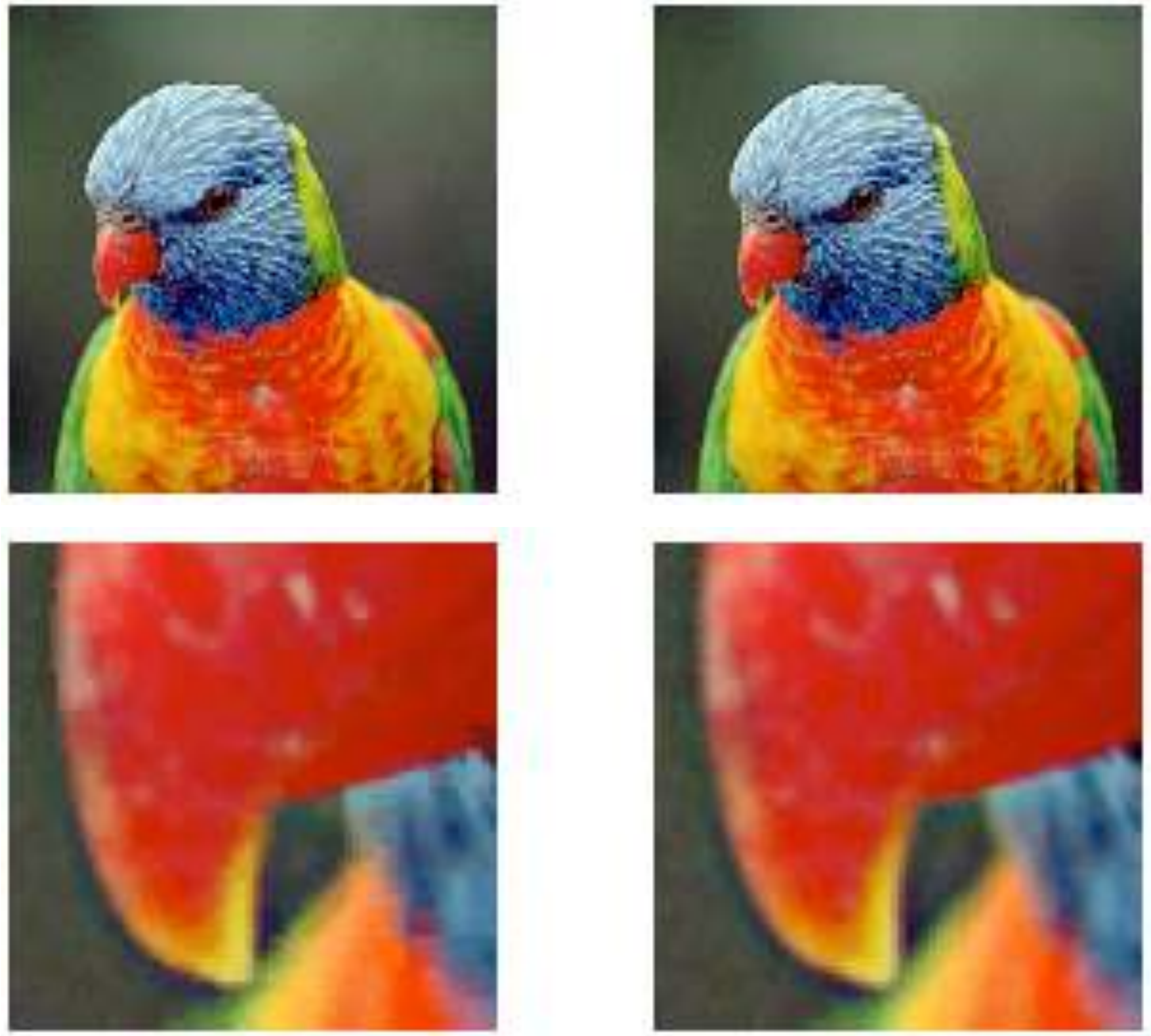


http://ls.wim.uni-mannheim.de/de/pi4/teaching/animations/

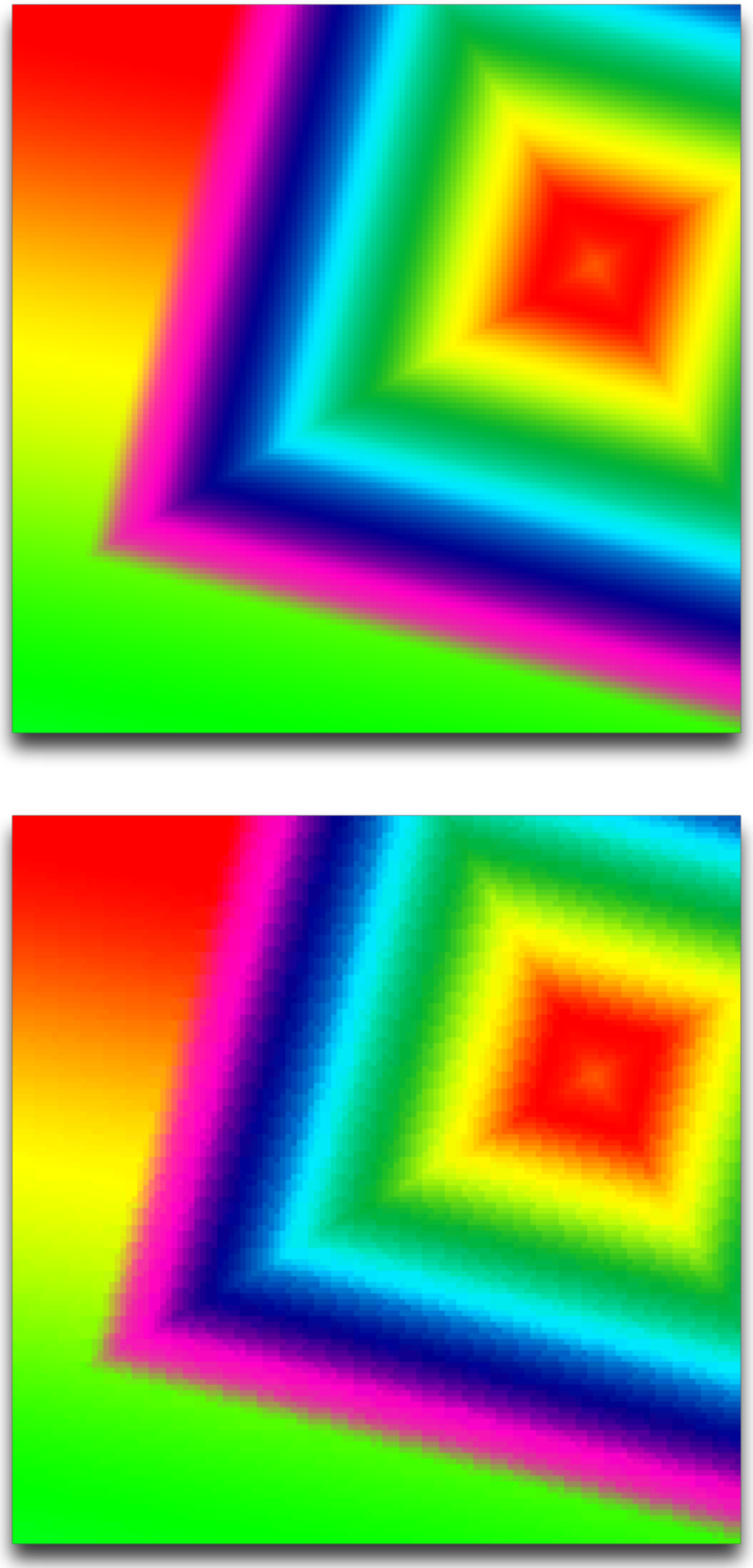
S3TC Texture Compression

- Comparison:

DXT1 Uncompressed



[Philipp Klaus Krause]



Uncompressed

DXT1

[Simon Brown]

- Advantage: bigger textures possible → higher quality
- Example from the Unreal Engine:

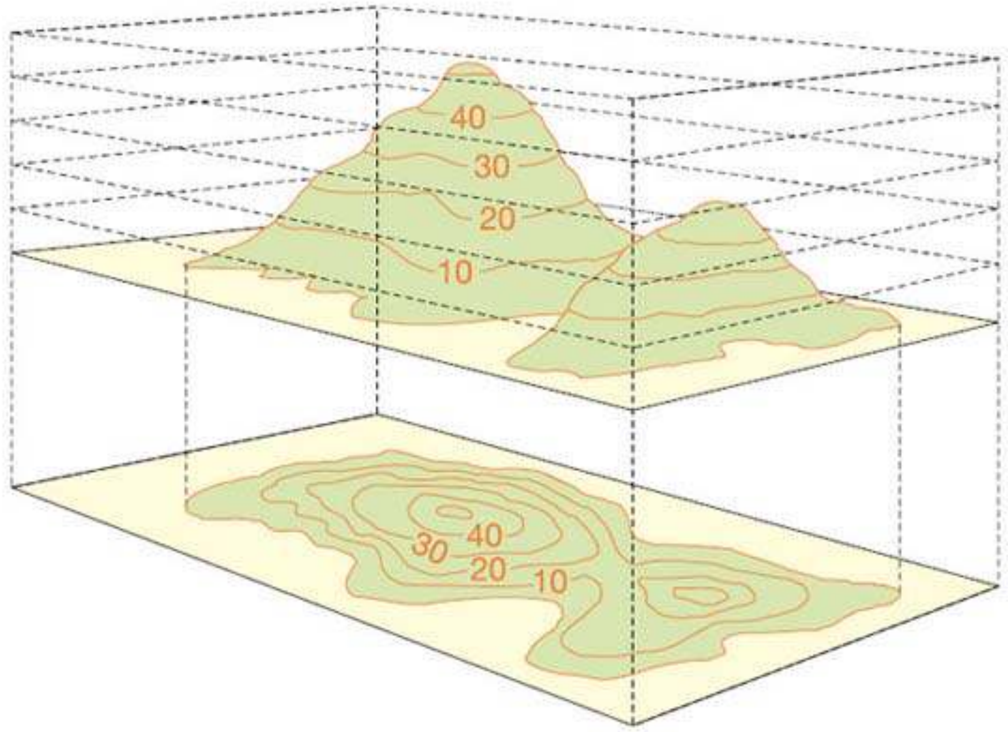
uncompressed

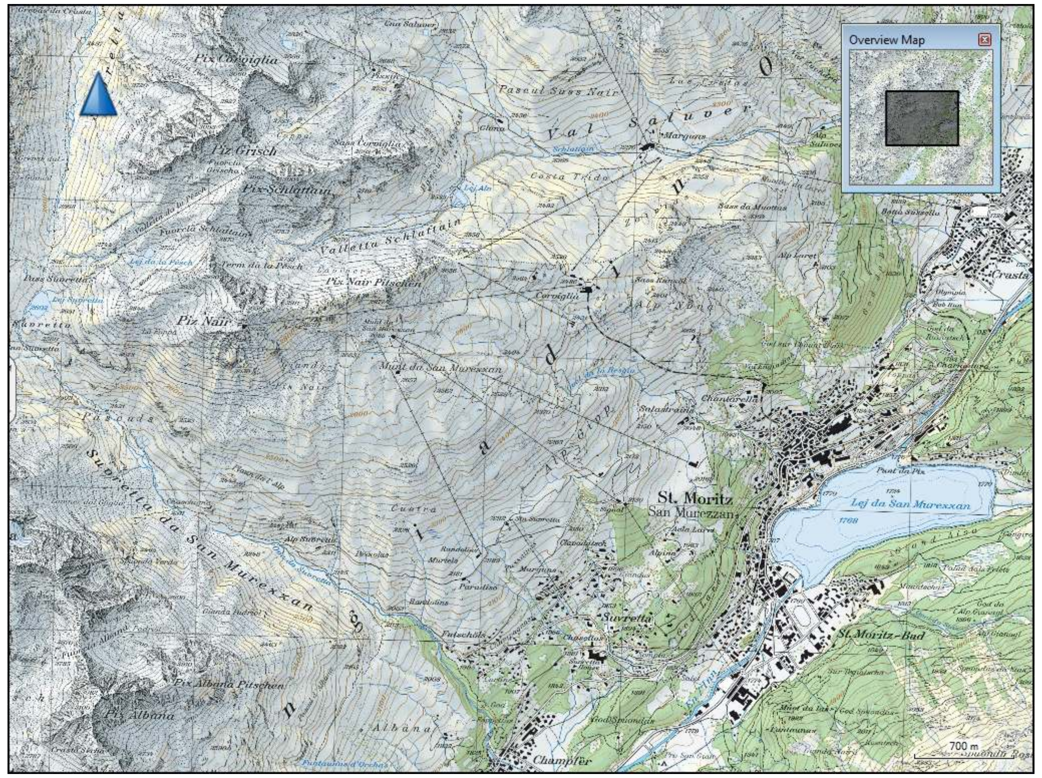
with S3TC

Unreal Retexturing Project

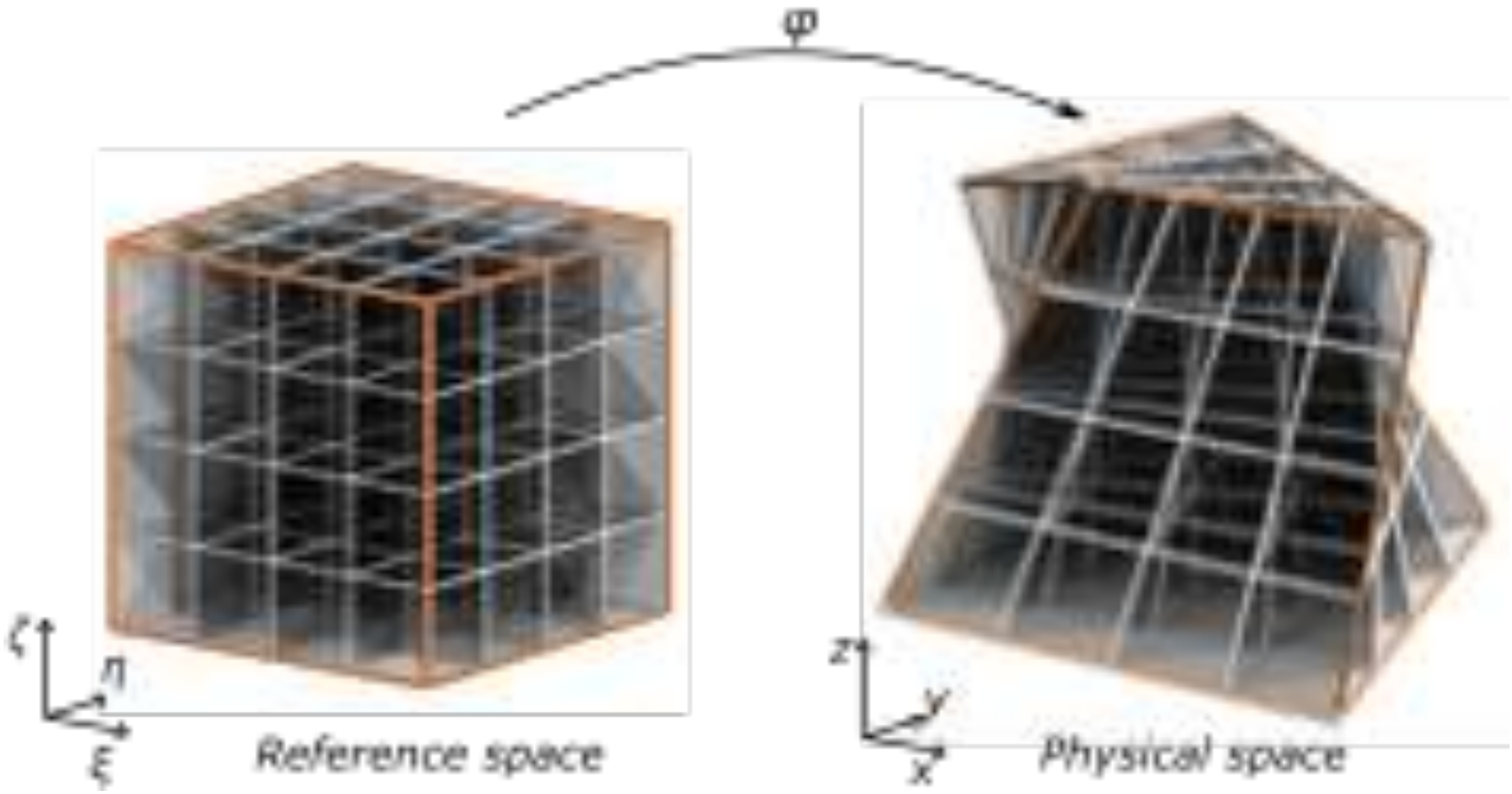
Isosurfaces

- 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





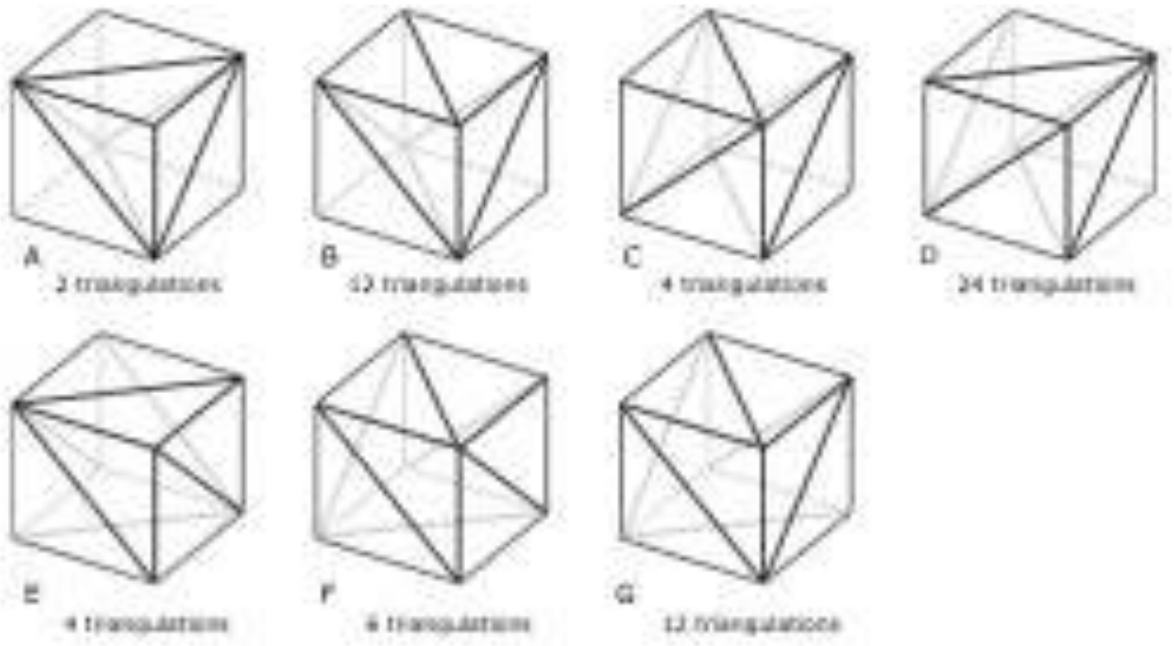
Computational vs Physical Space



How Many Triangulations has the Hexahedron?

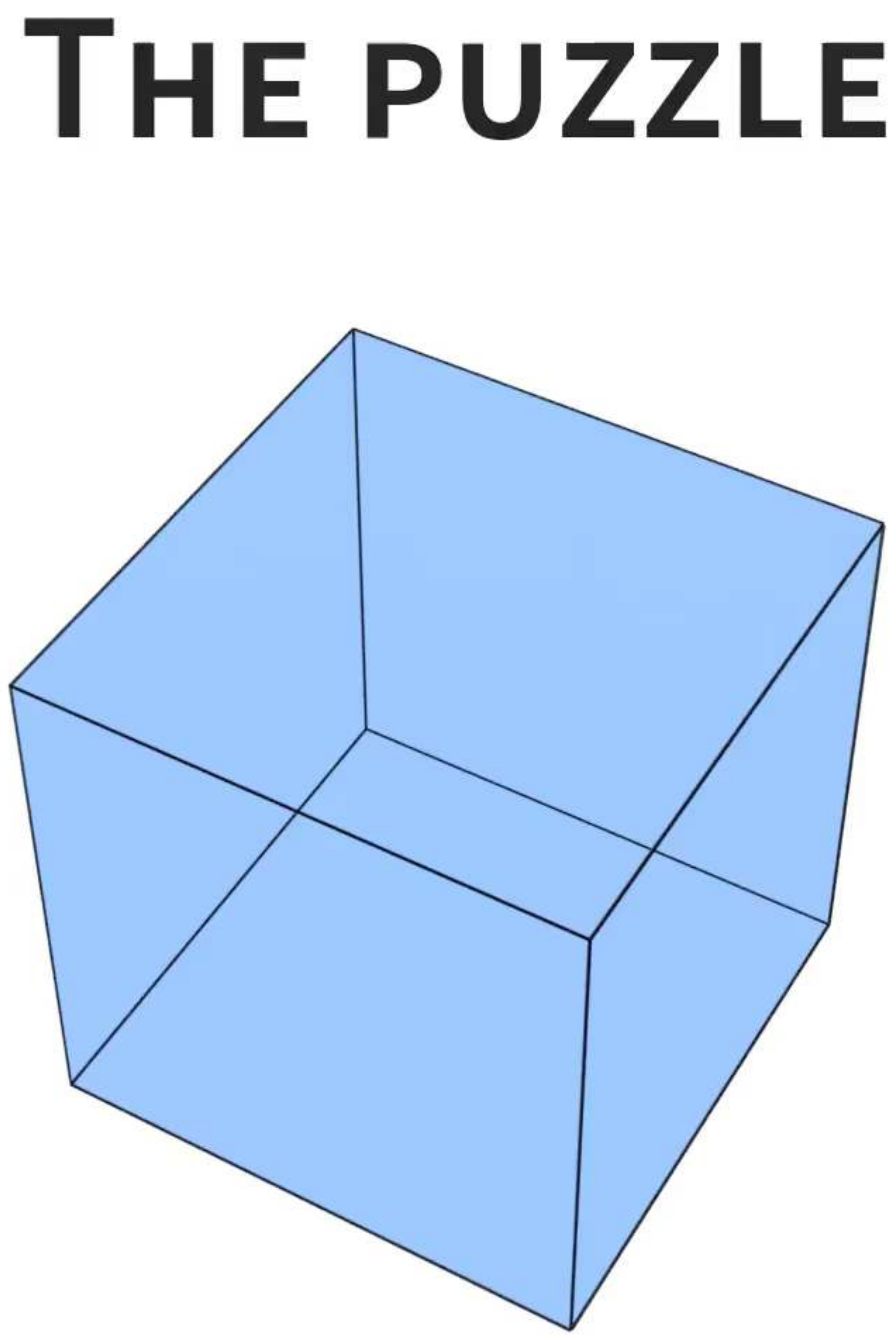
(= Tetrahedralizations)

- Cube $\longrightarrow$ 2 triangulations
- Hexahedron:
  - Triangulation must conform to border of hexahedron
  - 12 edges are fixed, 8 edges have 2 possibilities $\longrightarrow$ 2^6 possibilities to triangulate the surface of a hexahedron
  - Each of these could lead to a number of different triangulation of the hexahedron



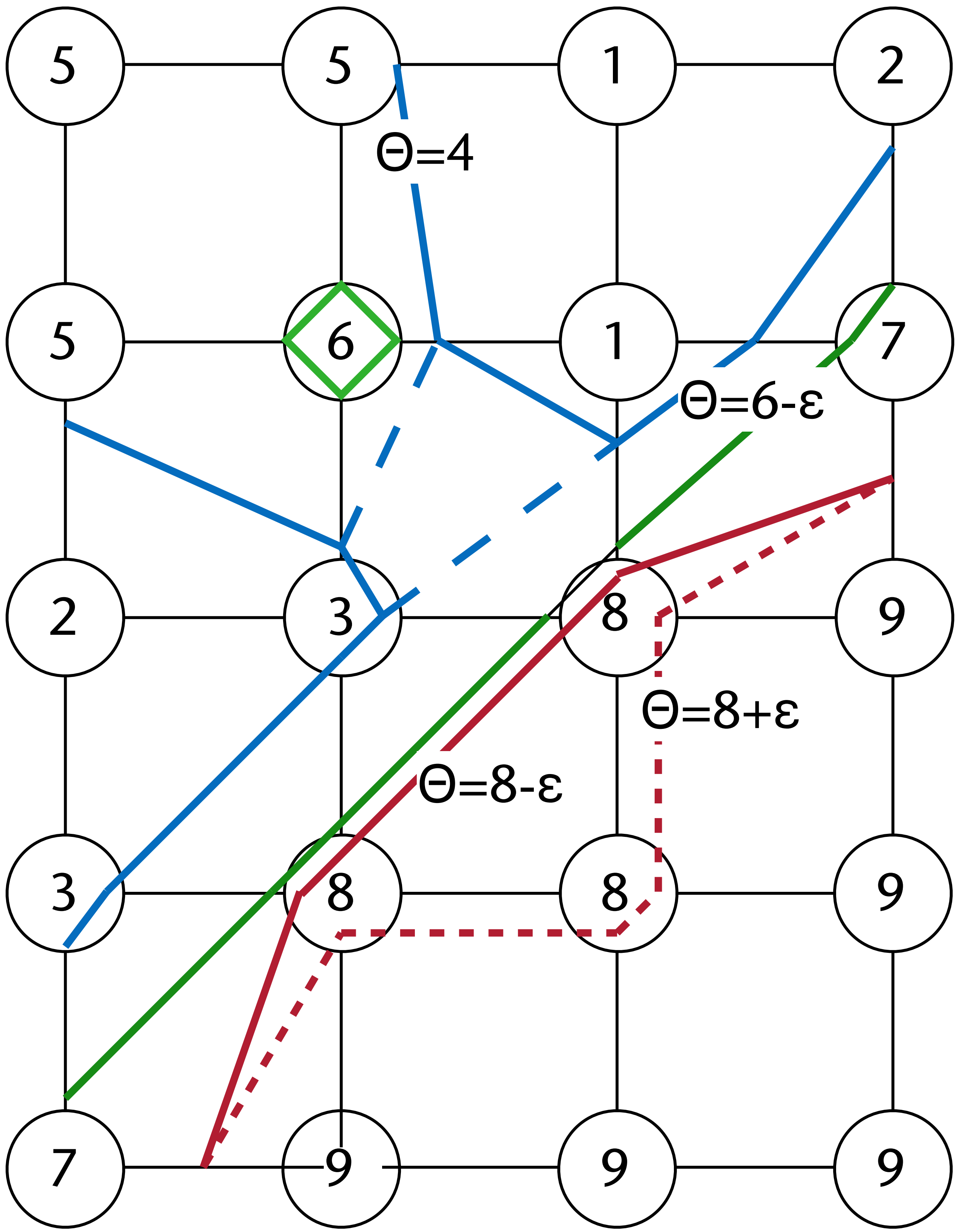
- Question: how many are there combinatorially? do all have a geometric

Video

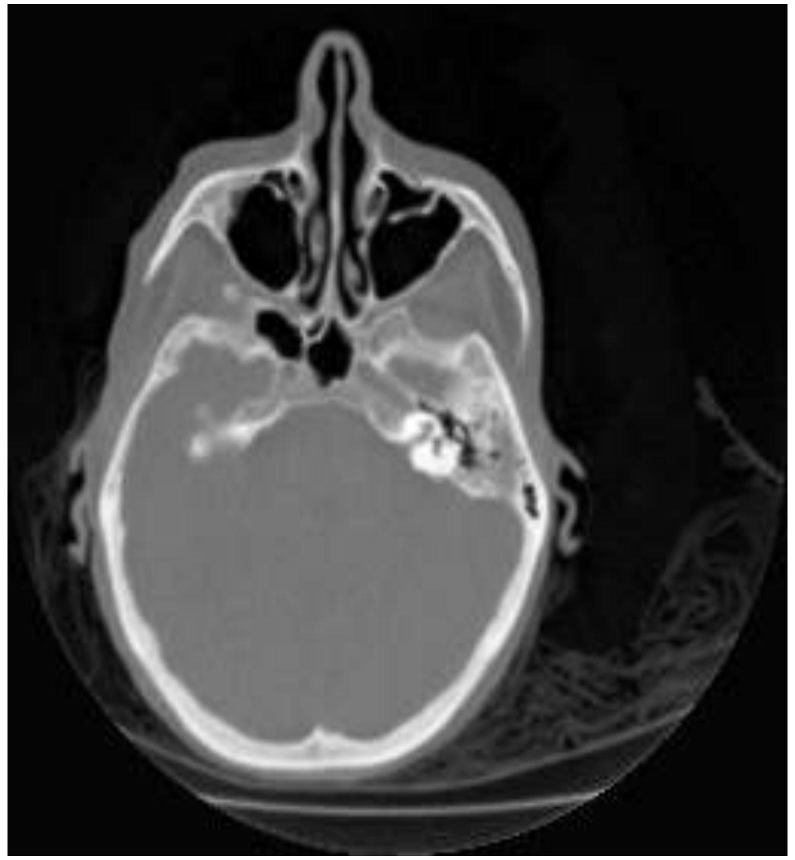


Problems / Challenges With Isosurface Computation

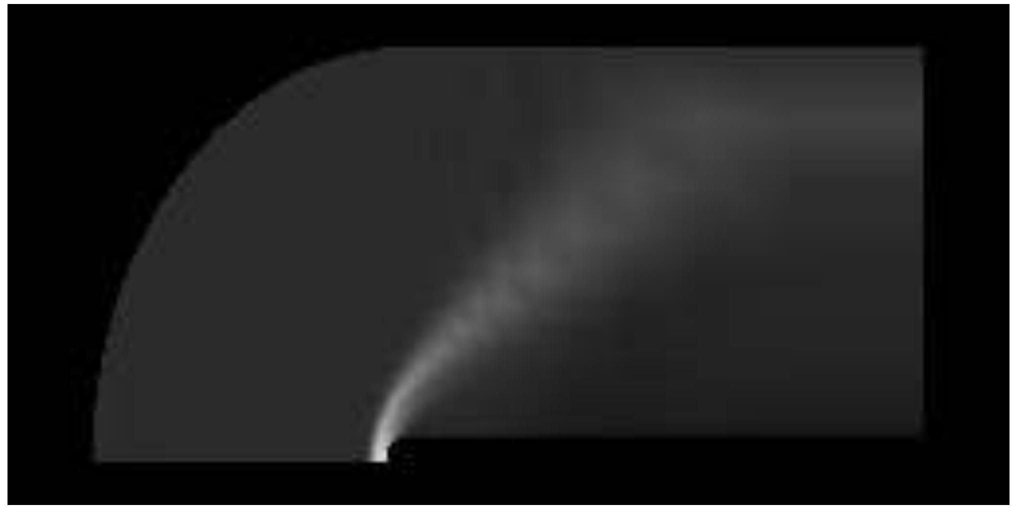
- Singularities ⟶ isosurface contracts to a point, or appears "out of nowhere" when isovalue crosses that point
- Ambiguities during tesselation
- Plateaus ⟶ large "jumps" of the location of the isosurface when isovalue changes by ε



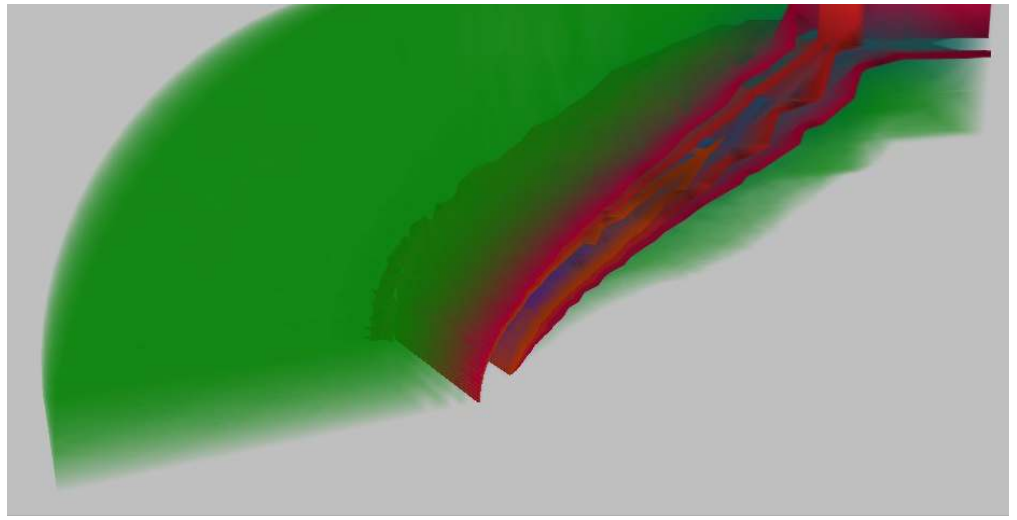
Examples for volume data records

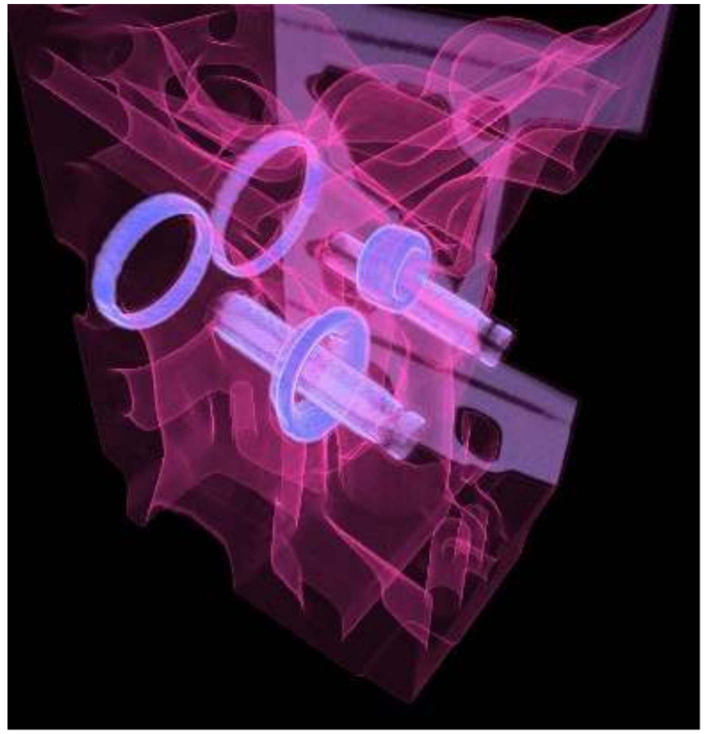


Chapel Hill CT Head

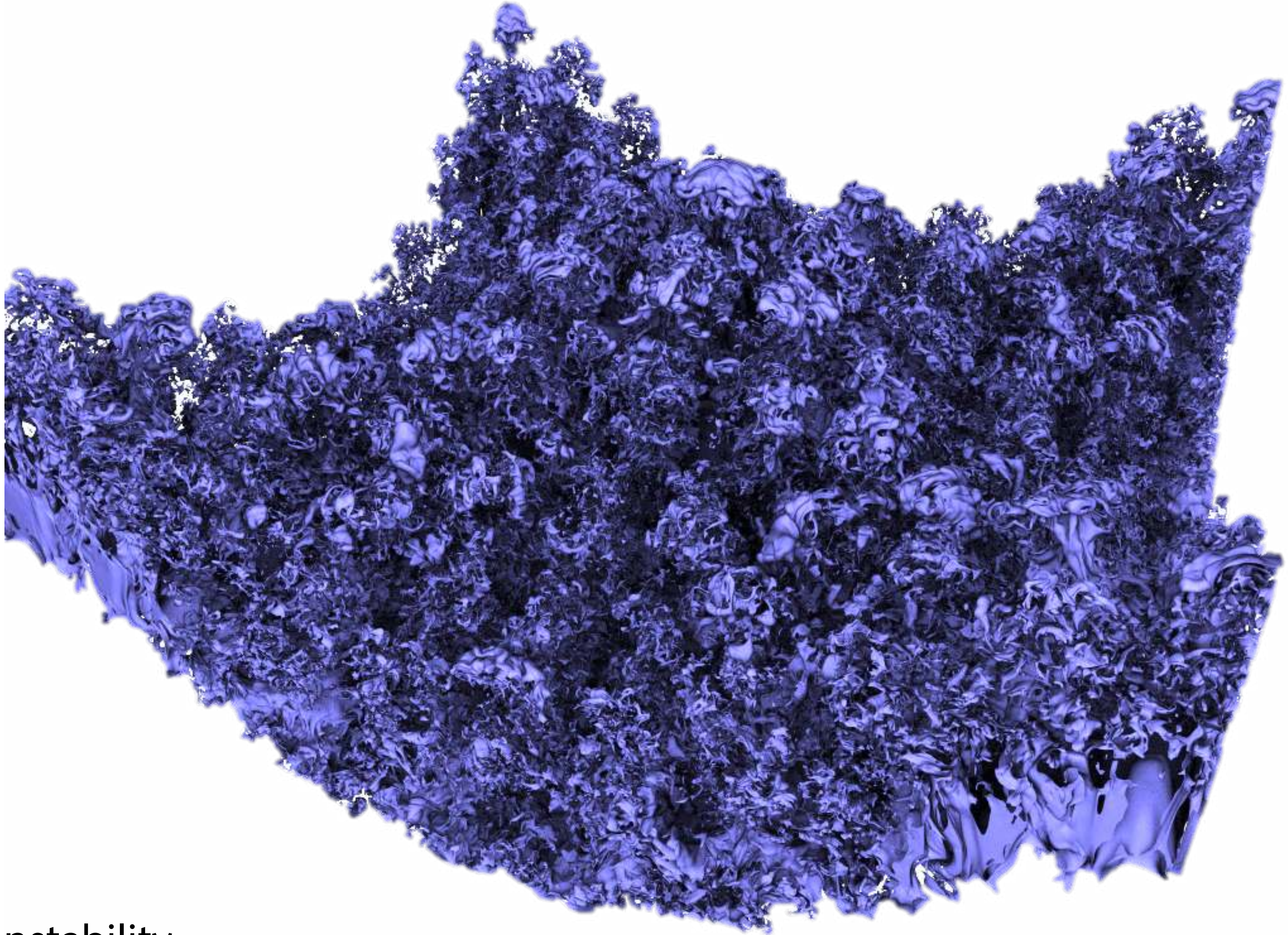


Blunt Fin





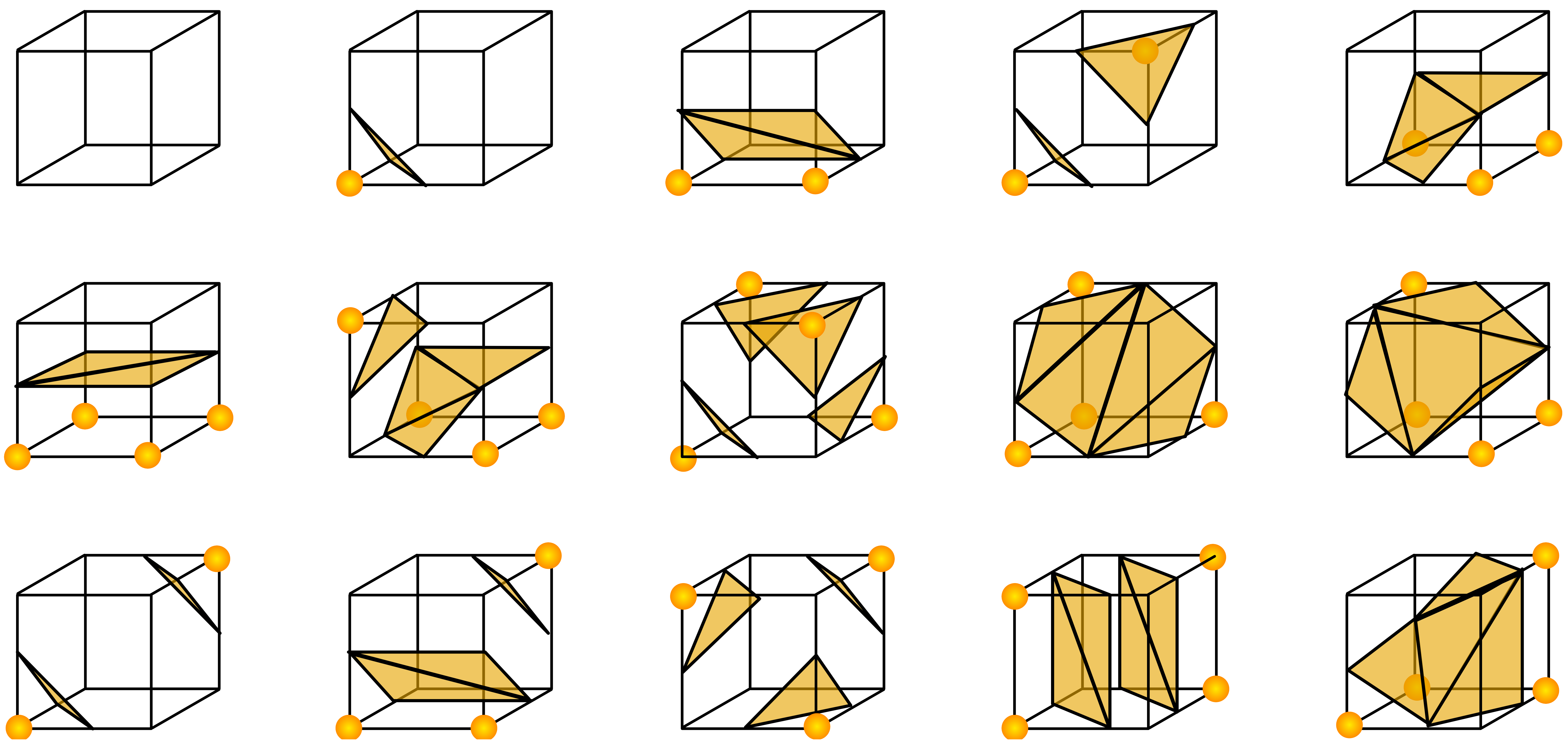
Engine Block



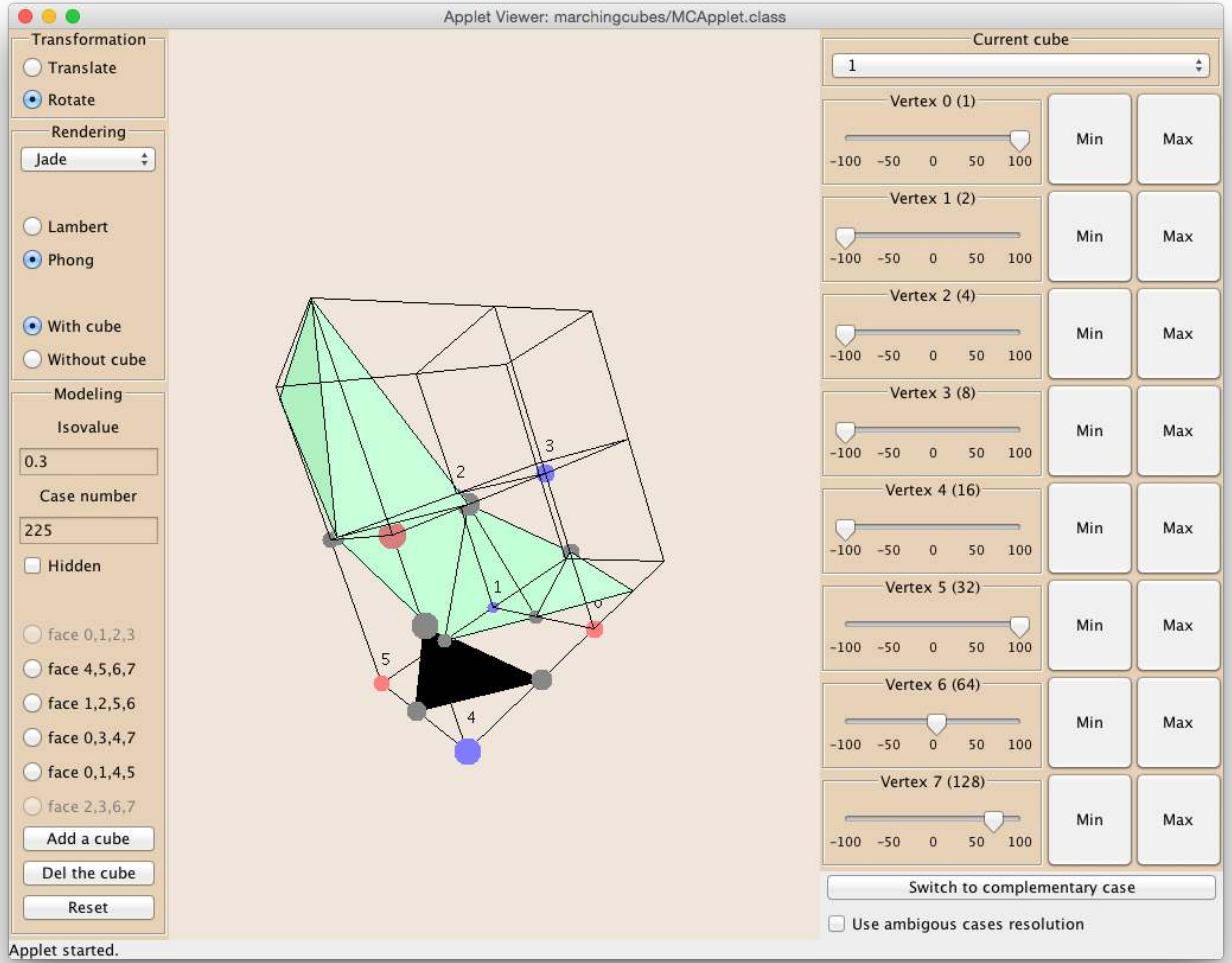
Isosurface of the
Richtmyer-Meshkov Instability
(Lawrence-Livermore National Labs (LLNL))

290M triangles, volume data set = 2.1 TB

The 15 really different cases in 3D Marching Cubes (modulo rotation & mirroring)



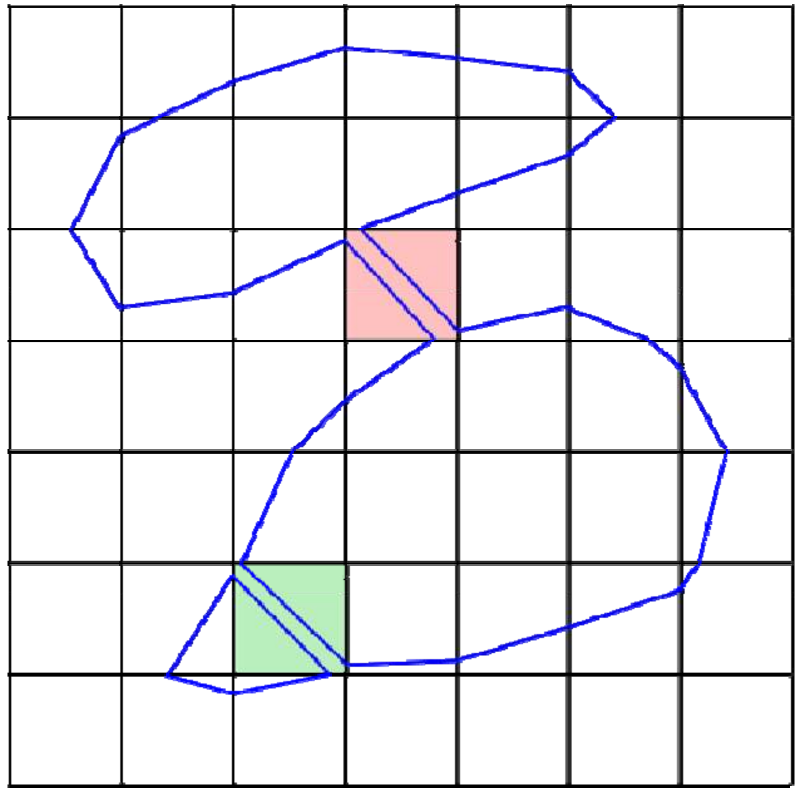
Demo



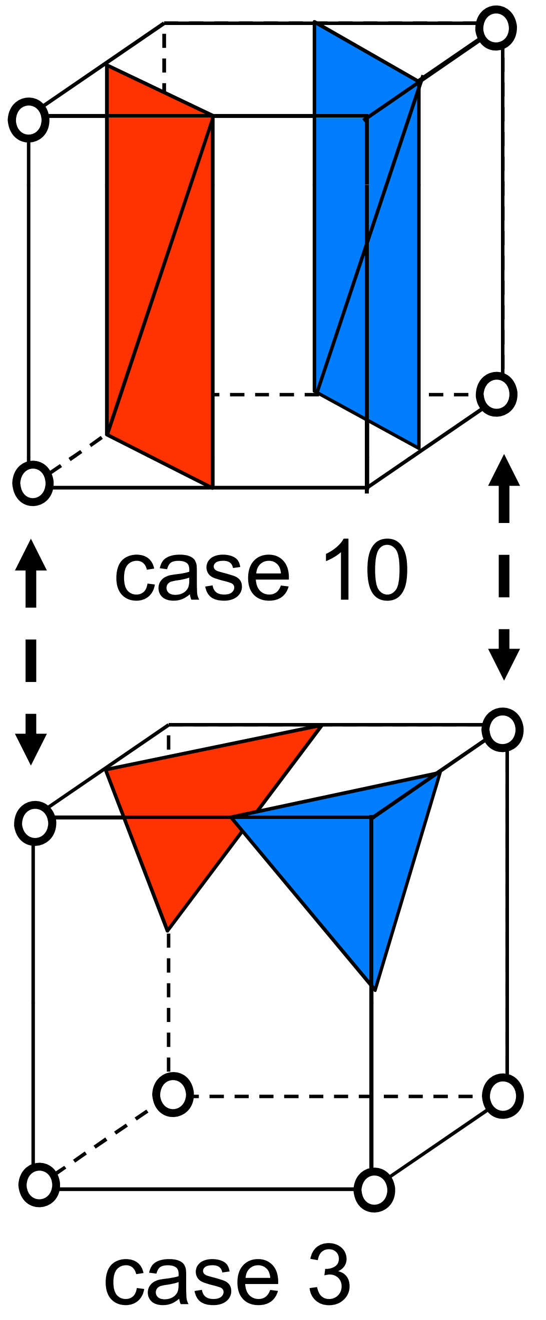
http://users.polytech.unice.fr/~lingrand/MarchingCubes/applet.html

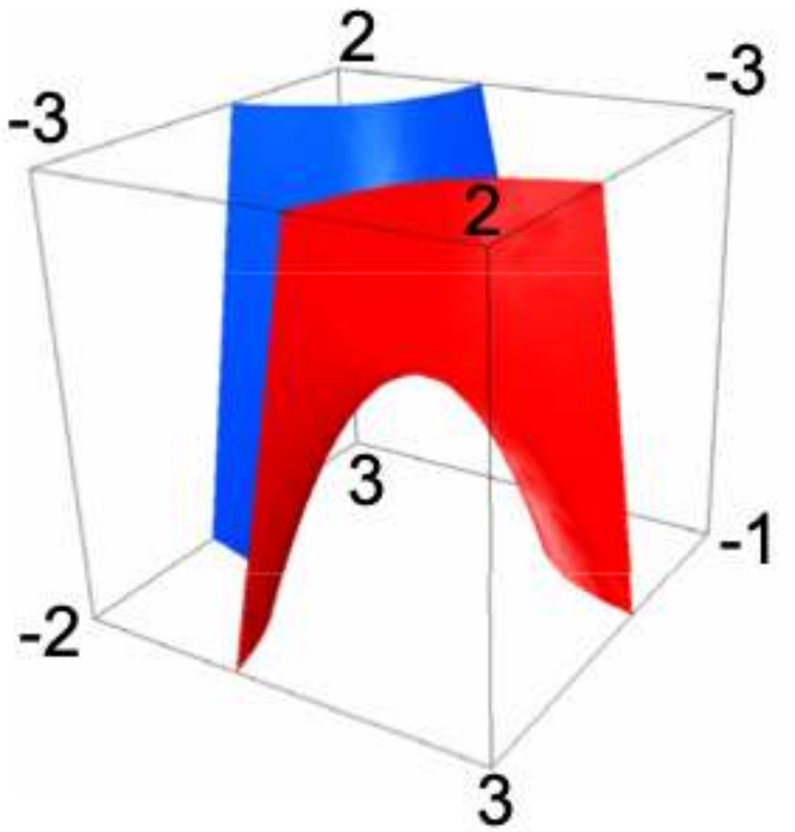
Difficult Cases for Every Isosurface Algorithm

- An ambiguous case in 2D:

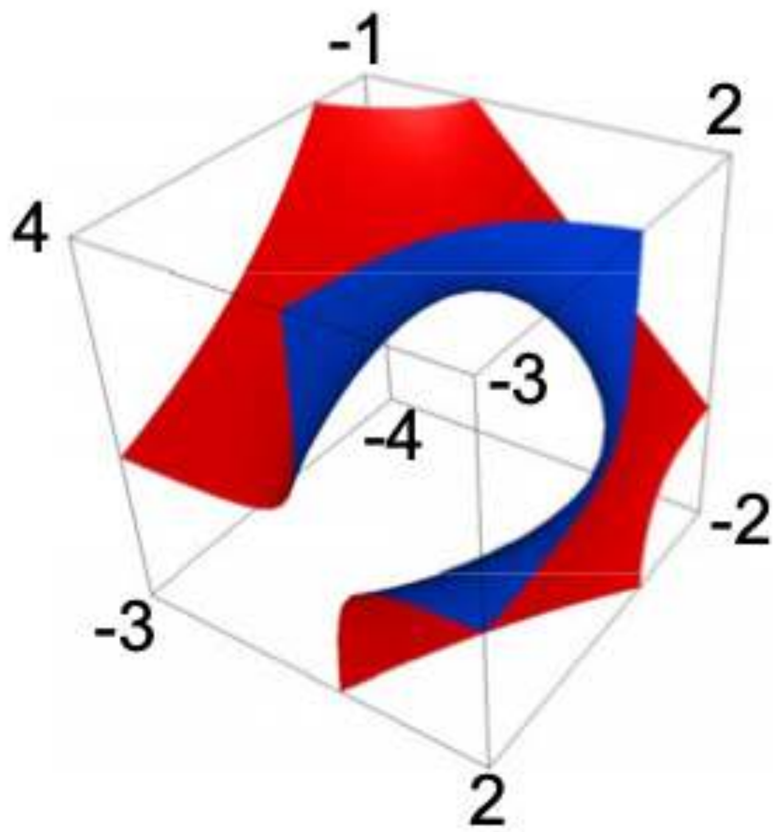


- Sometimes, triangulations of adjacent voxels won't match:
  - More on that ⟶ Advanced Computer Graphics

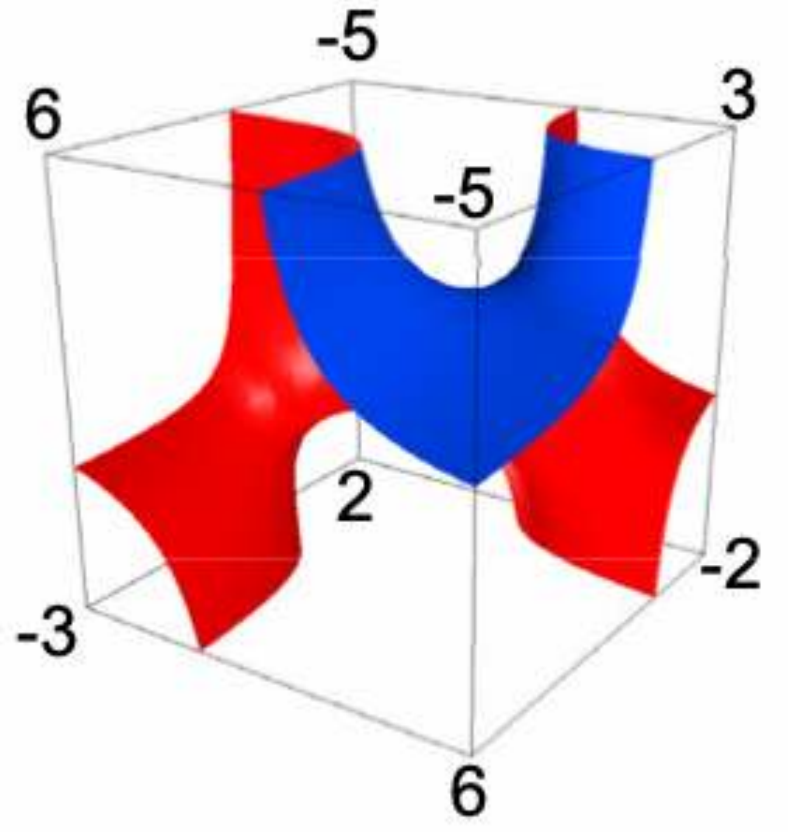




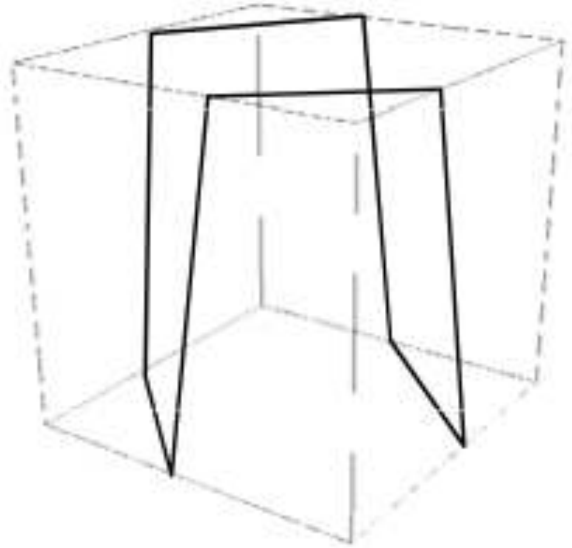
8-sided polygon



9-sided polygon



12-sided polygon

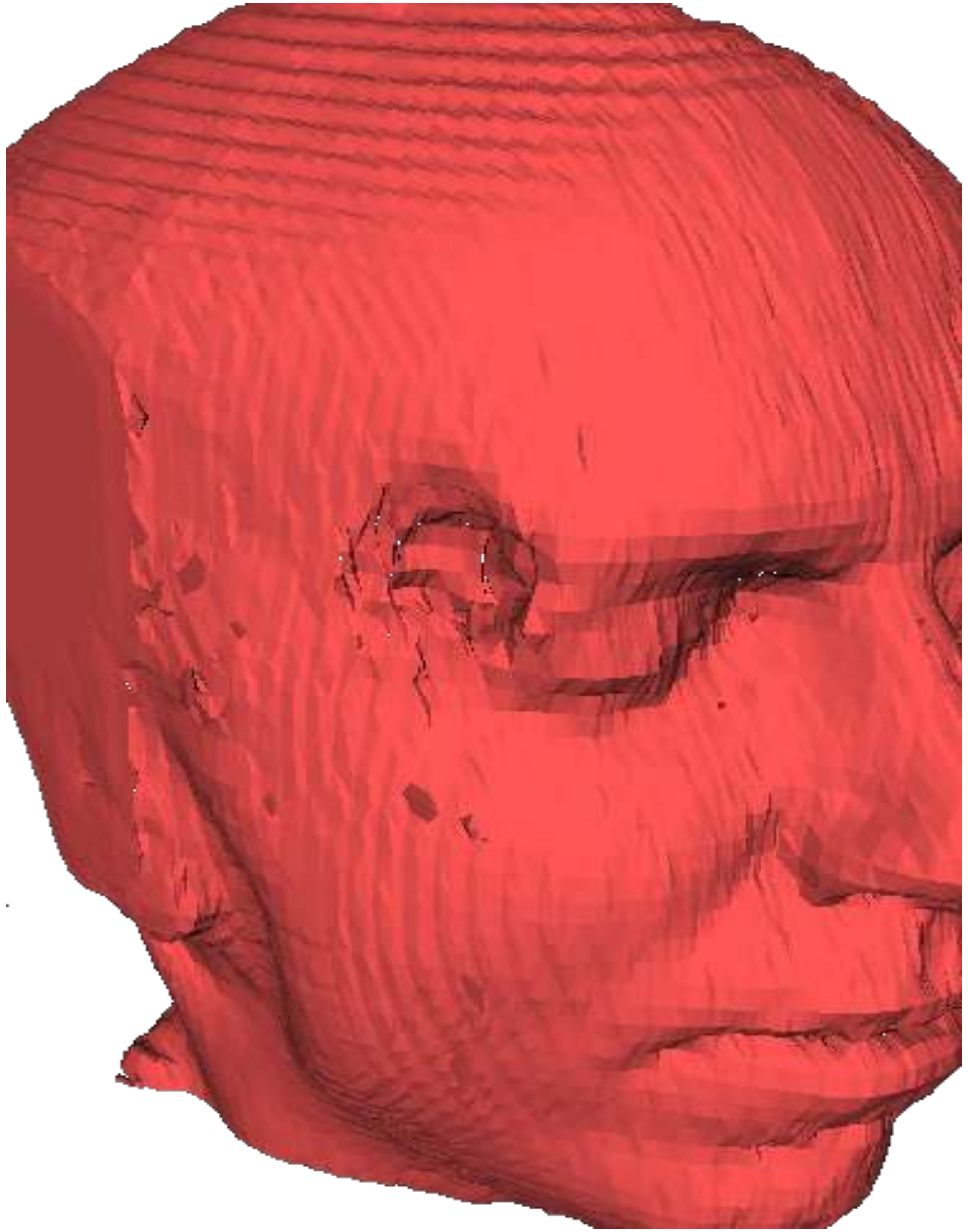


The 8-sided polygon has no valid triangulation!

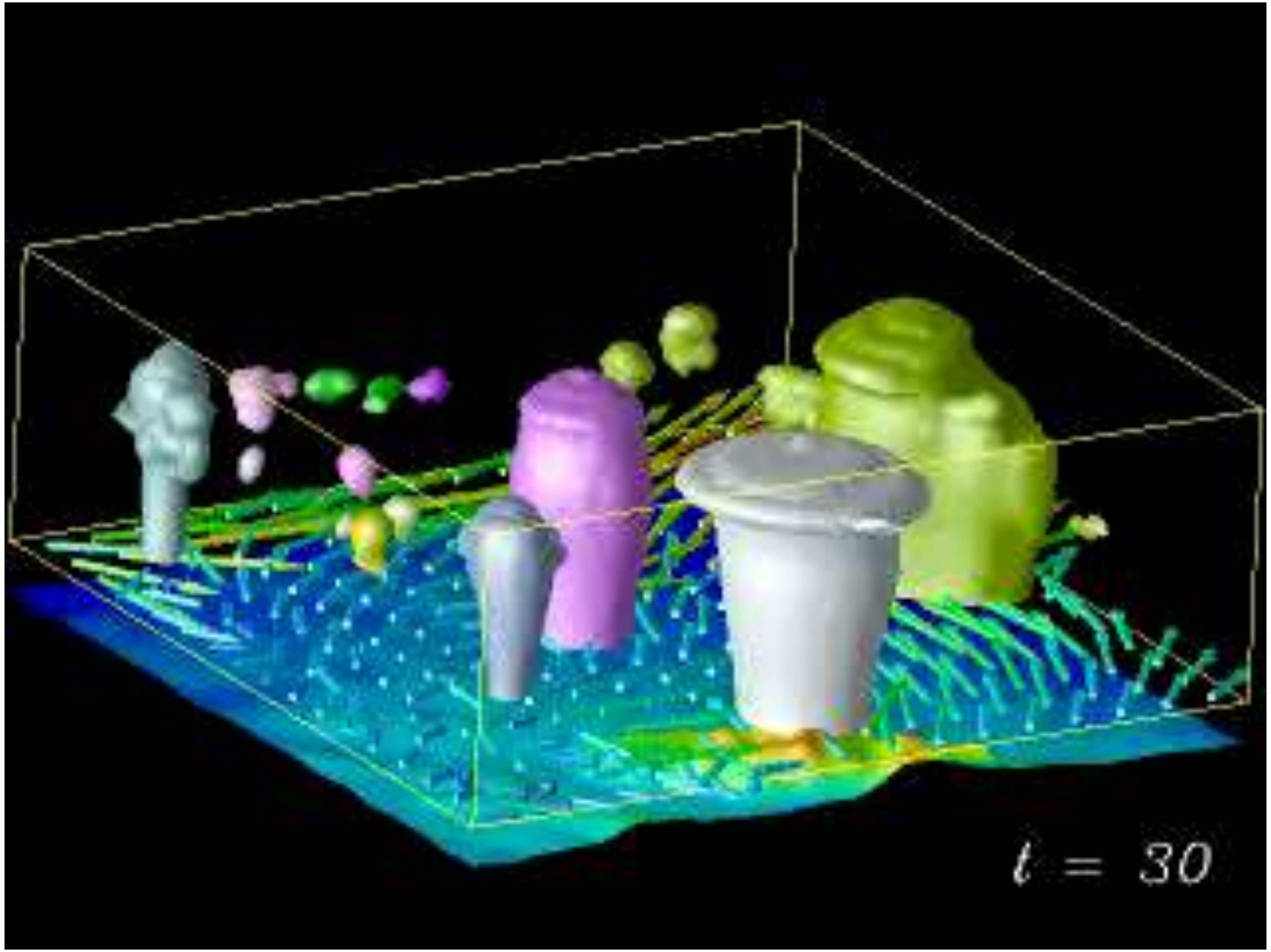
- either some triangles lie on faces of the cell
- or an extra vertex has to be used

~/avs/networks/SciVis/AD*net

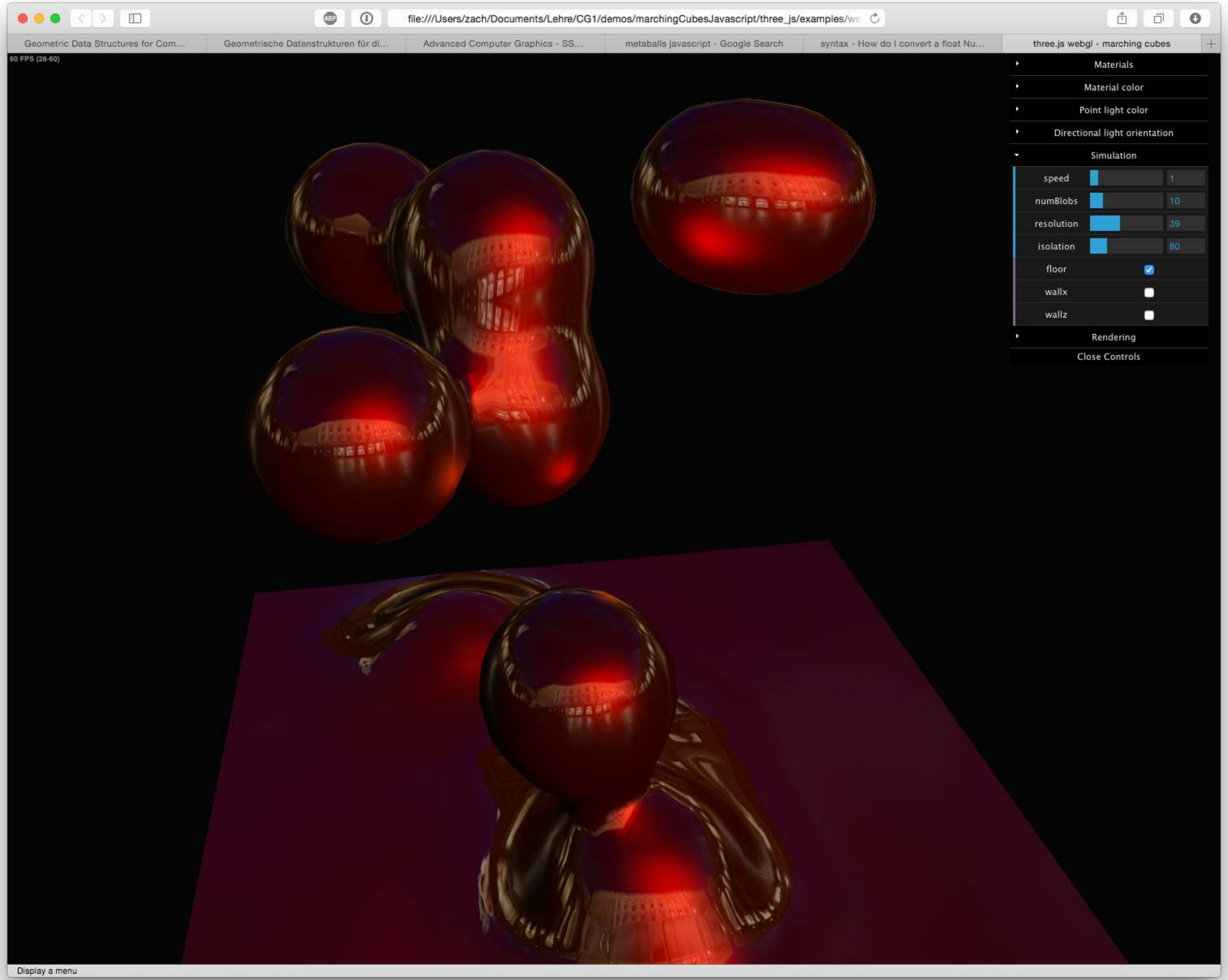
- Output of a single Marching-Cube-Algorithm:



Example time-varying volume data set



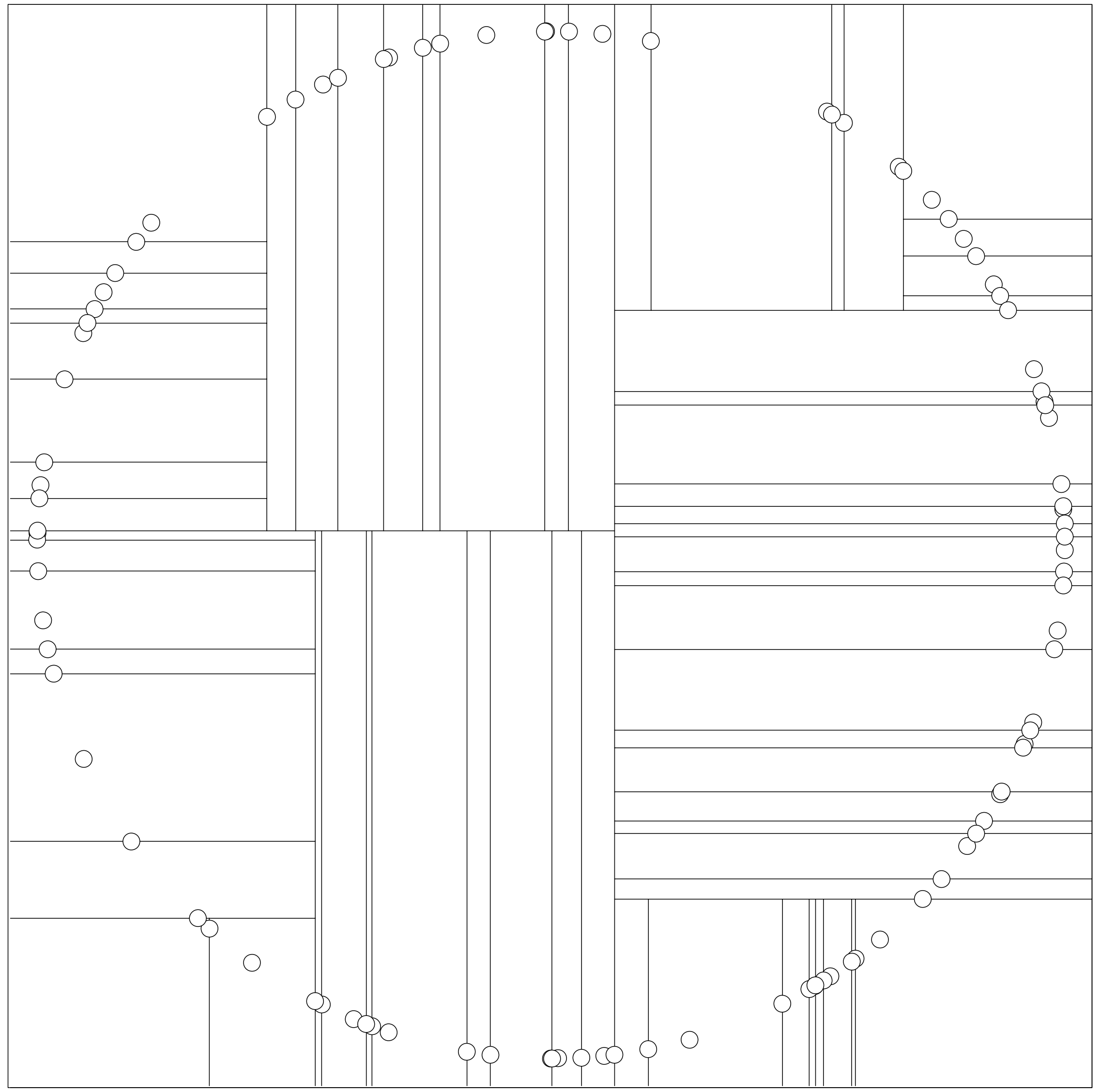
Another Demo (Metaballs)



http://threejs.org/

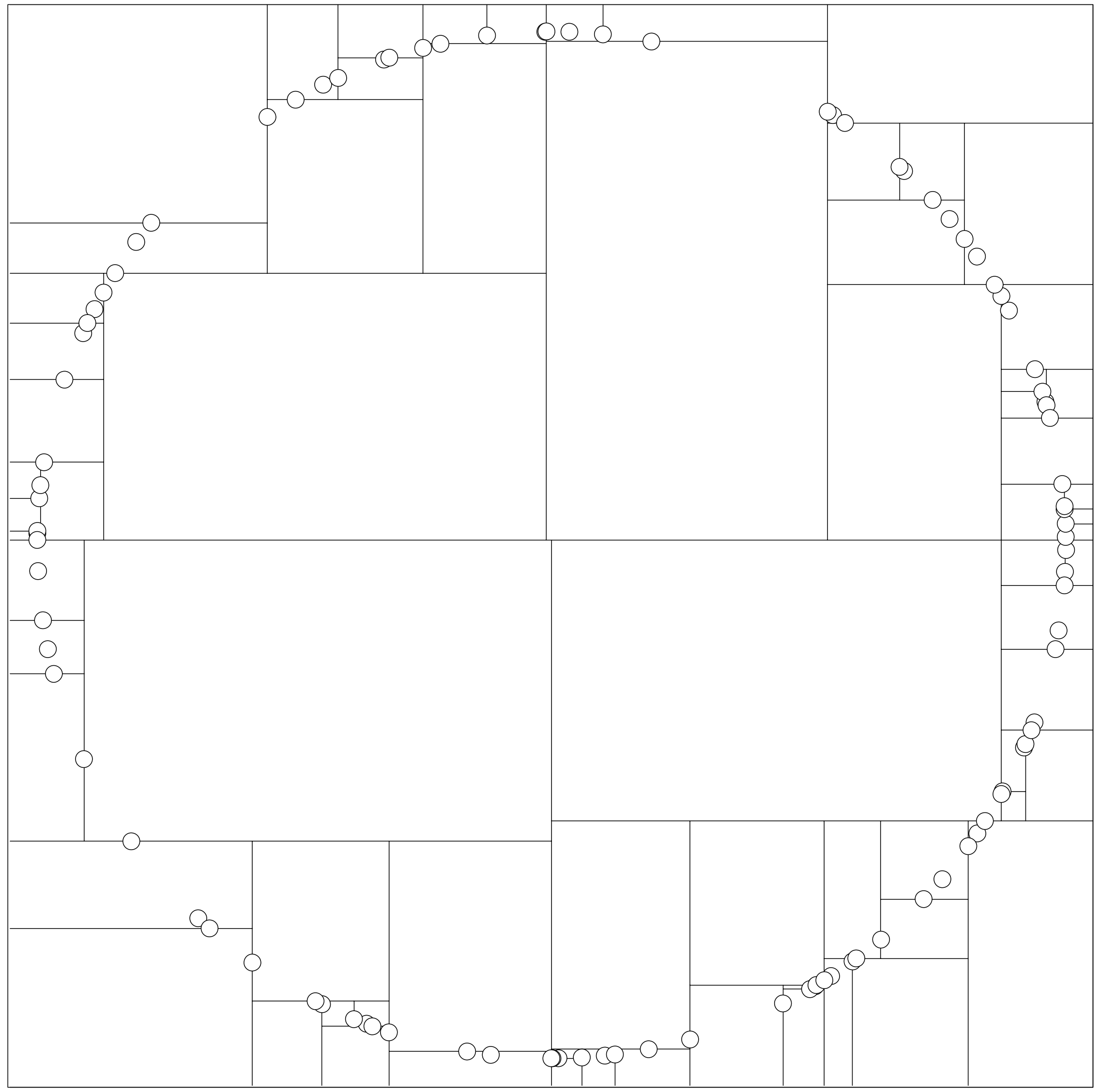
Splitting strategies when building kd-Trees

Widest spread kd-tree



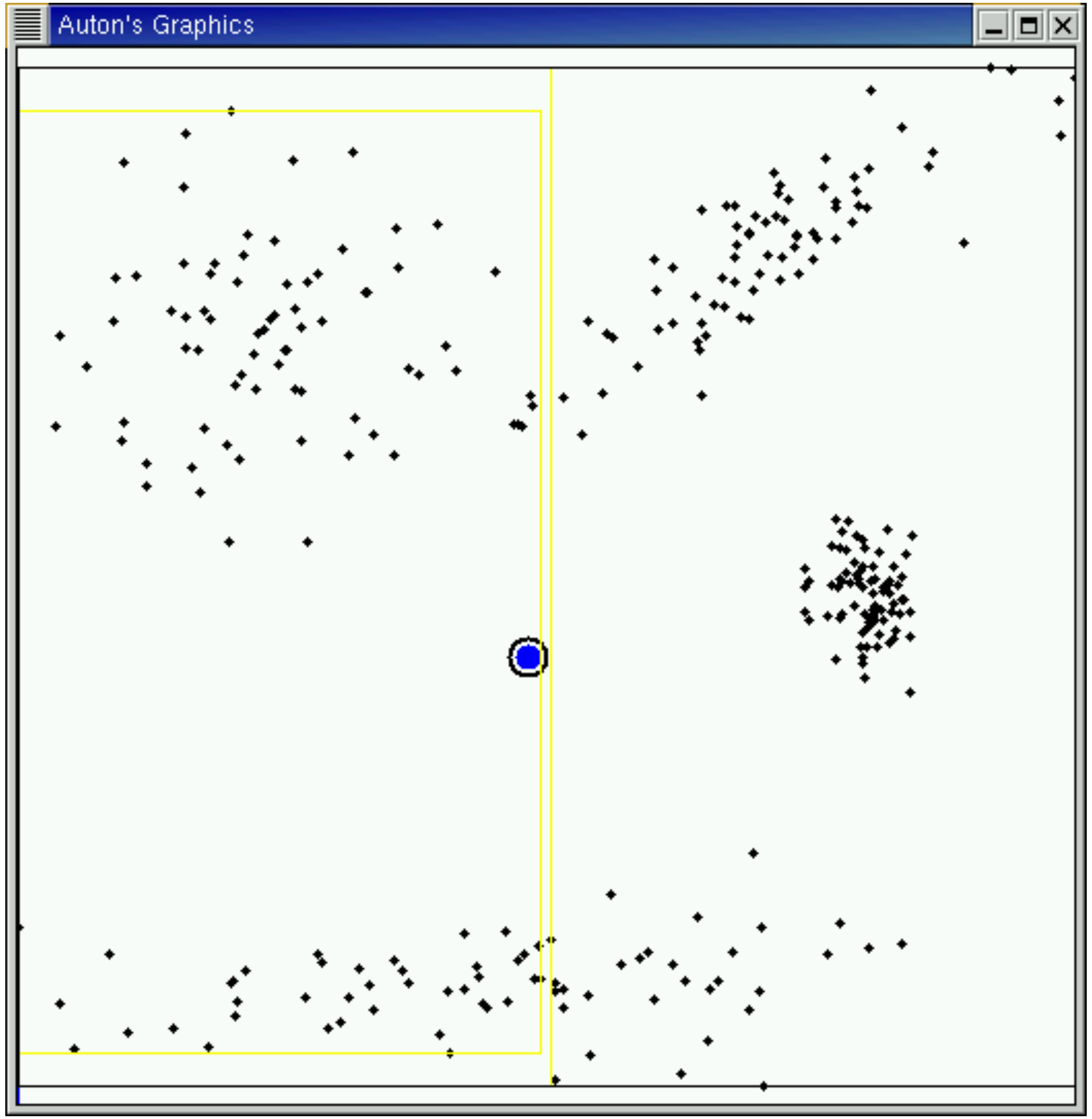
Along the dimension
with the widest spread of the points,
at the median of the coords

Longest side kd-tree

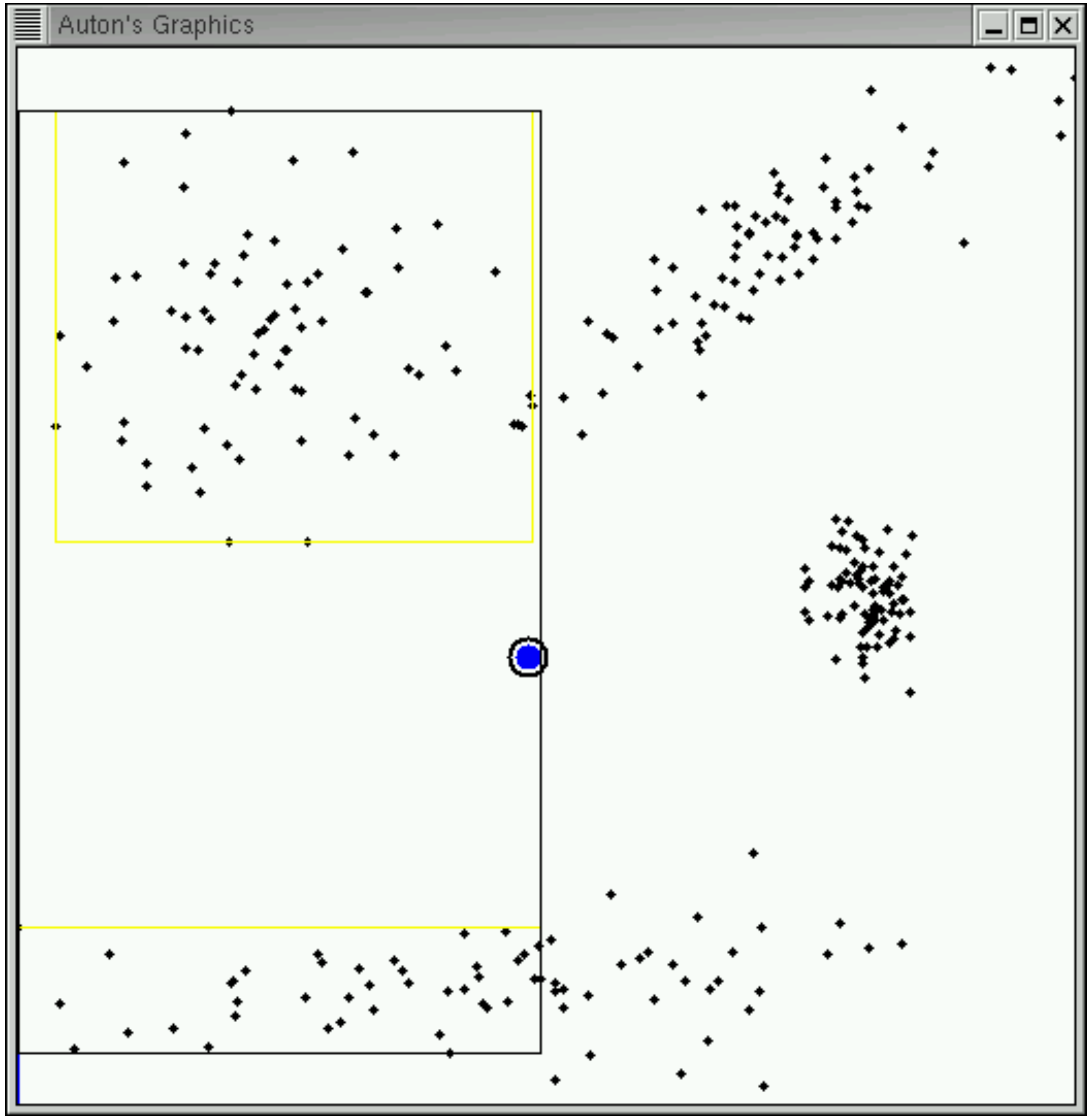


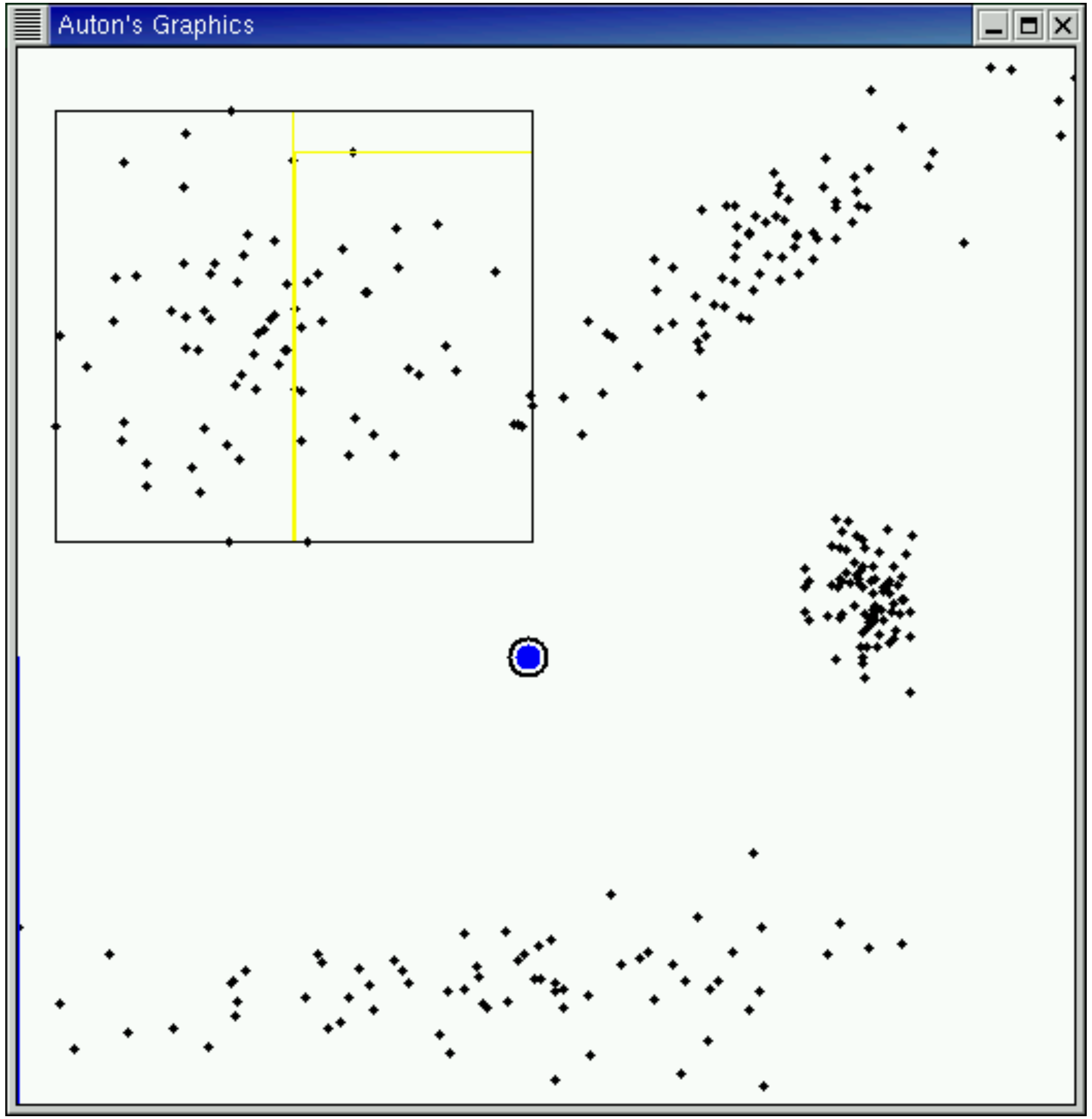
Along the dimension with the
longest side of the region,
at the coordinate closest to the
middle of the extent of the region

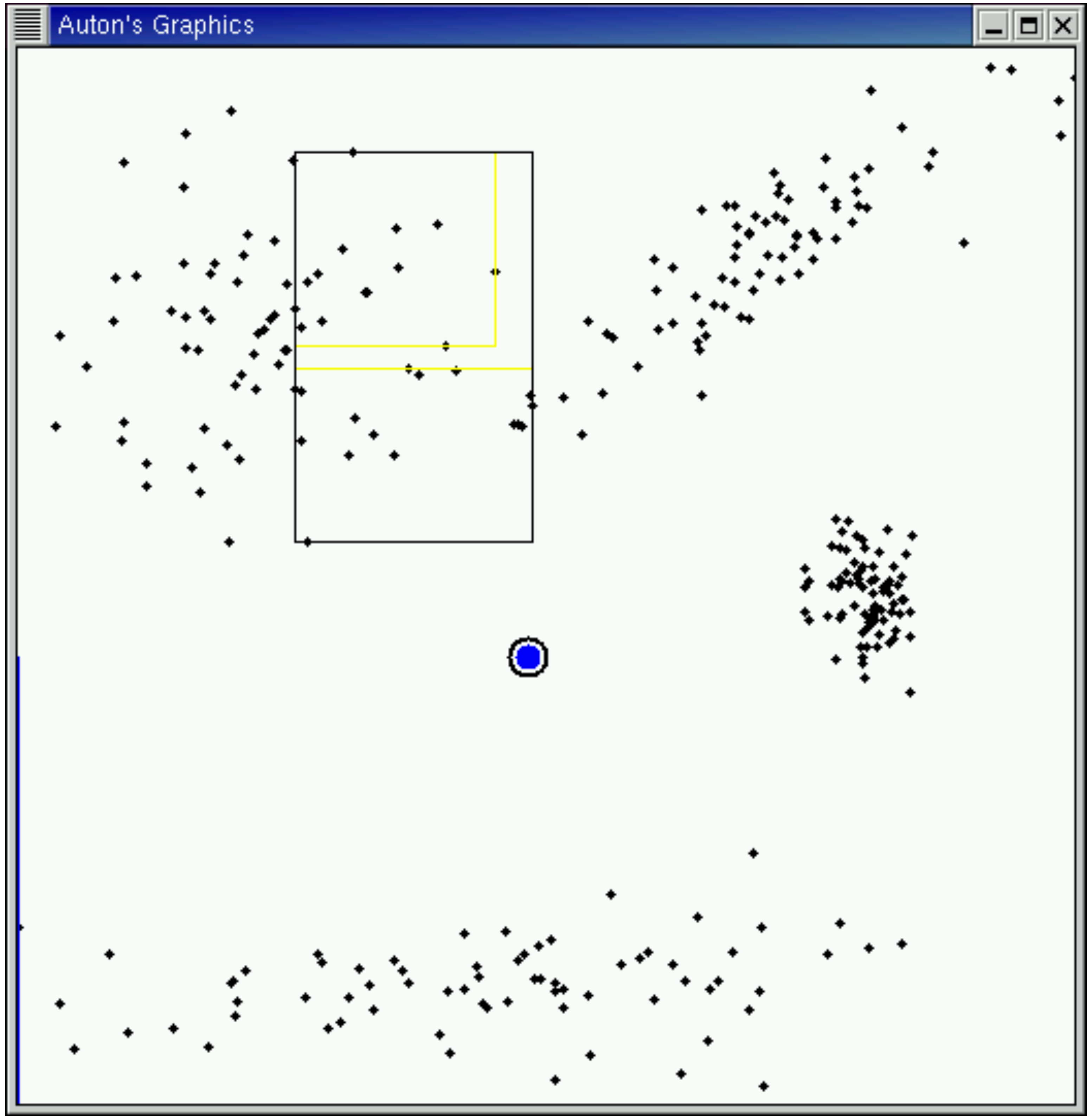
Animation of Nearest-Neighbor using kd-Trees

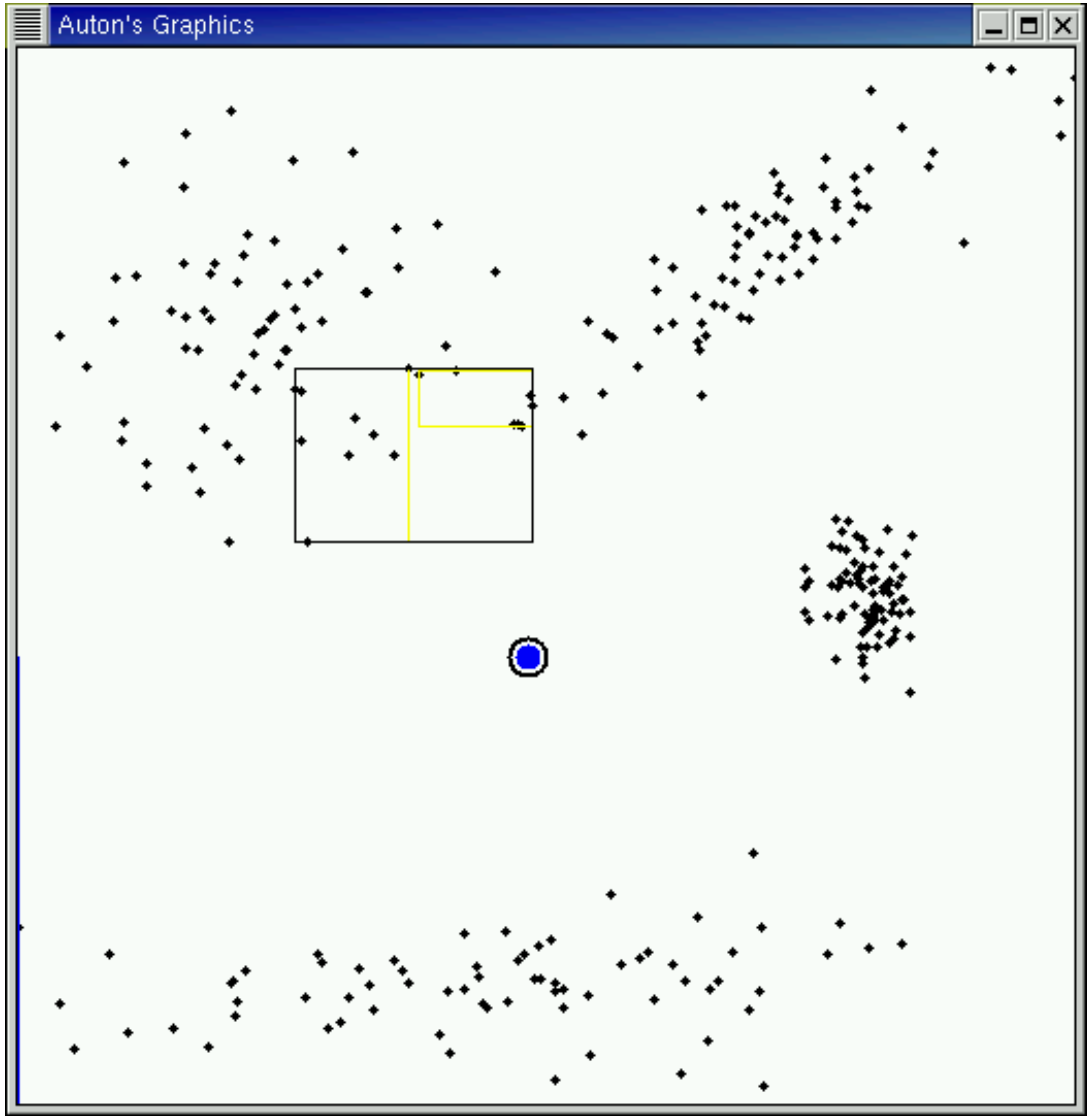


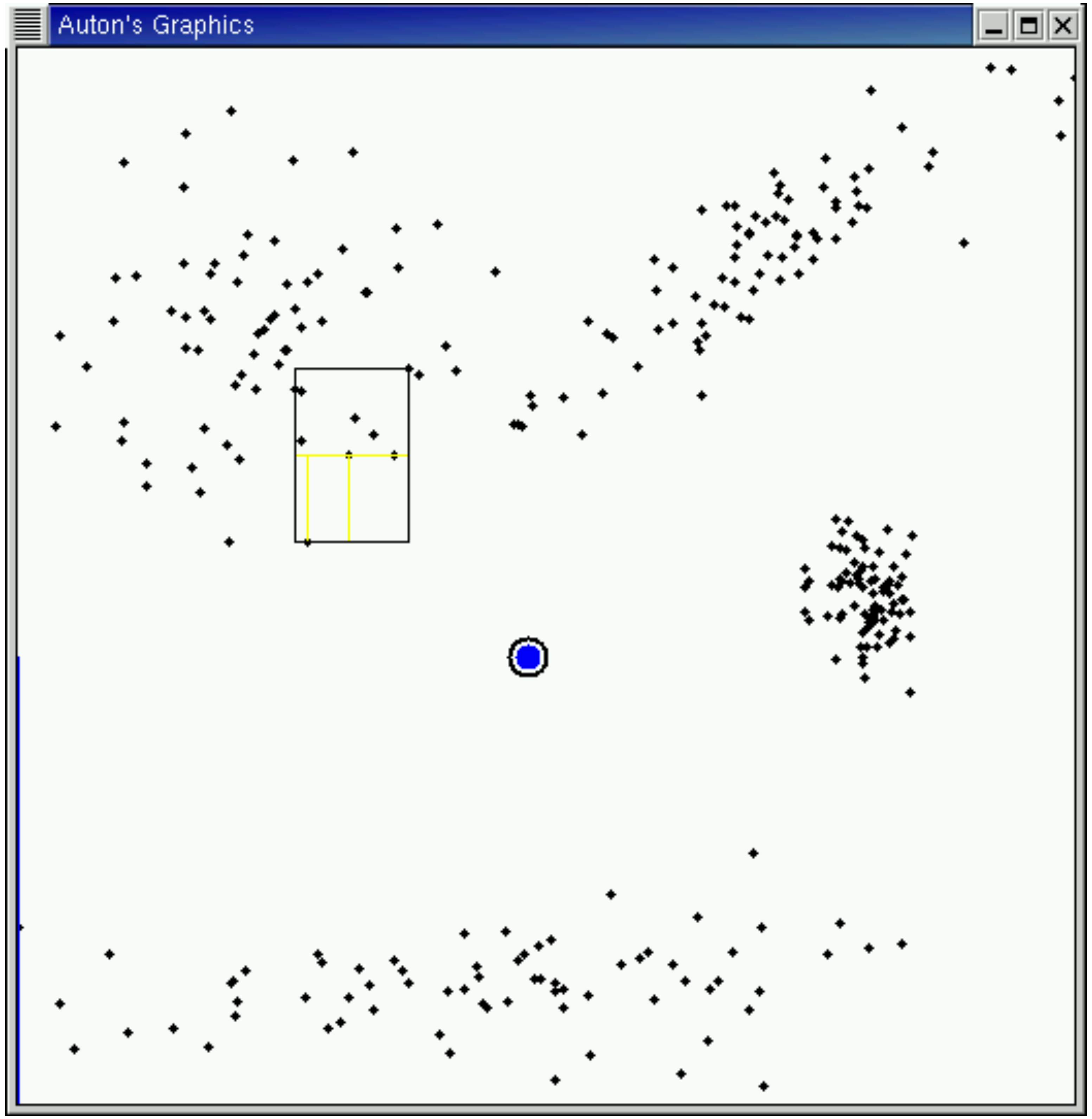
Andrew Moore, CMU

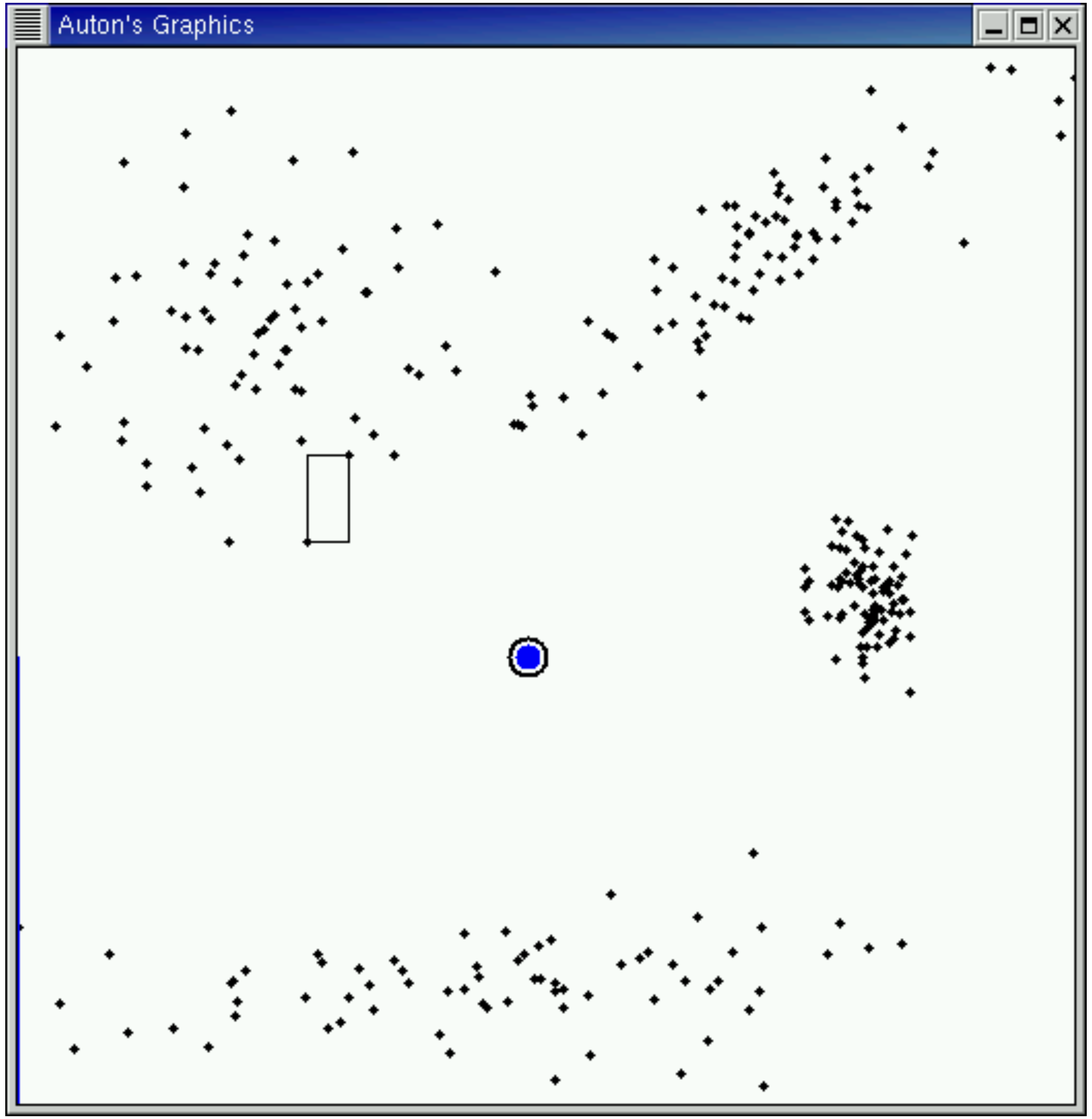

Auton's Graphics

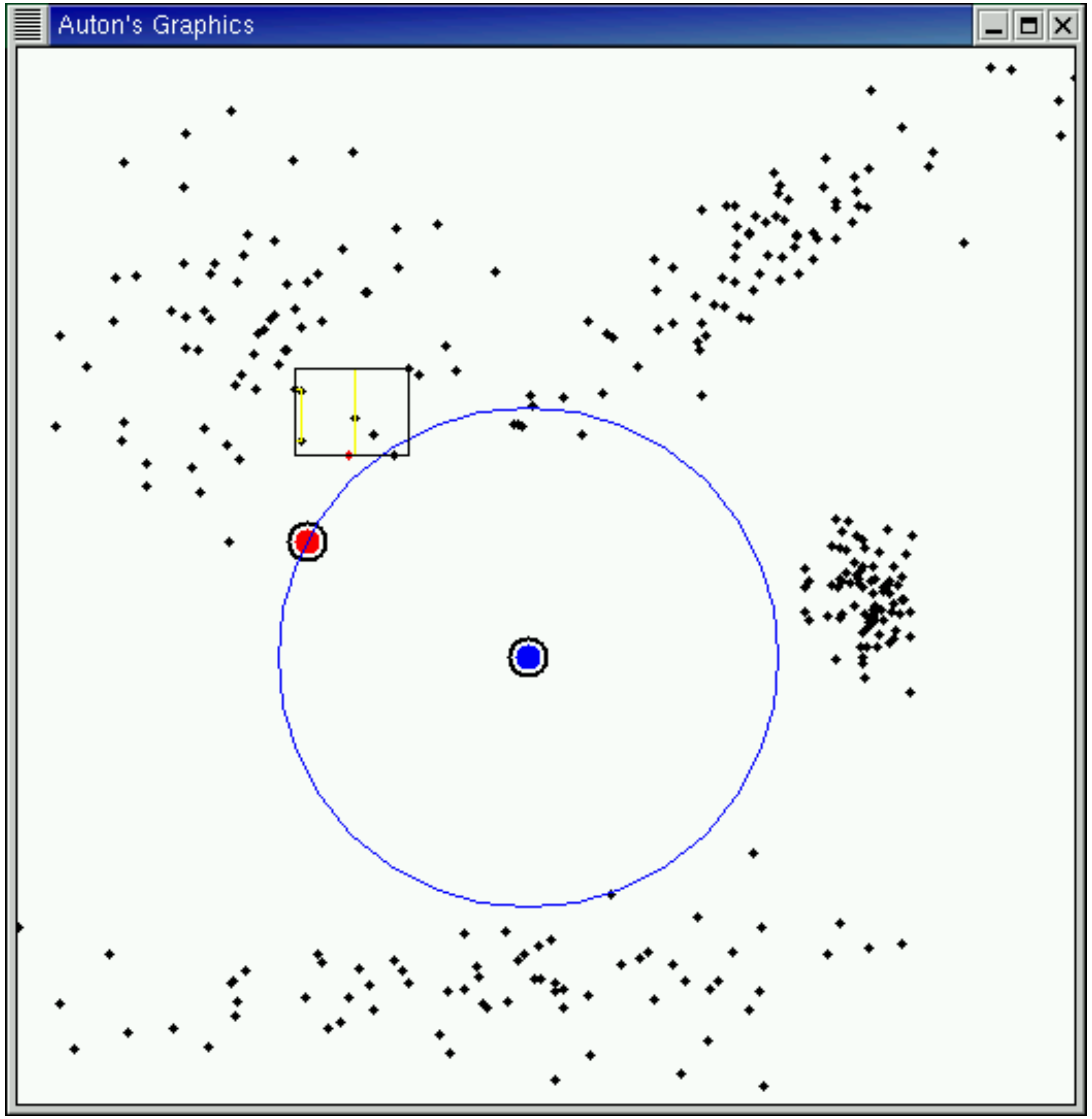

Auton's Graphics

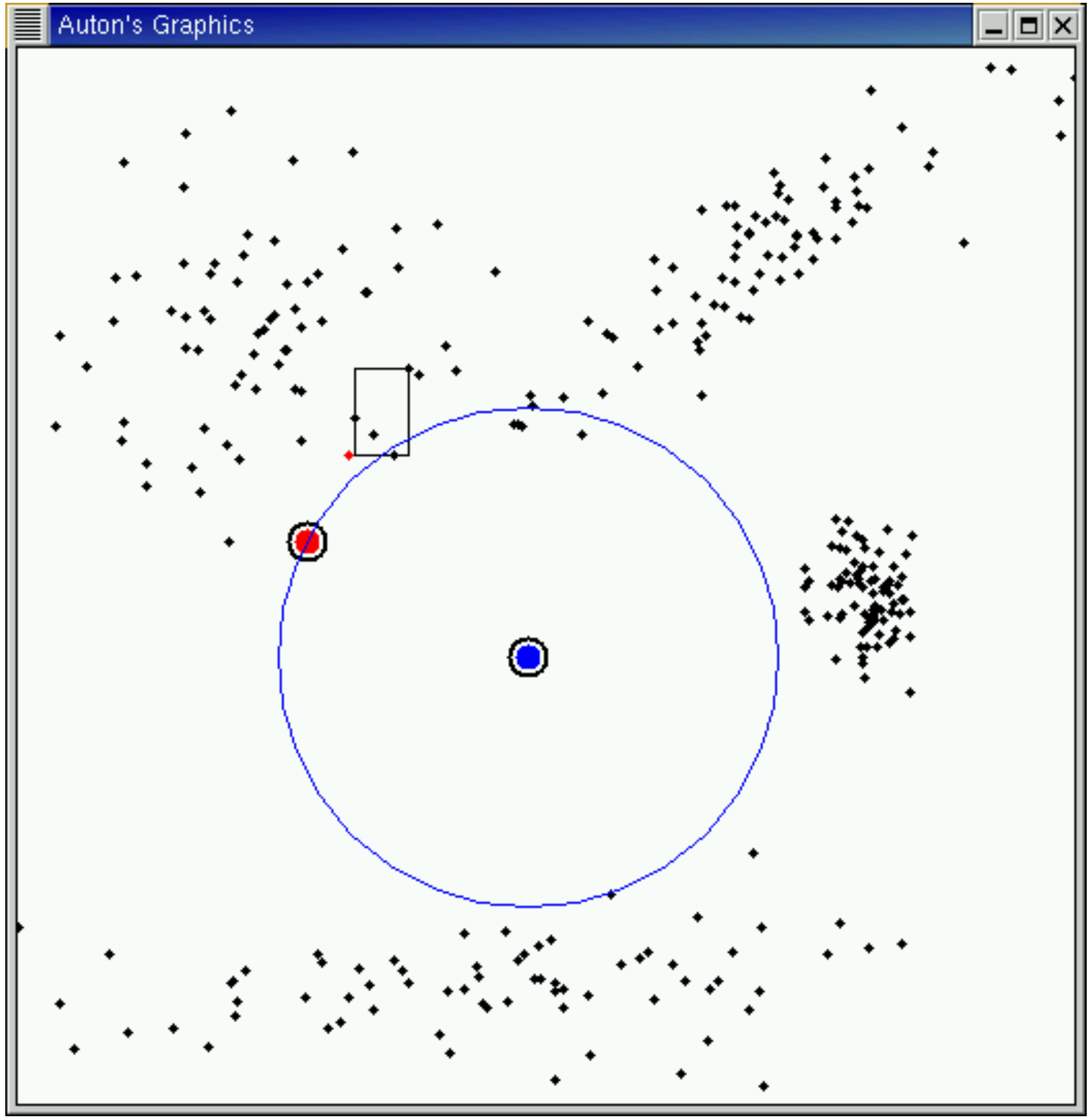

Auton's Graphics

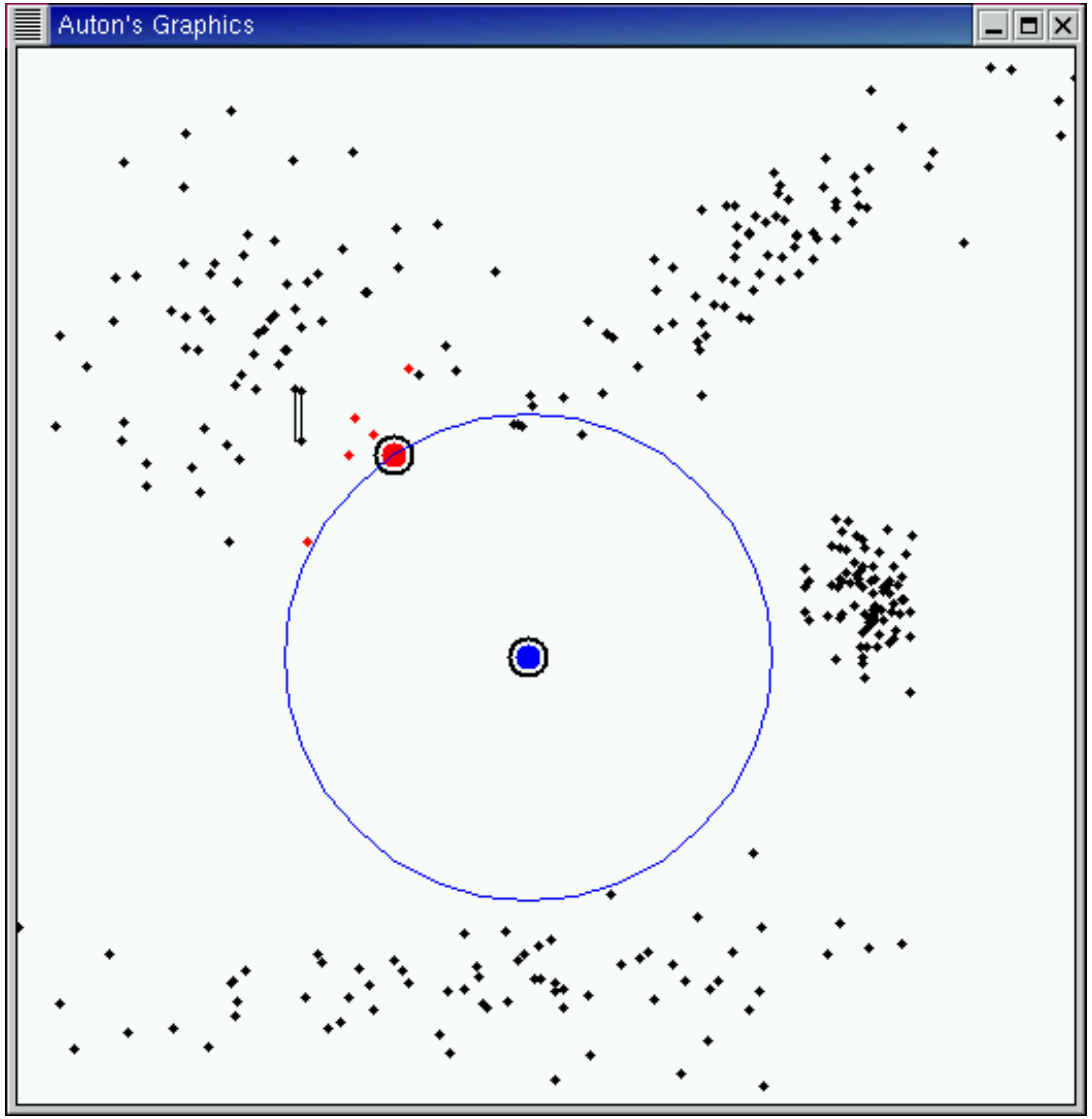

Auton's Graphics

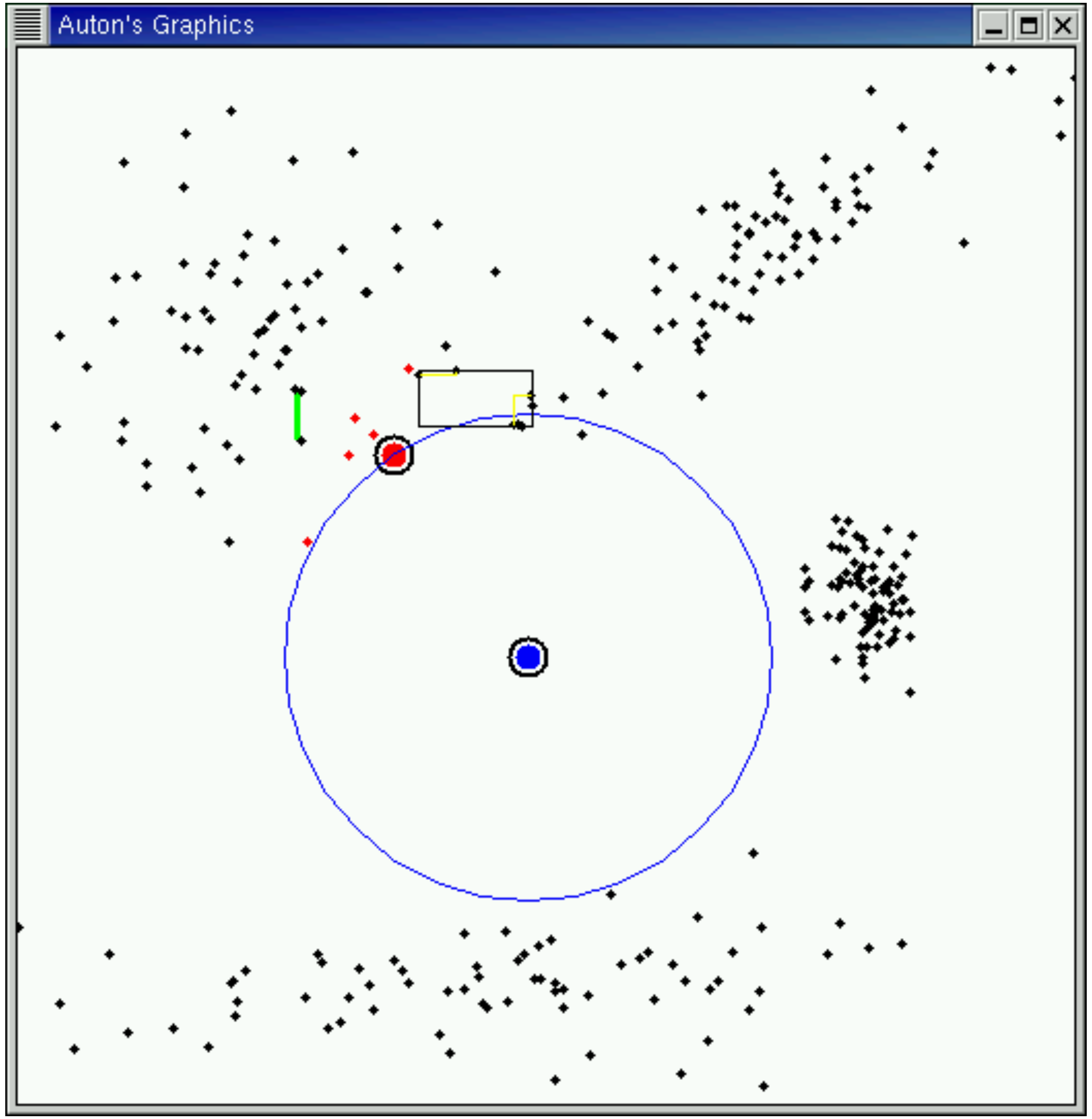

Auton's Graphics

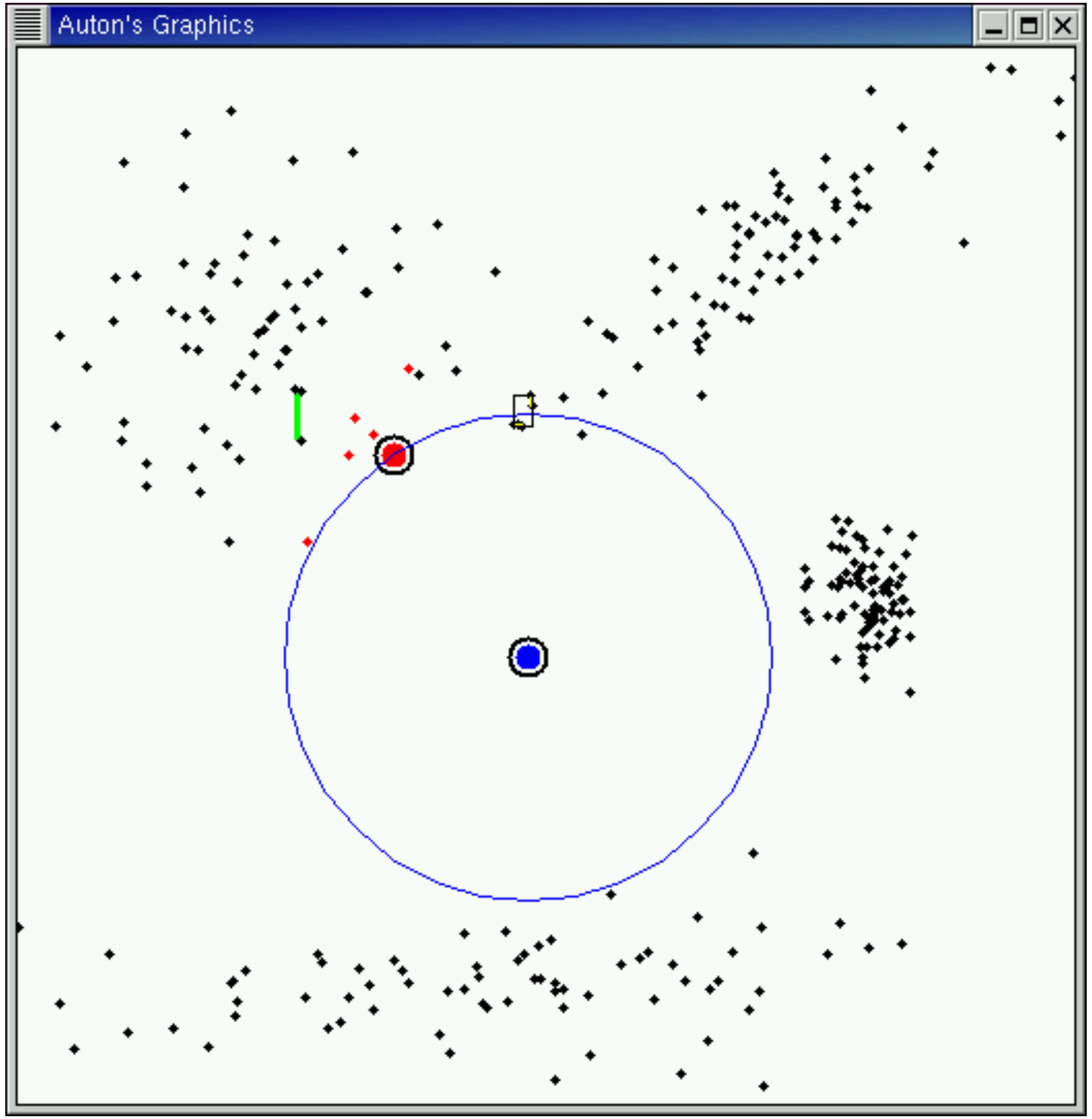

Auton's Graphics

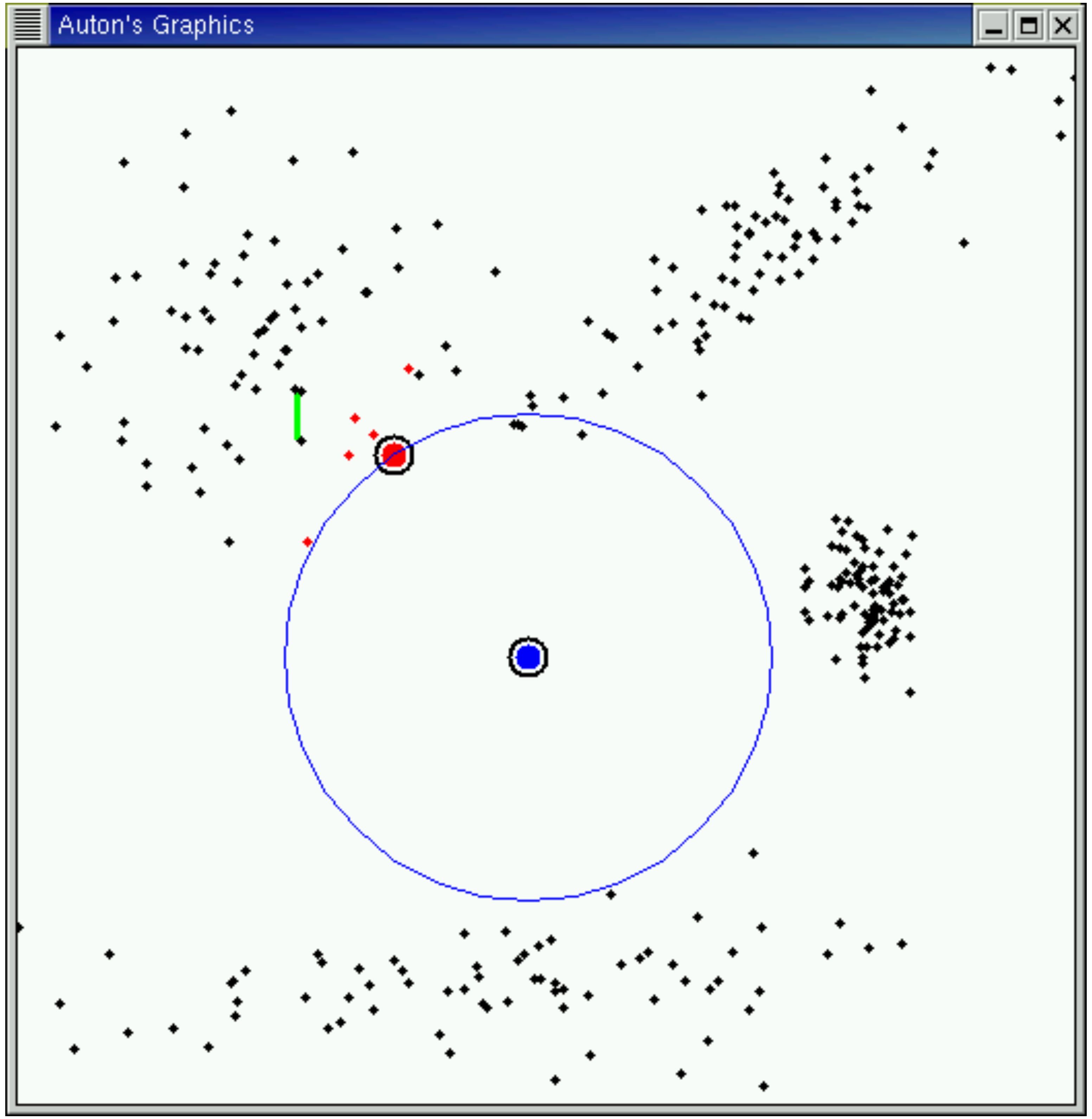

Auton's Graphics

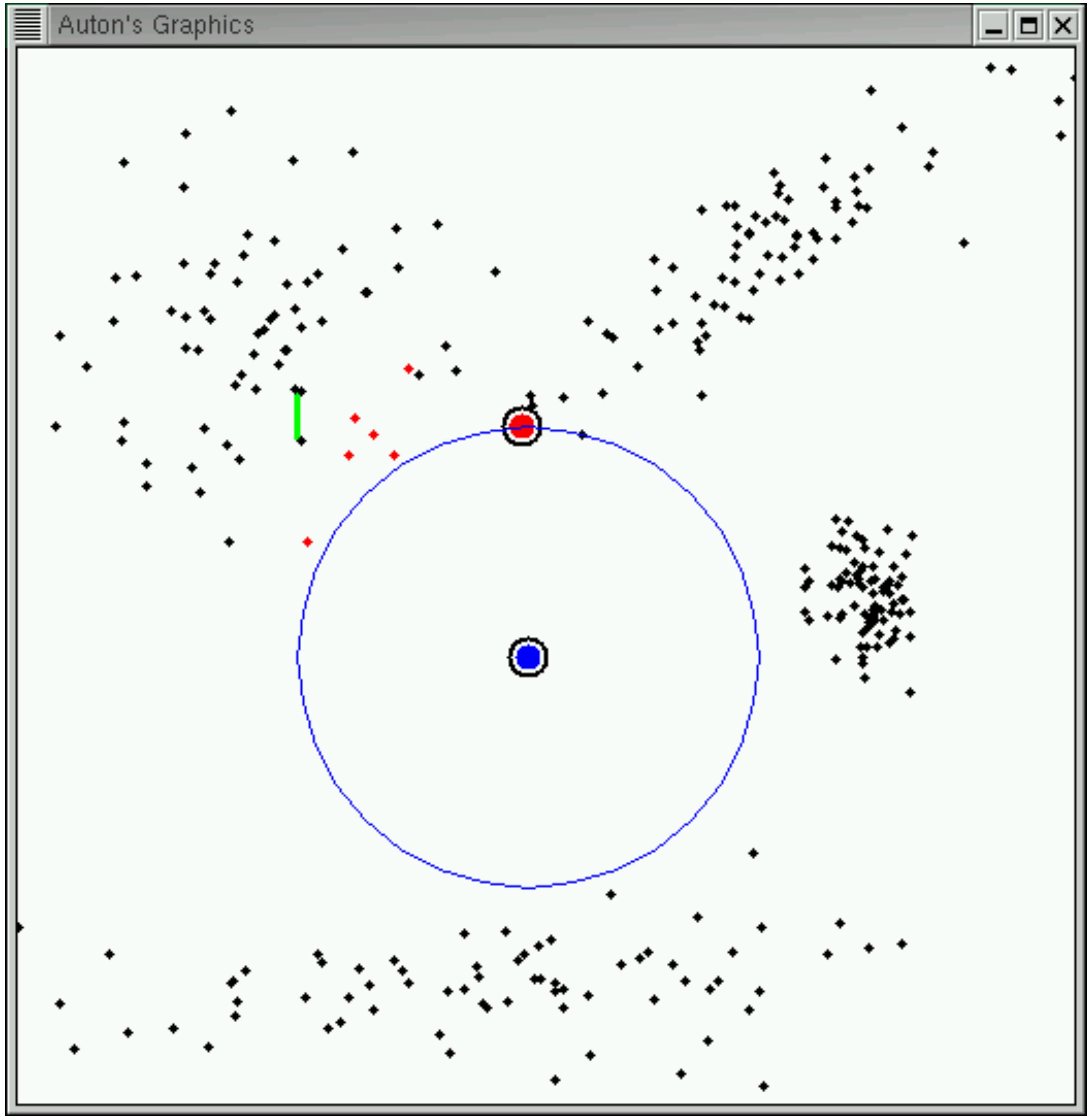

Auton's Graphics

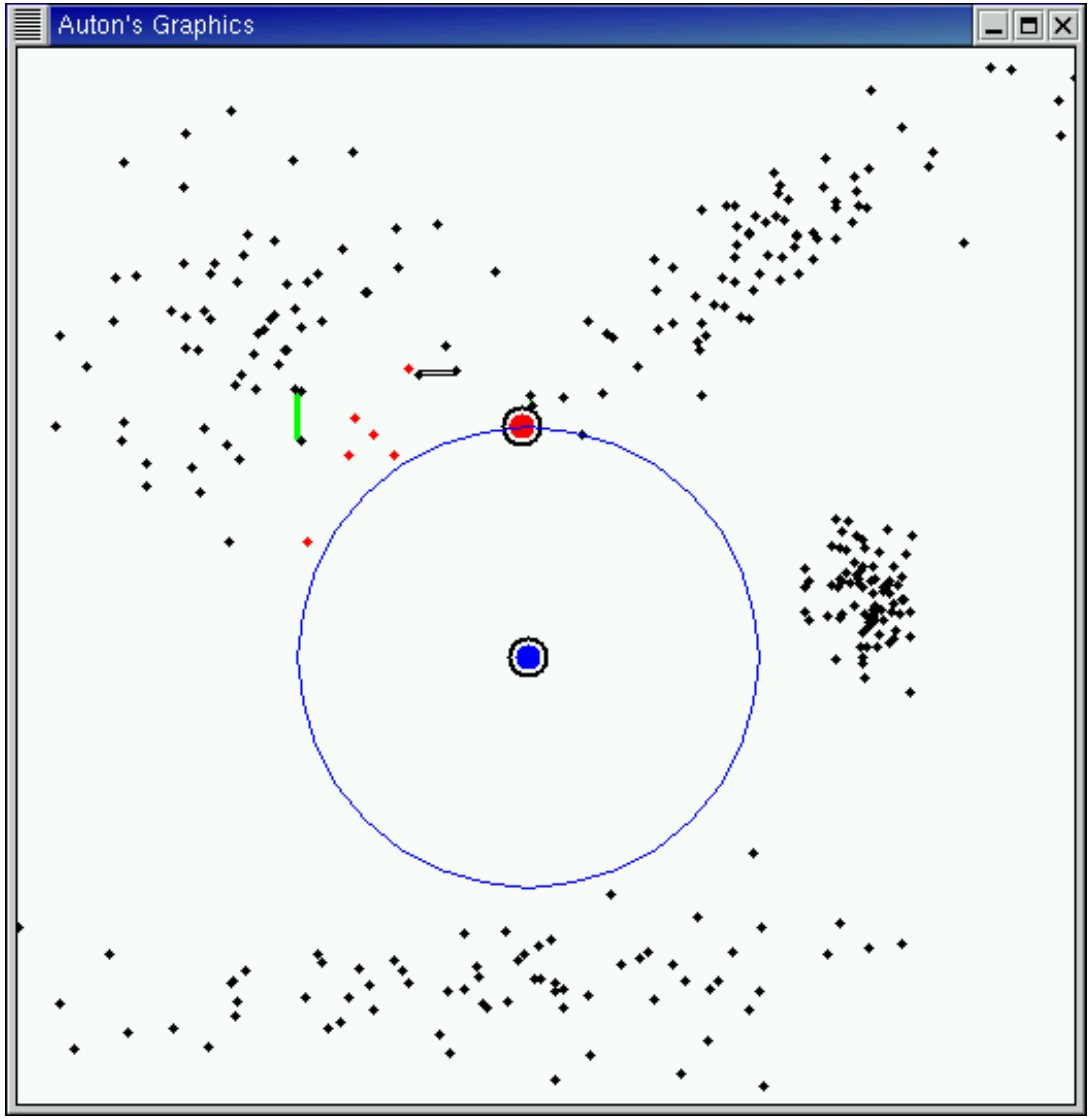

Auton's Graphics

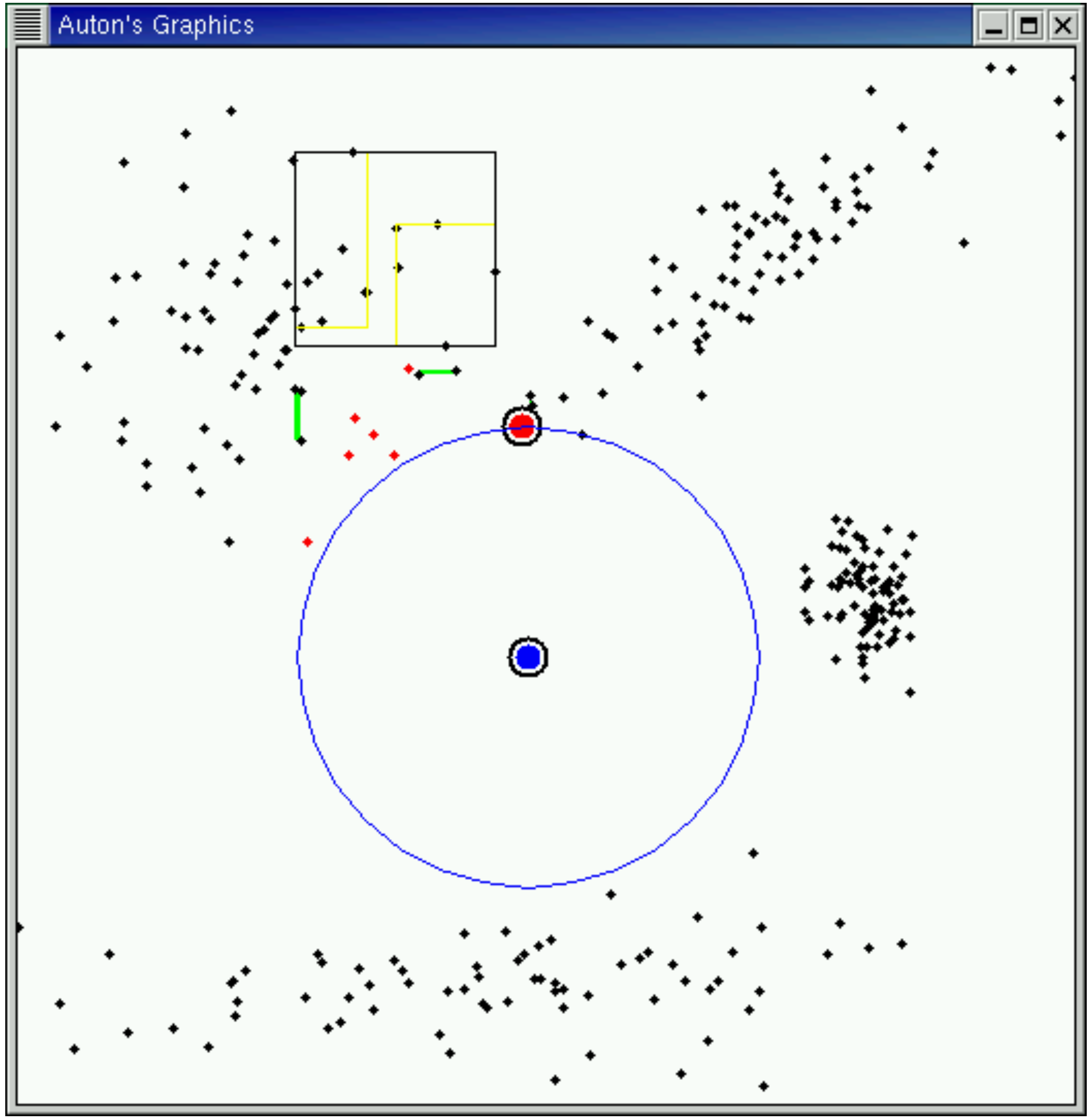

Auton's Graphics

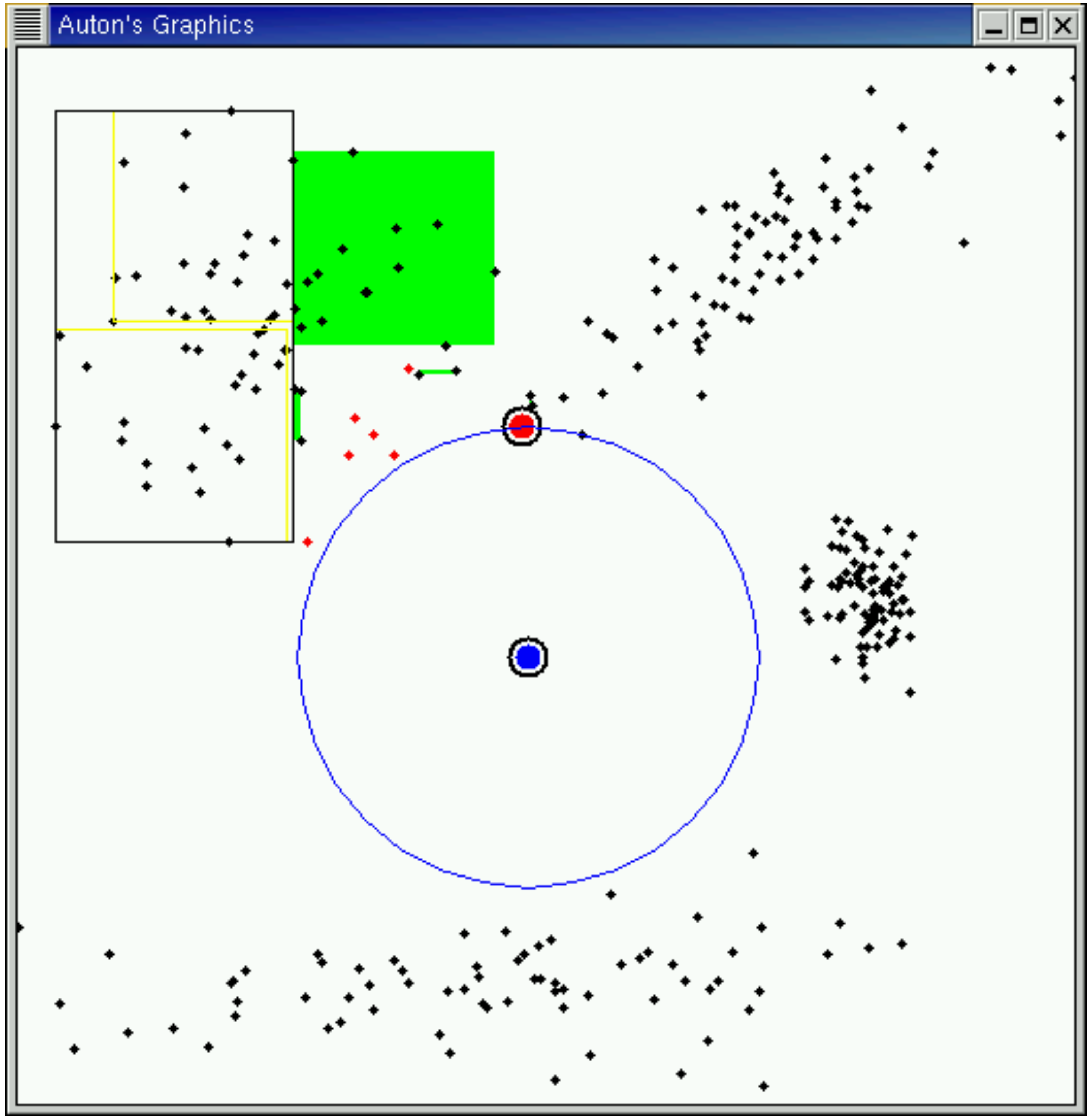

Auton's Graphics

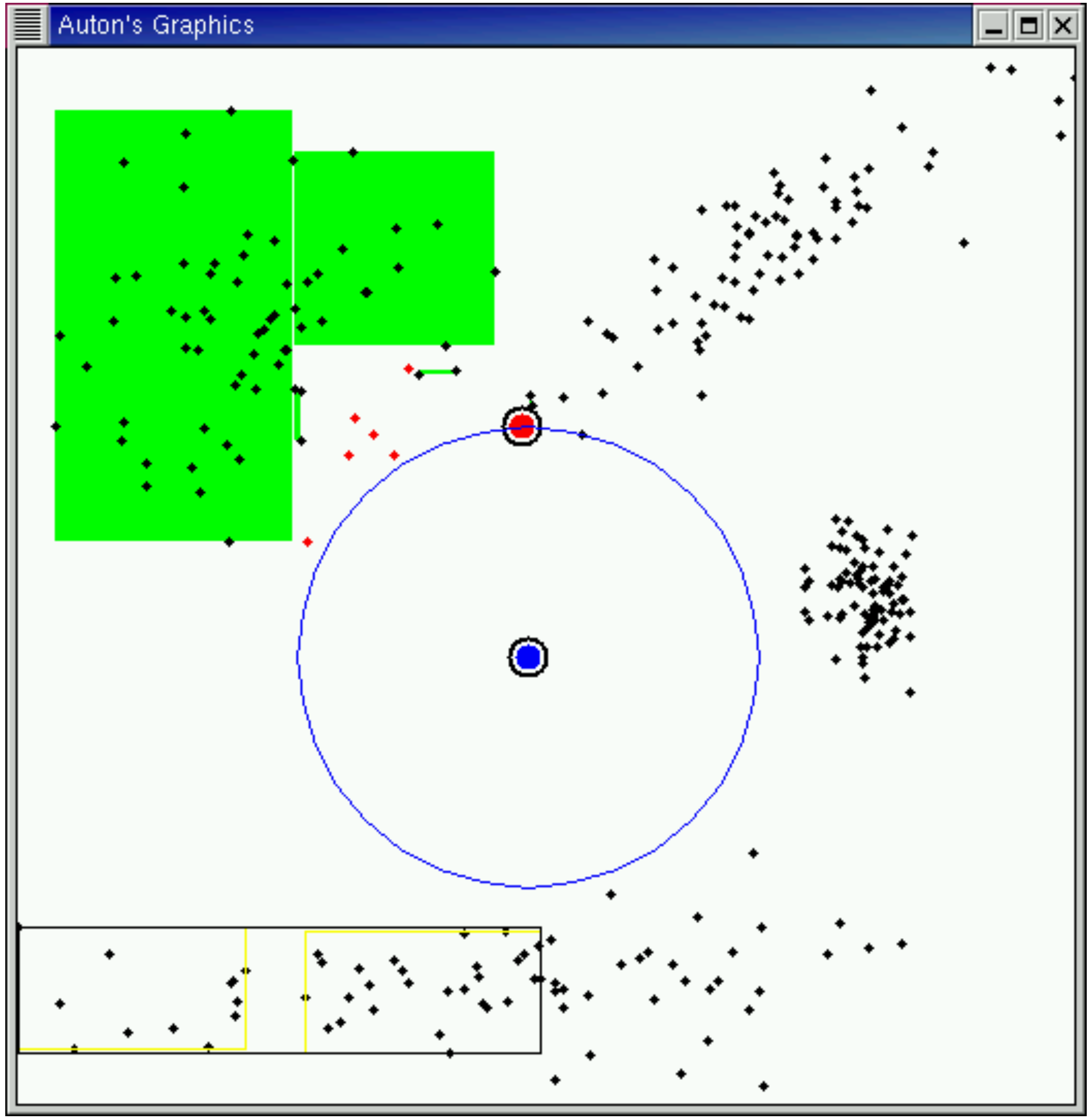

Auton's Graphics

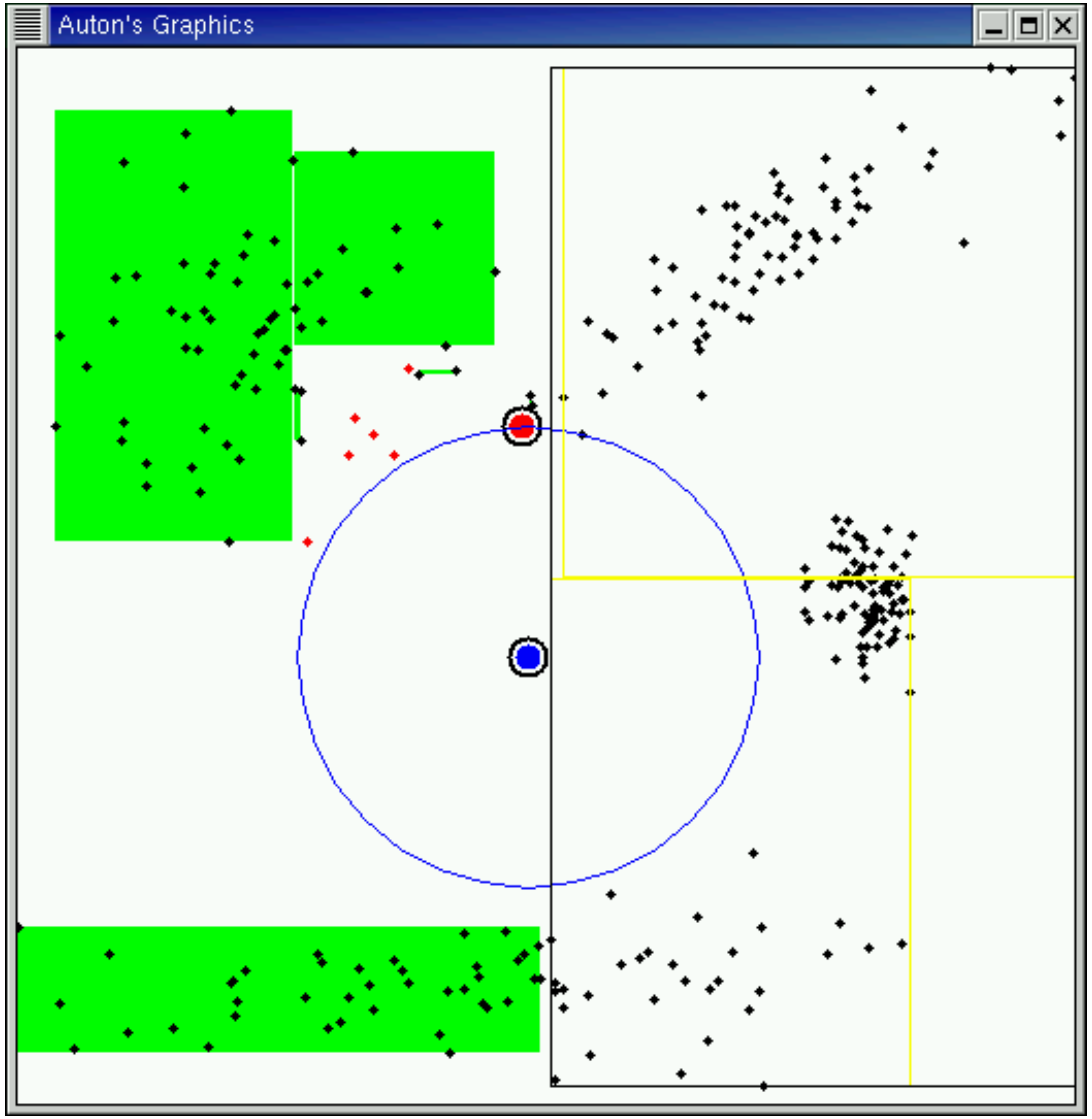

Auton's Graphics

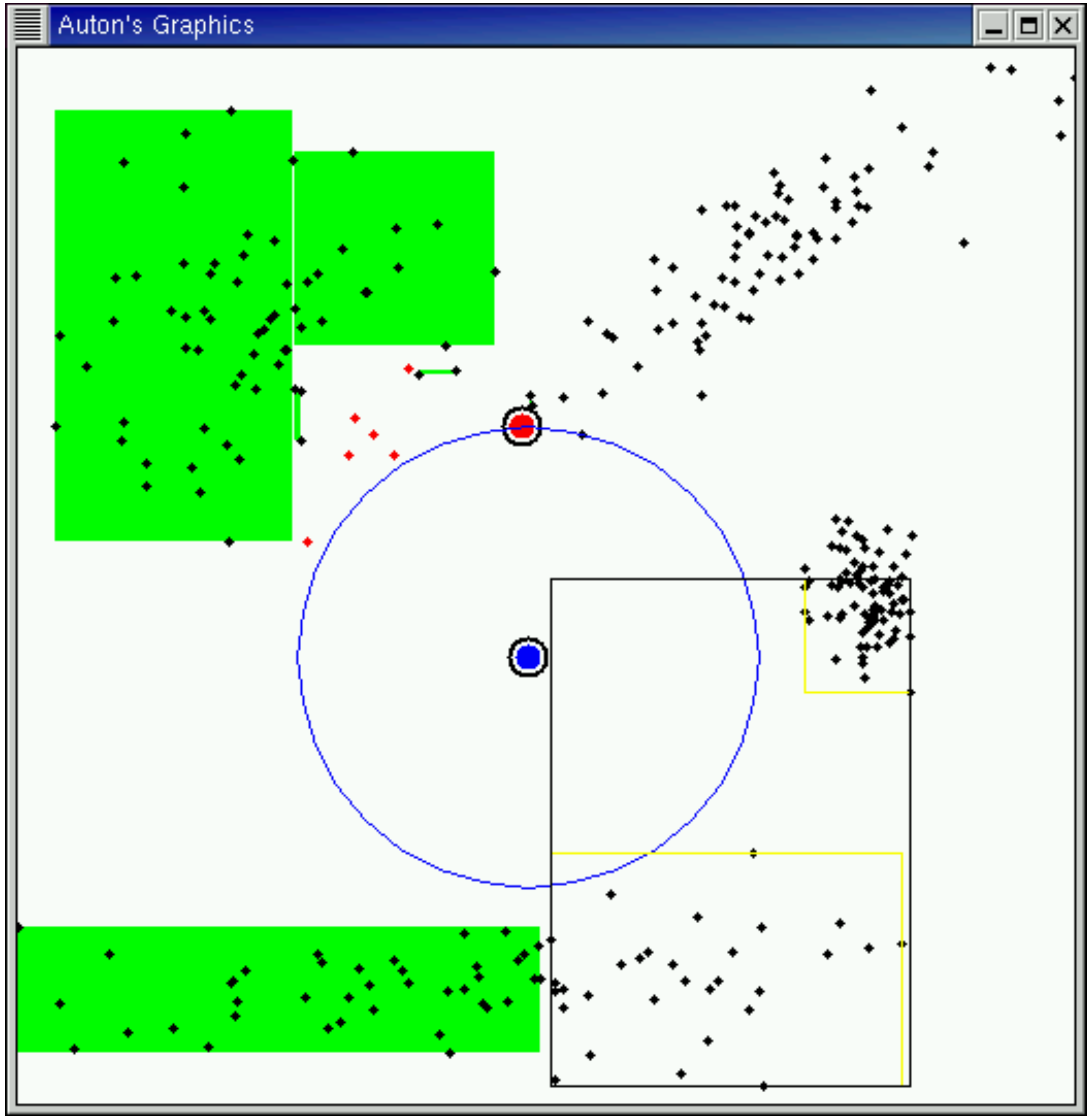

Auton's Graphics

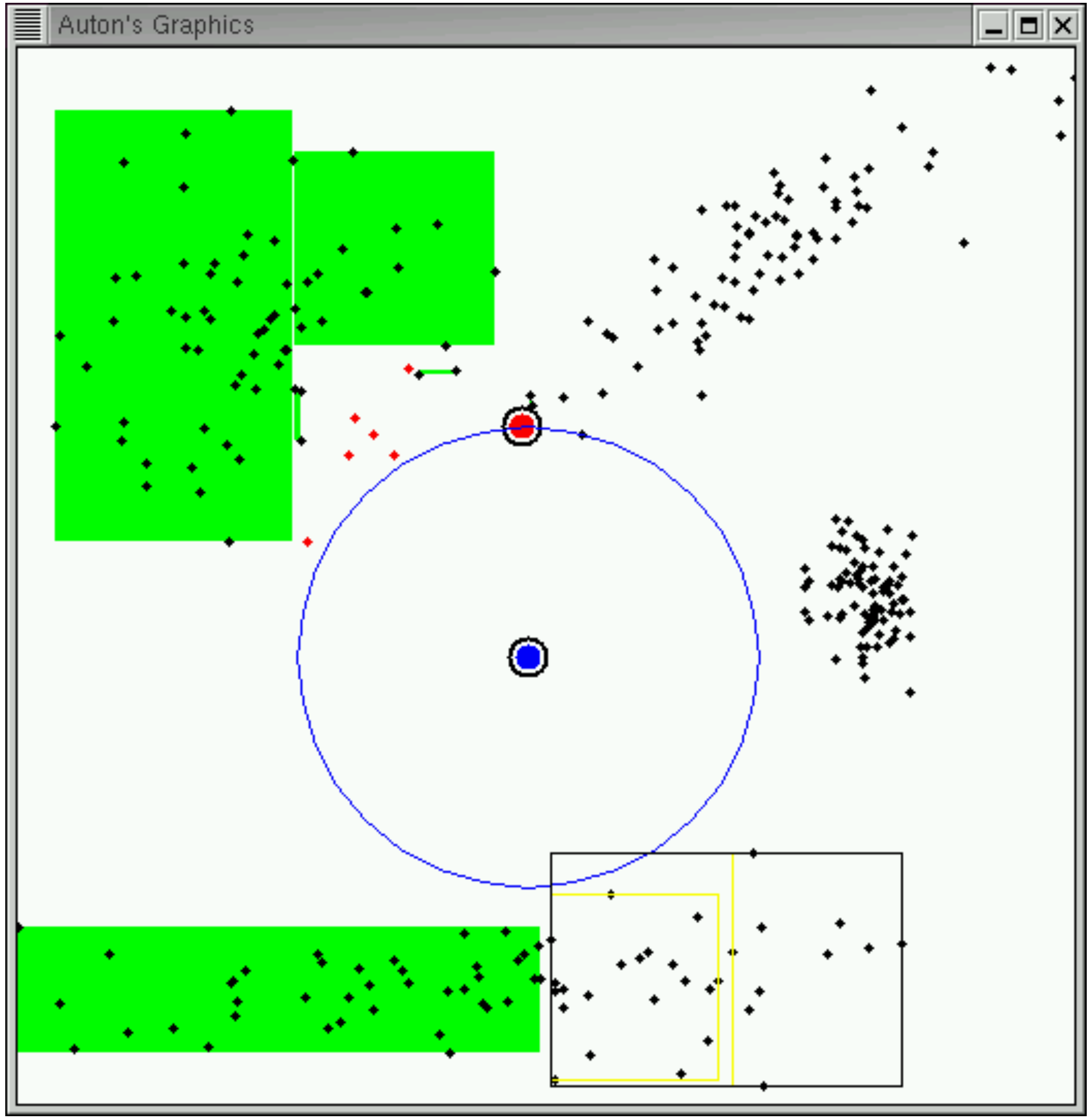

Auton's Graphics

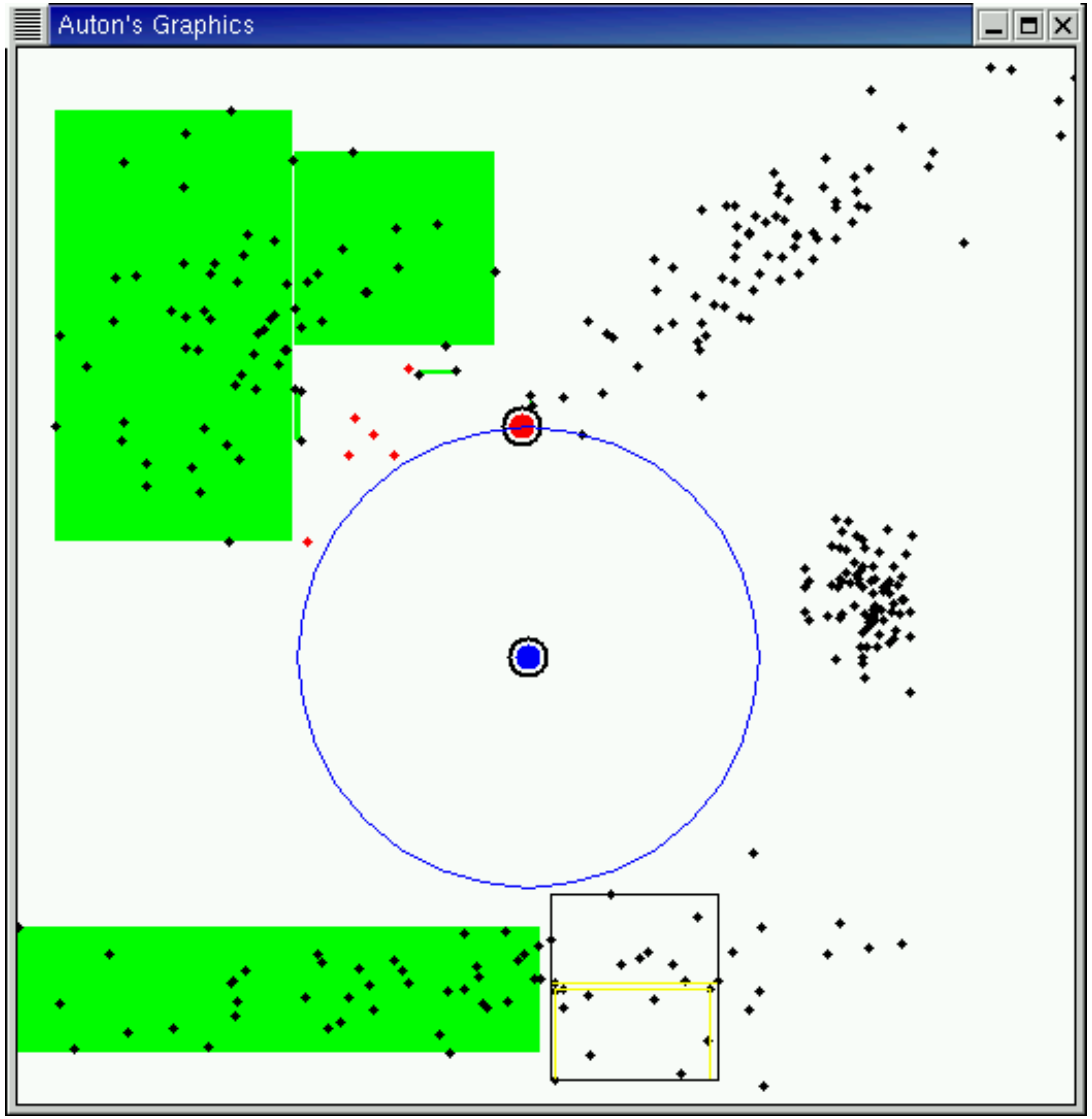

Auton's Graphics

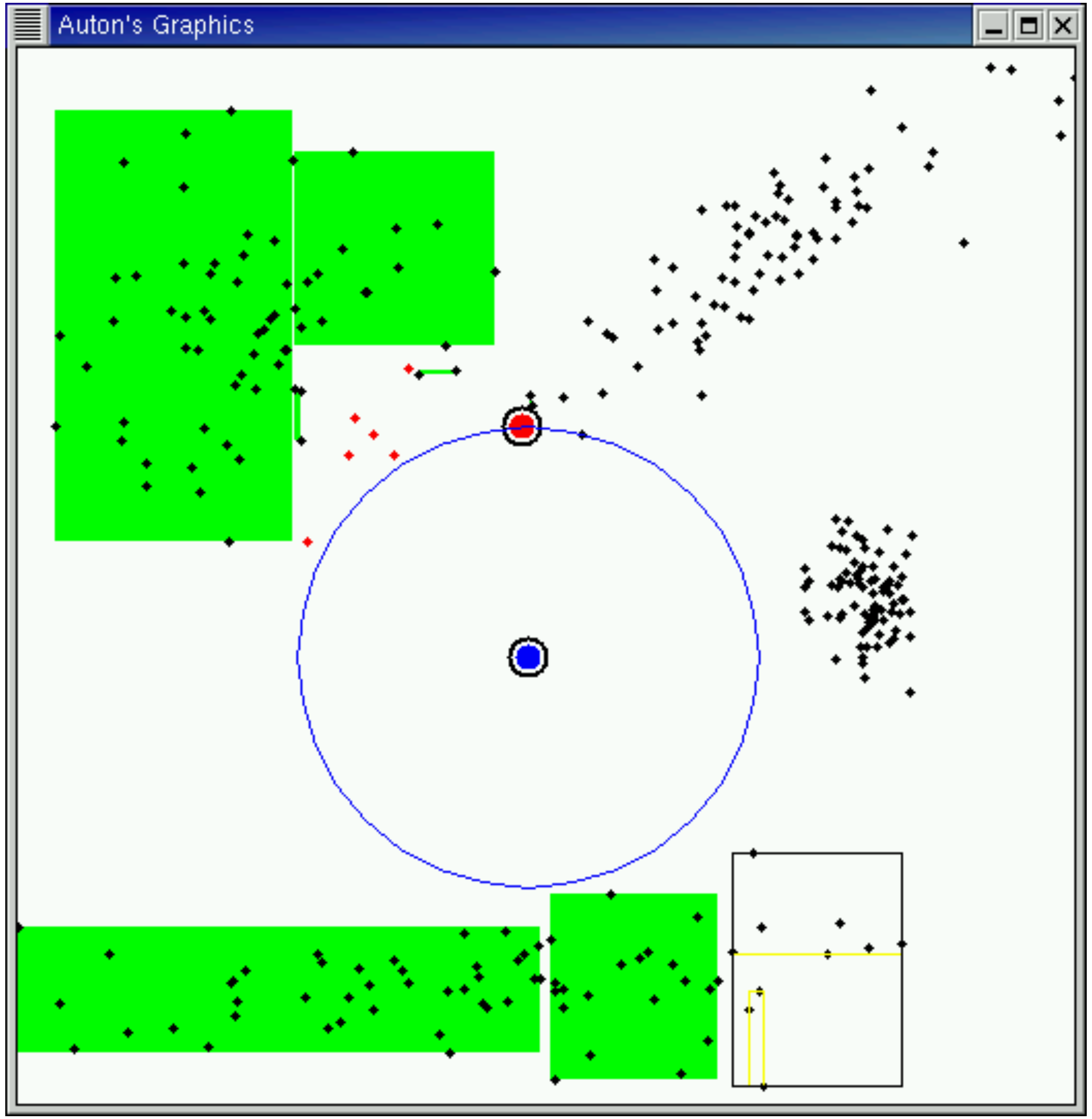

Auton's Graphics

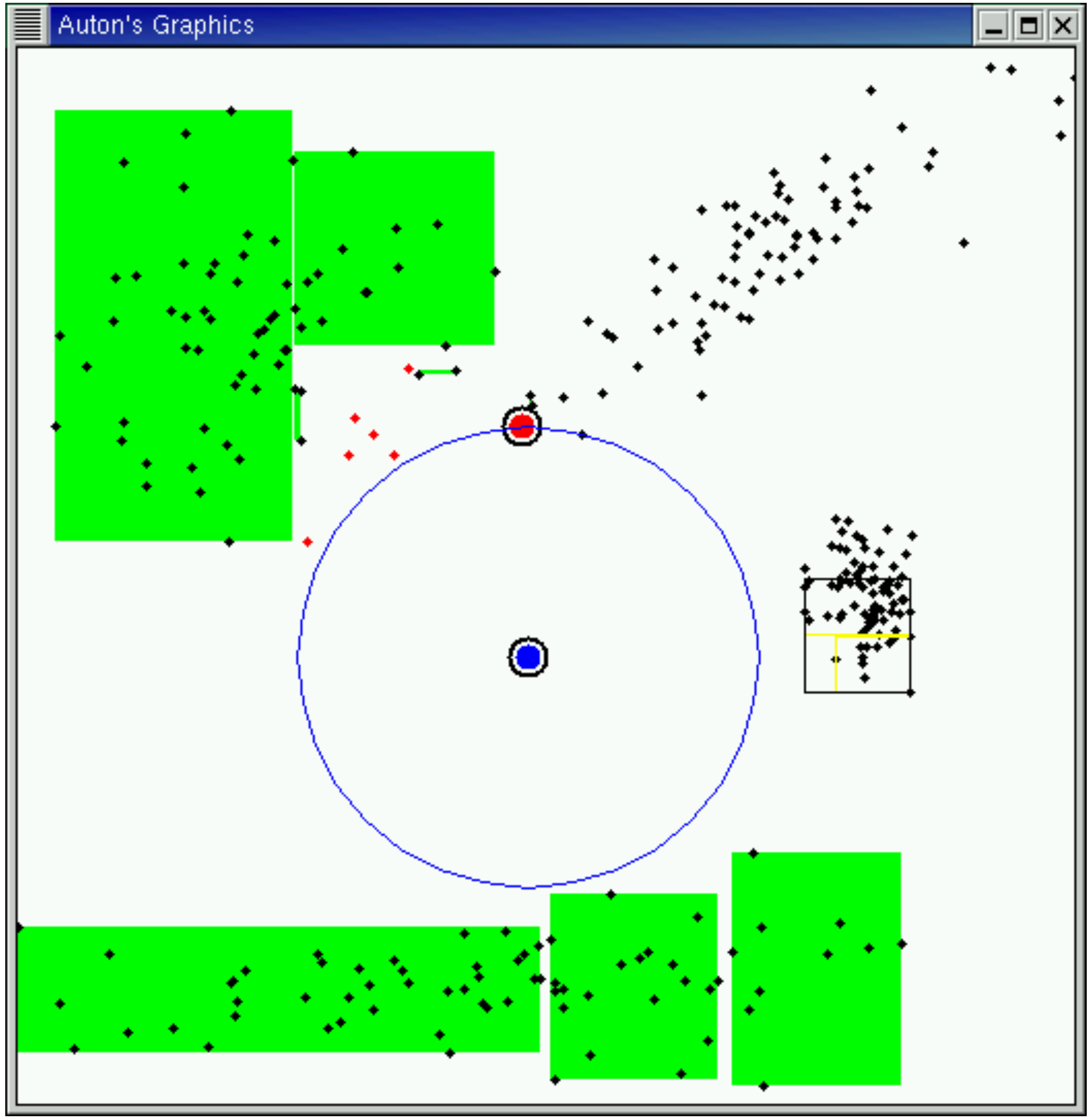

Auton's Graphics

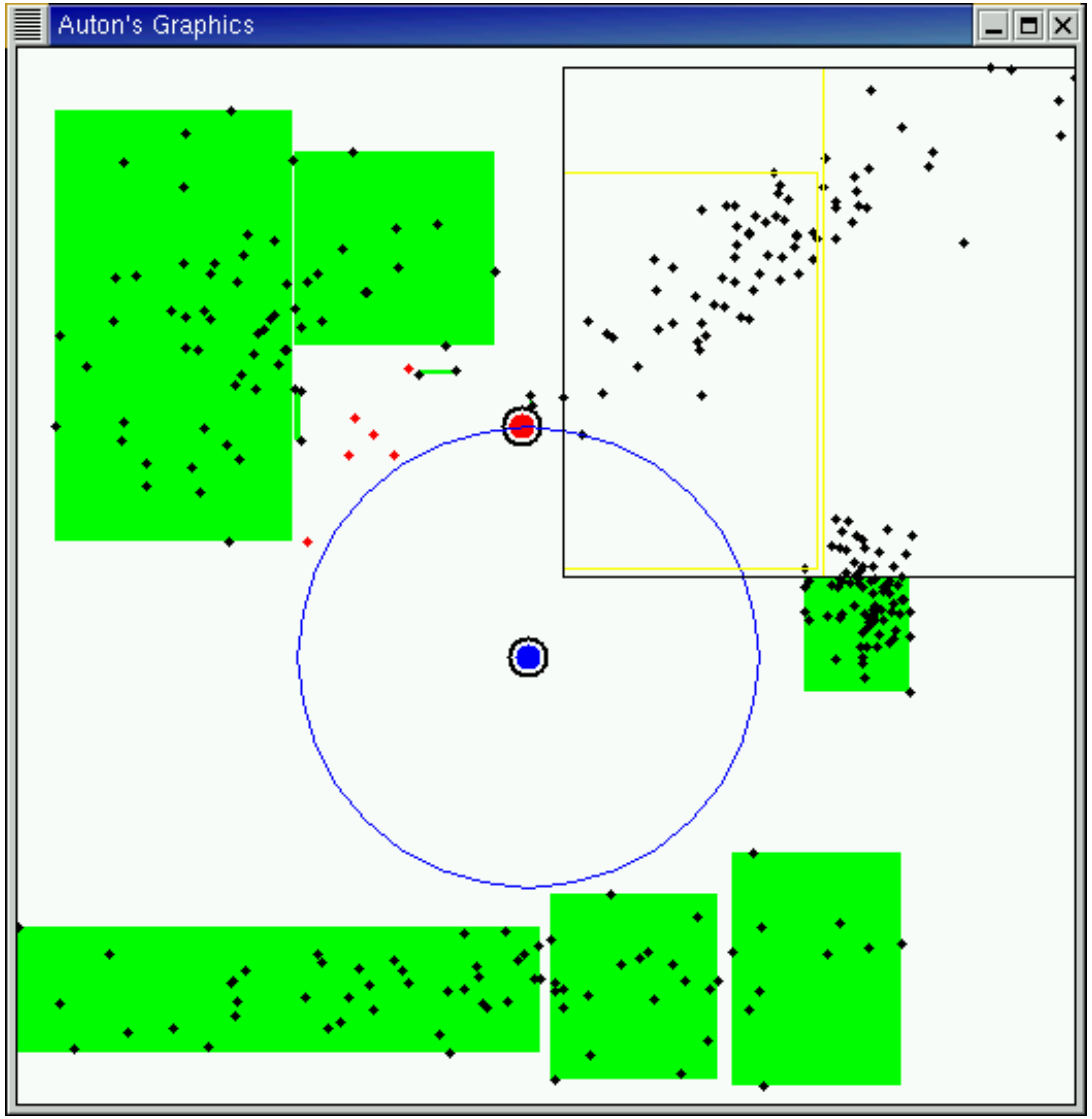

Auton's Graphics

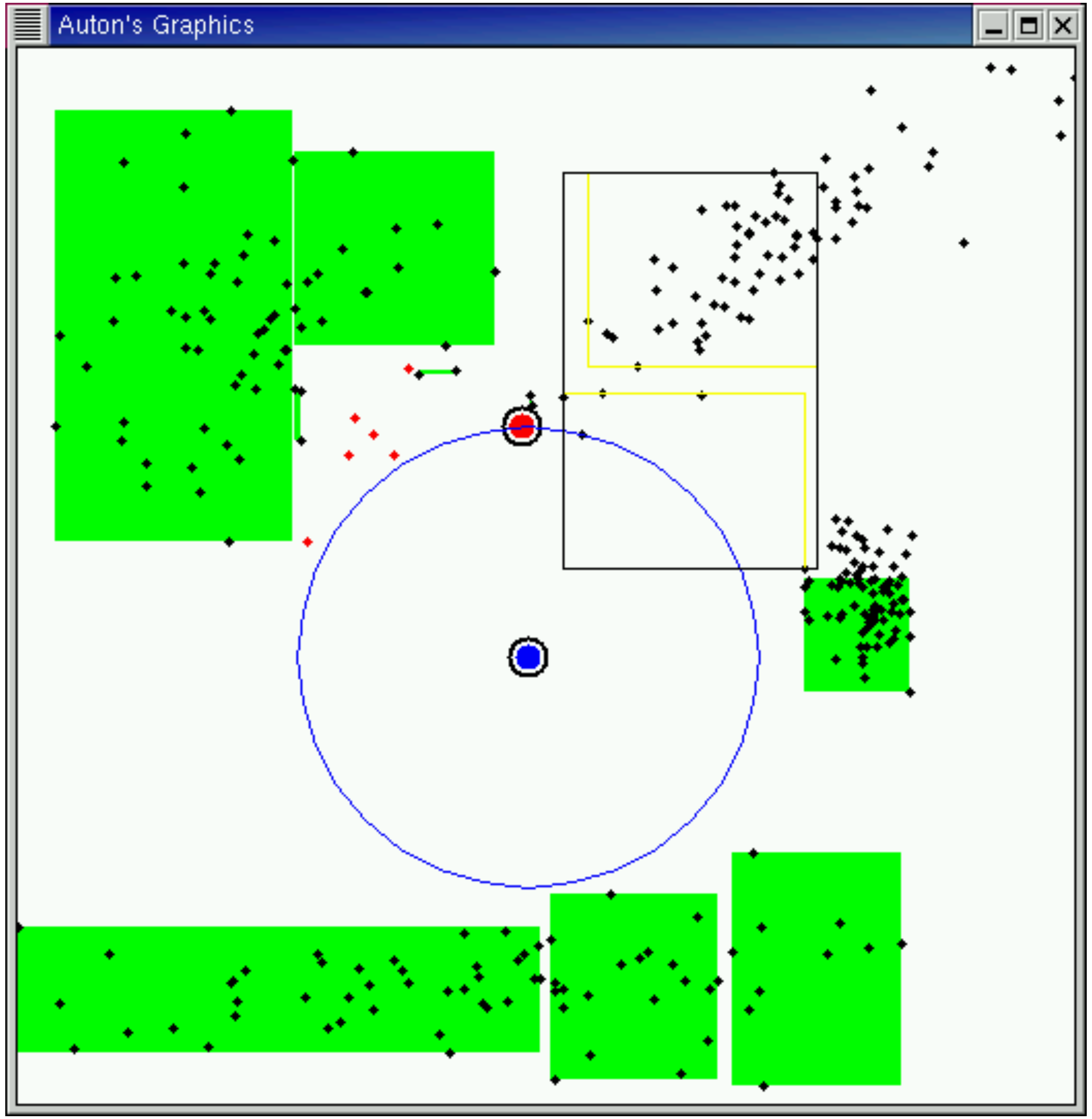

Auton's Graphics


Auton's Graphics

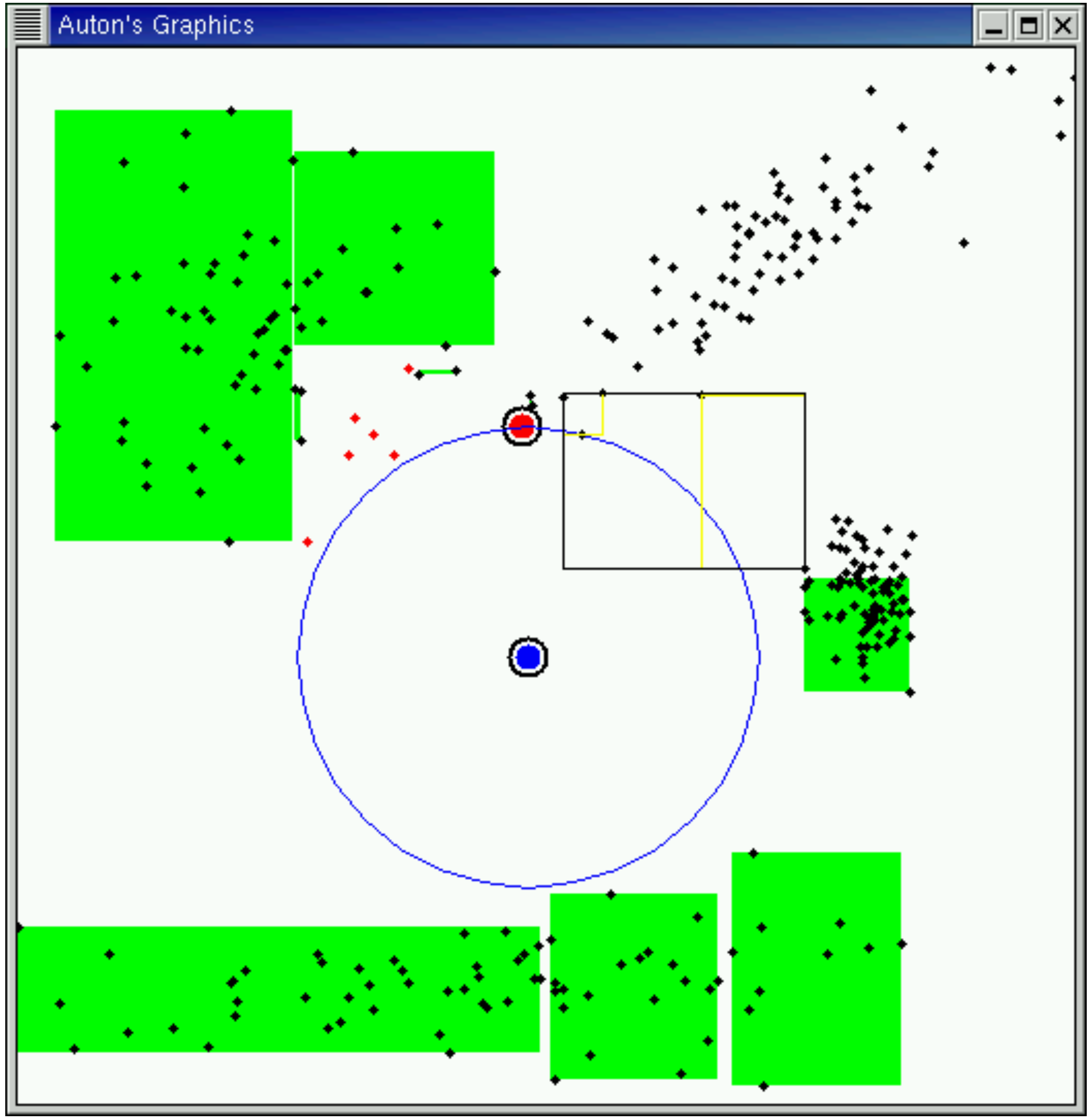

Auton's Graphics

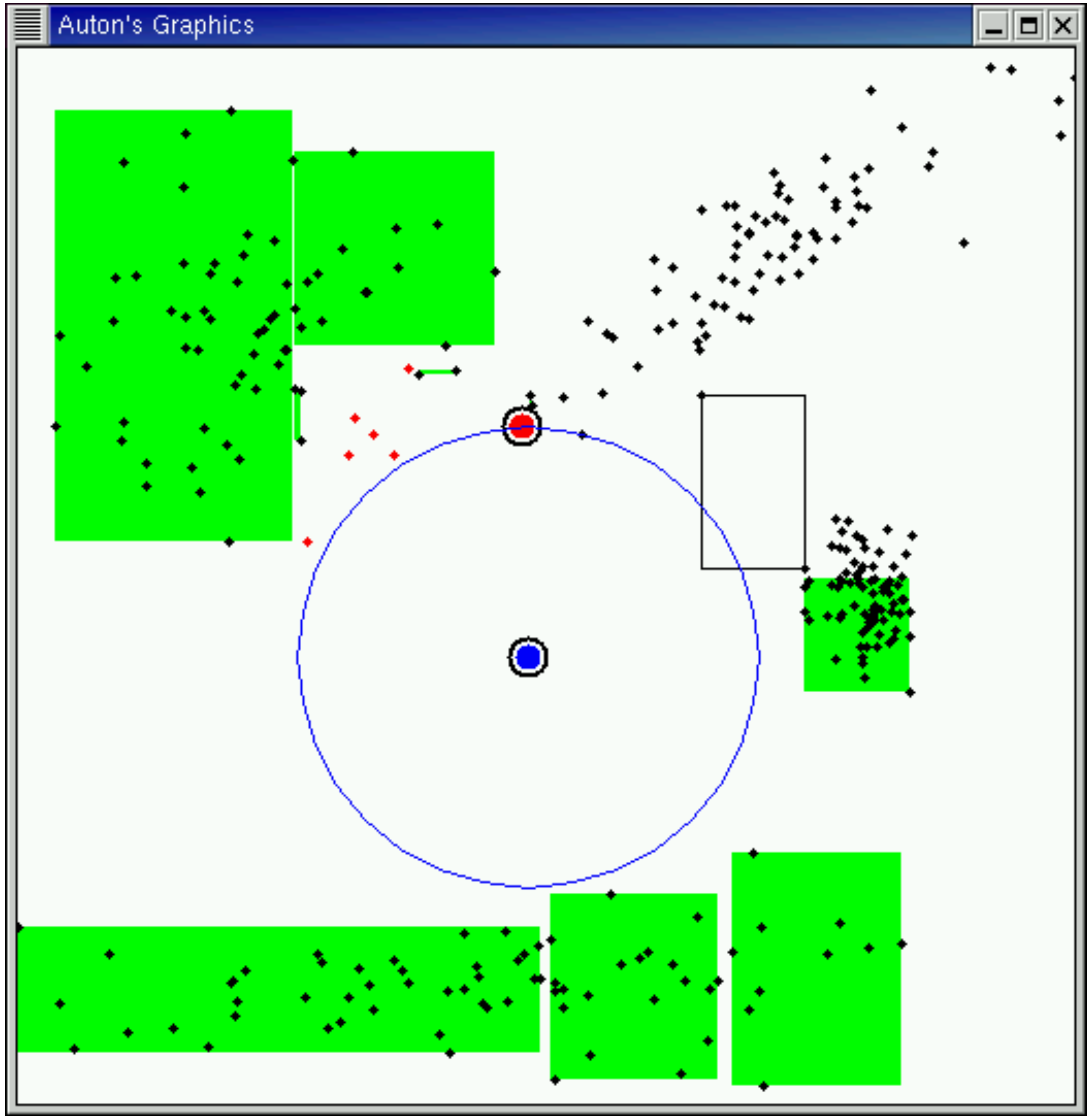

Auton's Graphics

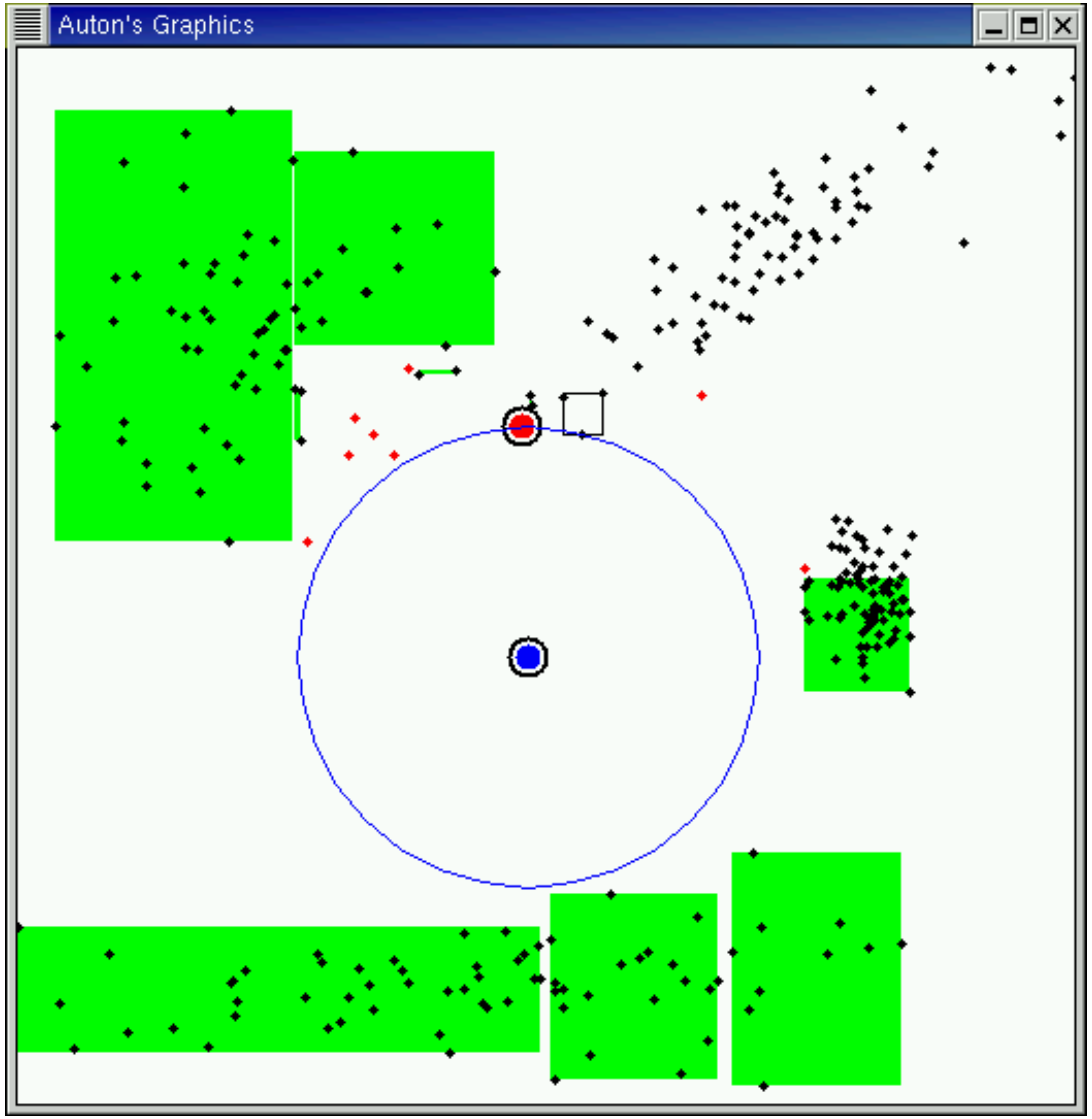

Auton's Graphics

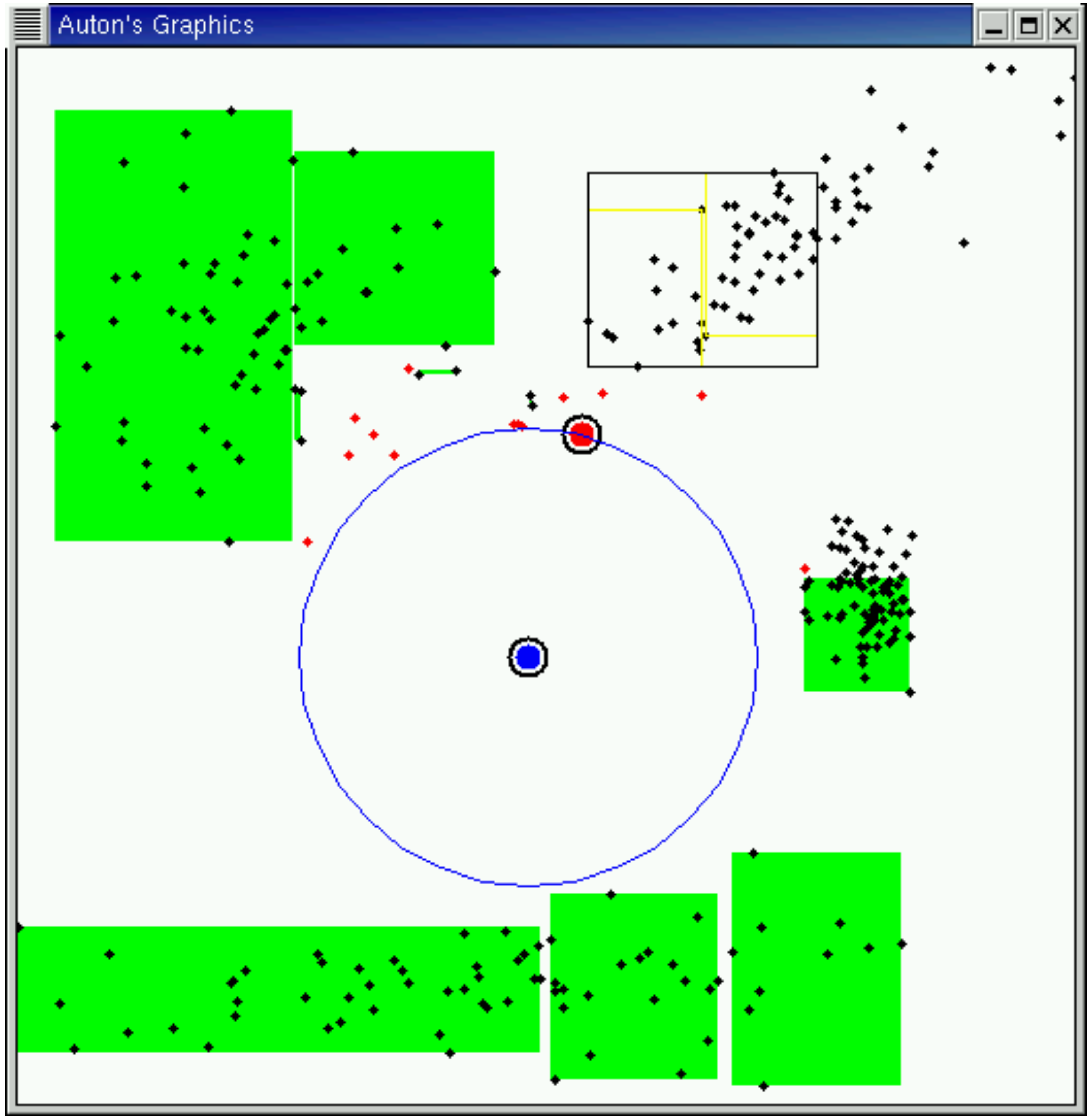


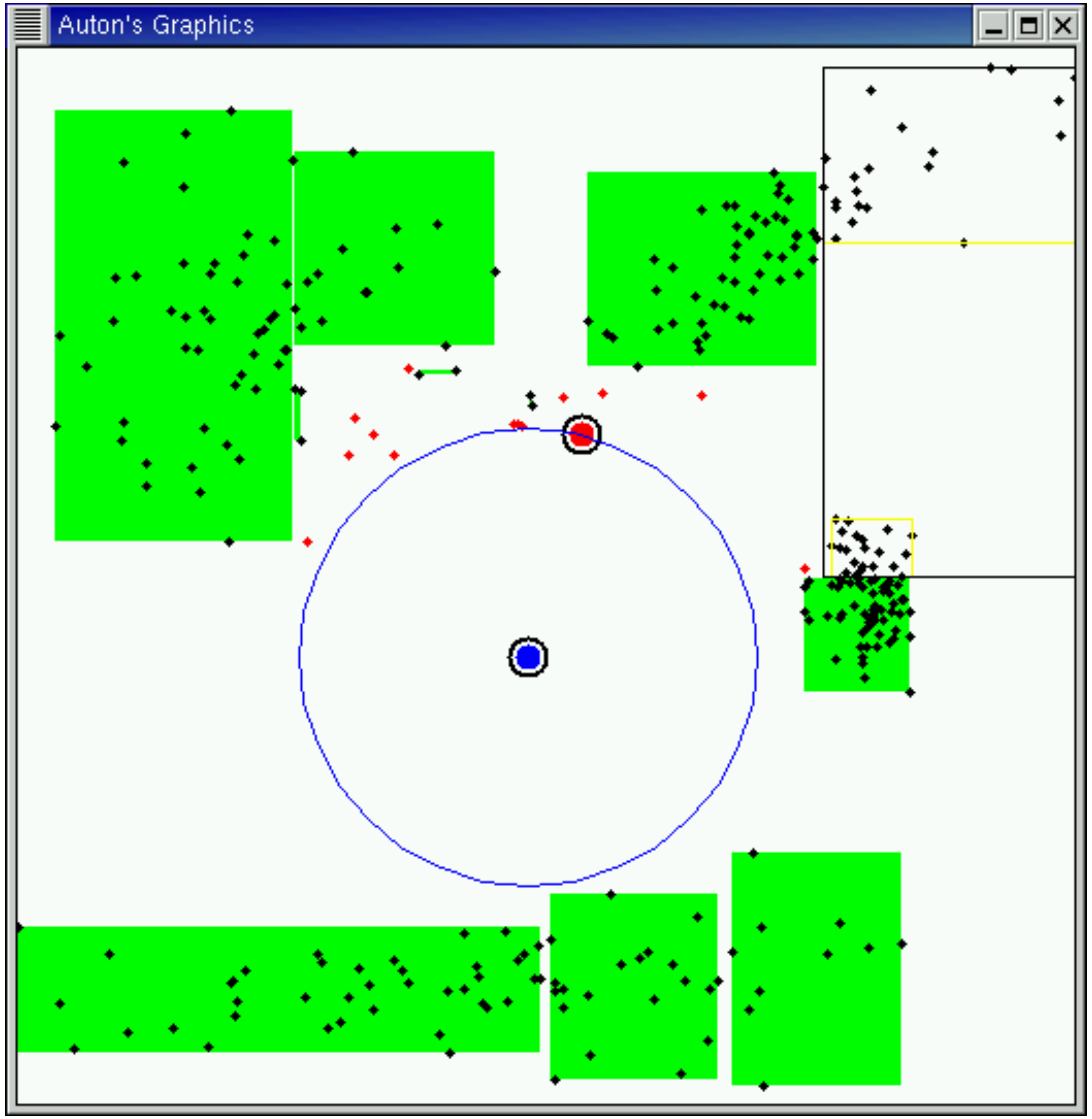
Auton's Graphics

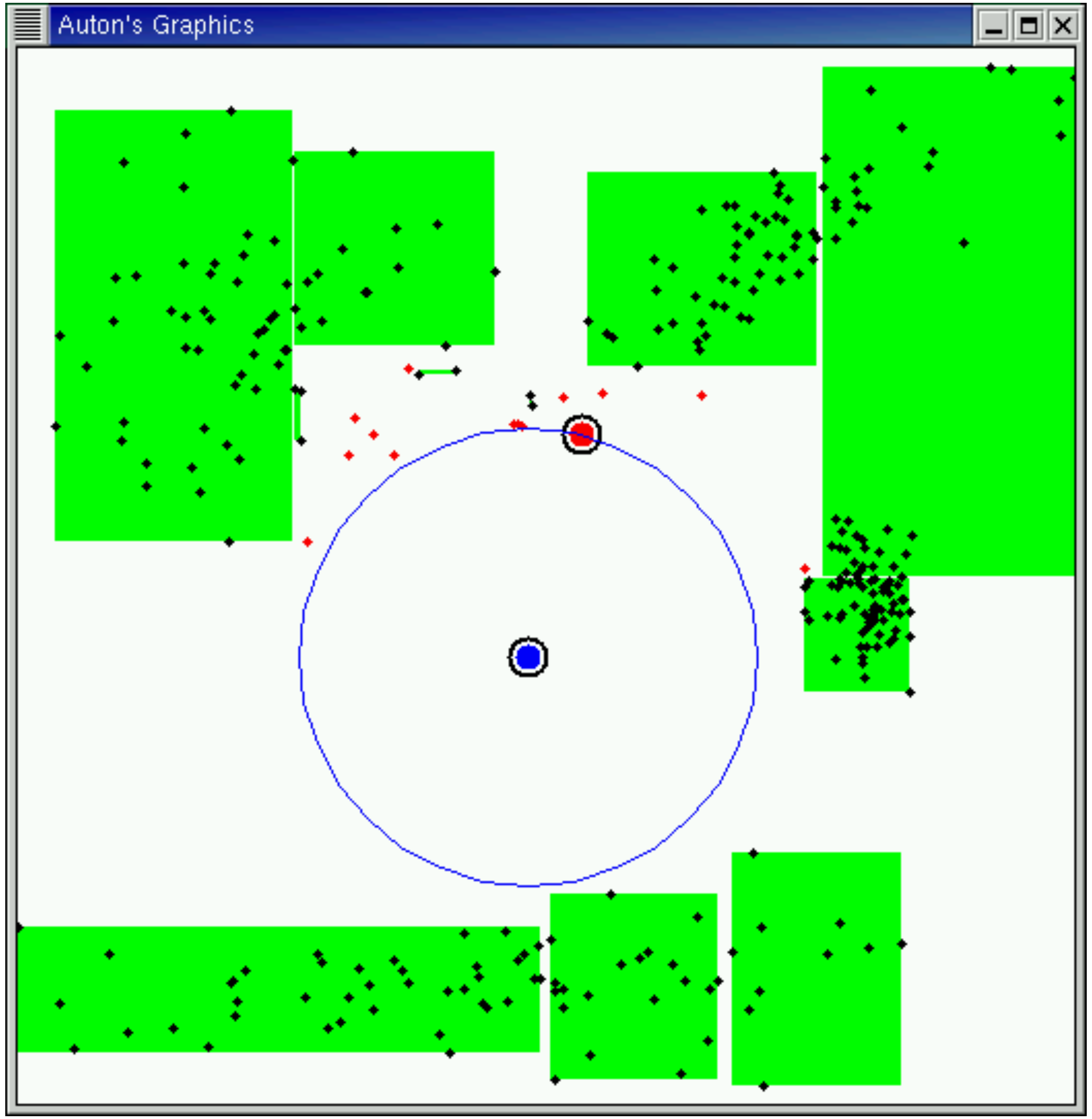

Auton's Graphics


Auton's Graphics

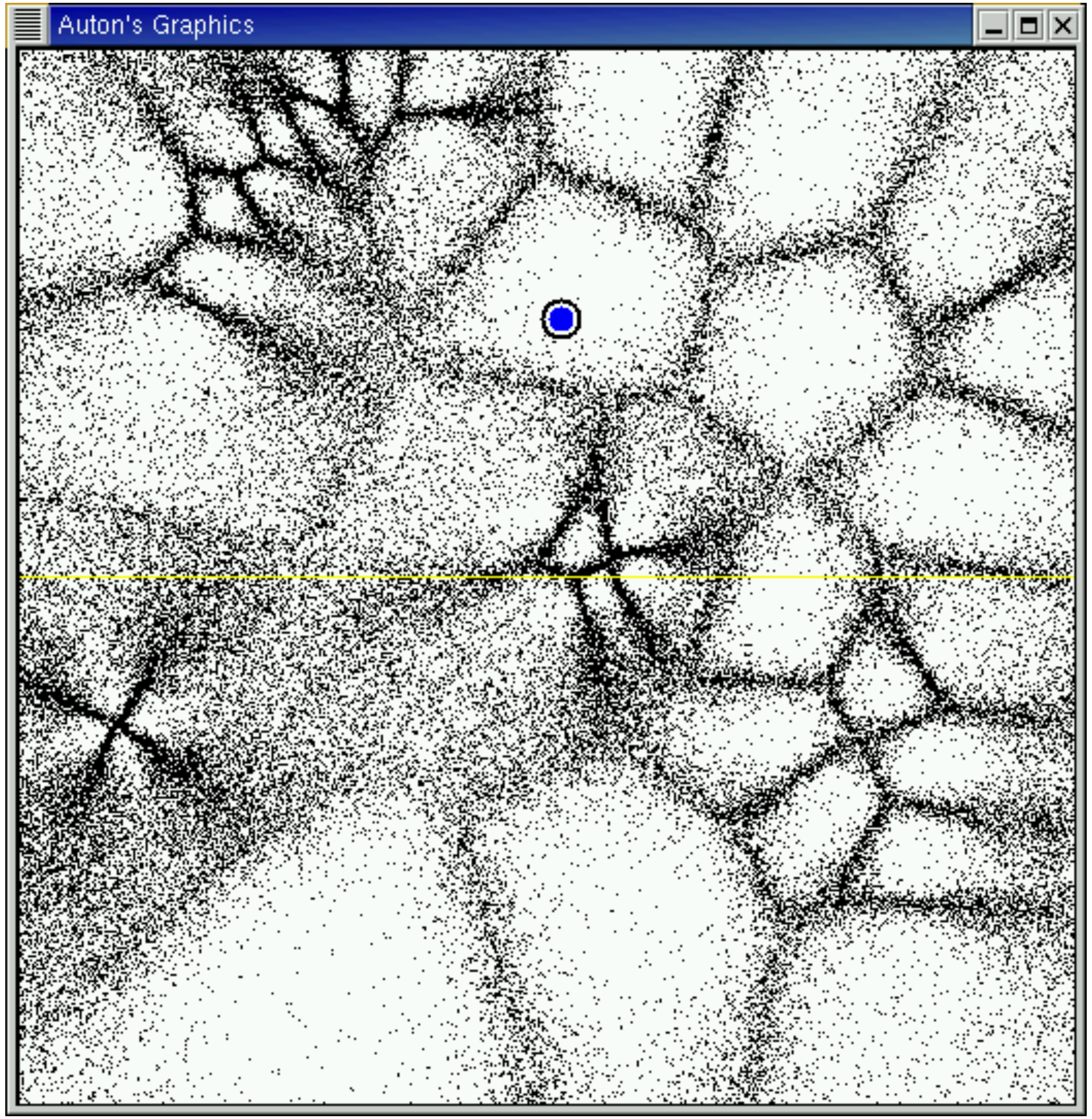

Auton's Graphics

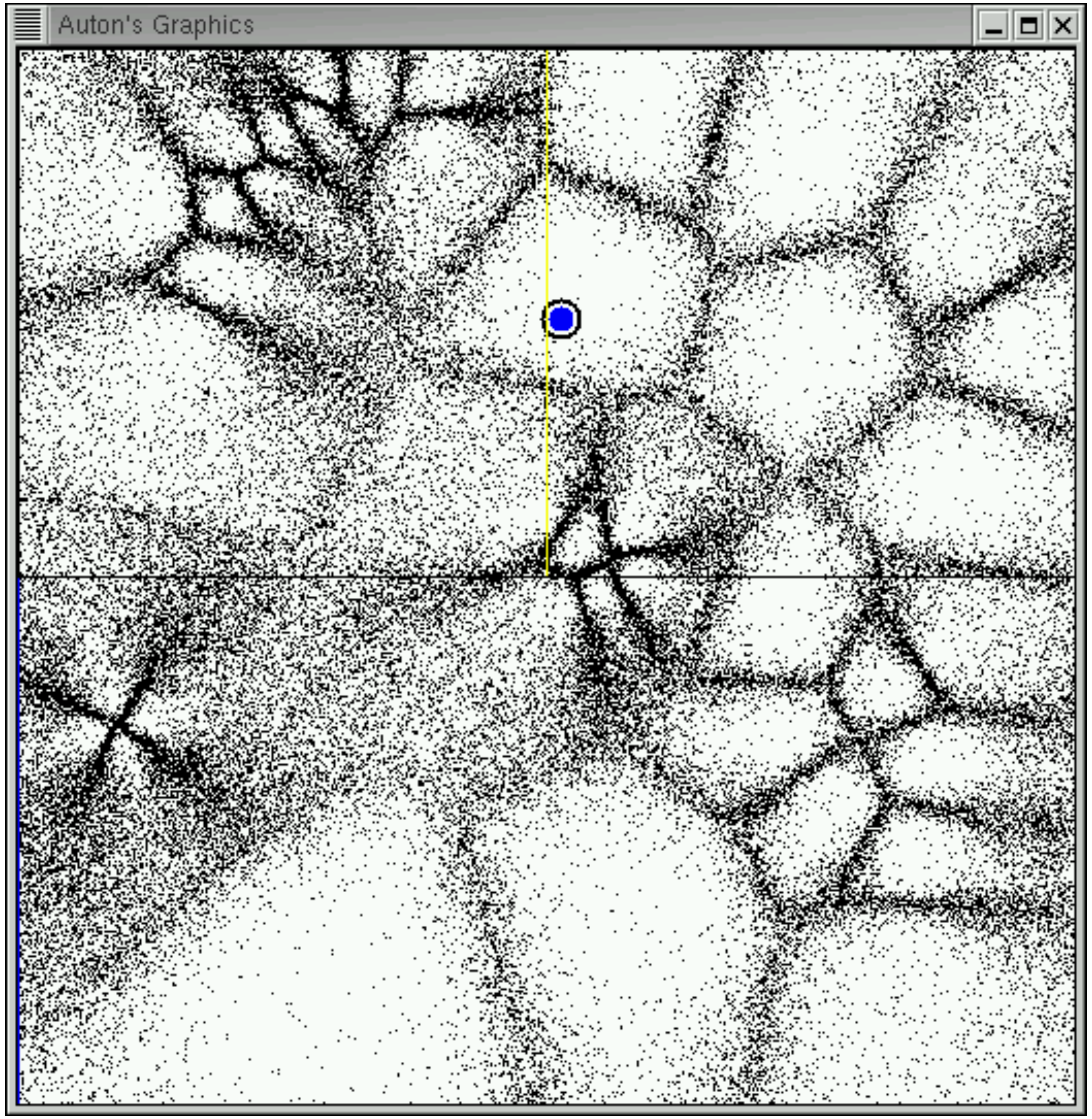

Auton's Graphics

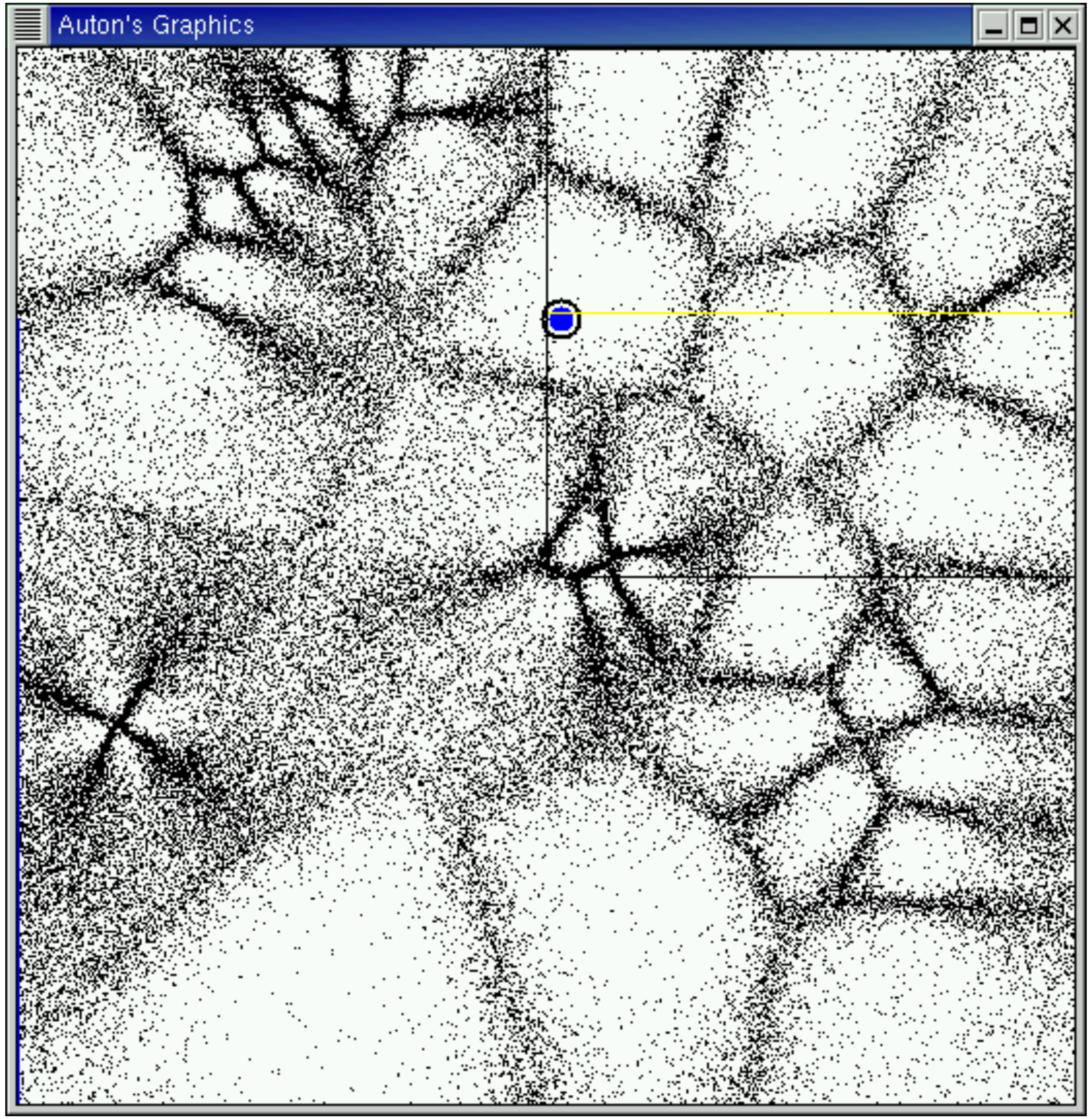

Auton's Graphics

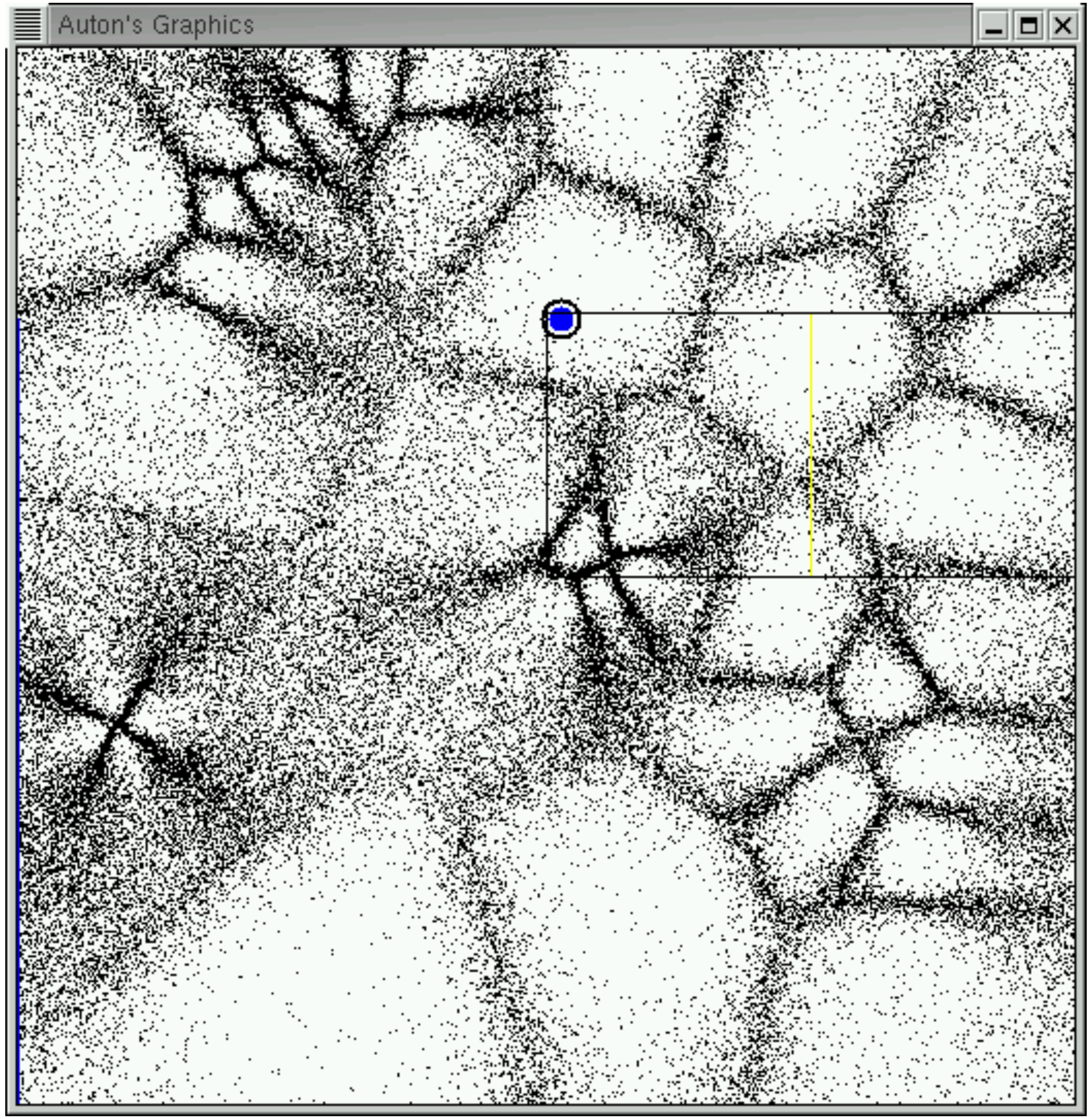

Auton's Graphics

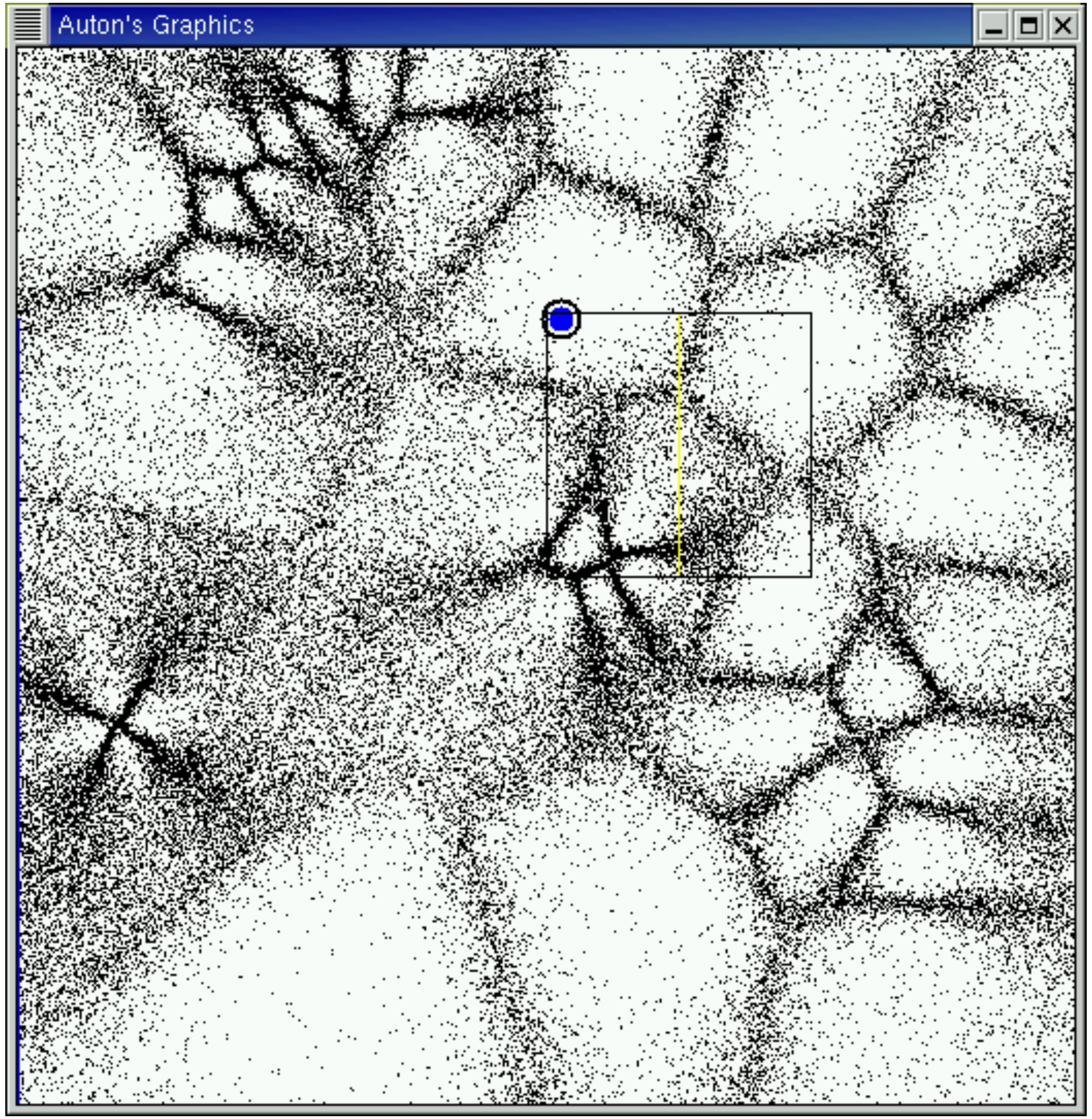
Animation of NN search with large data set

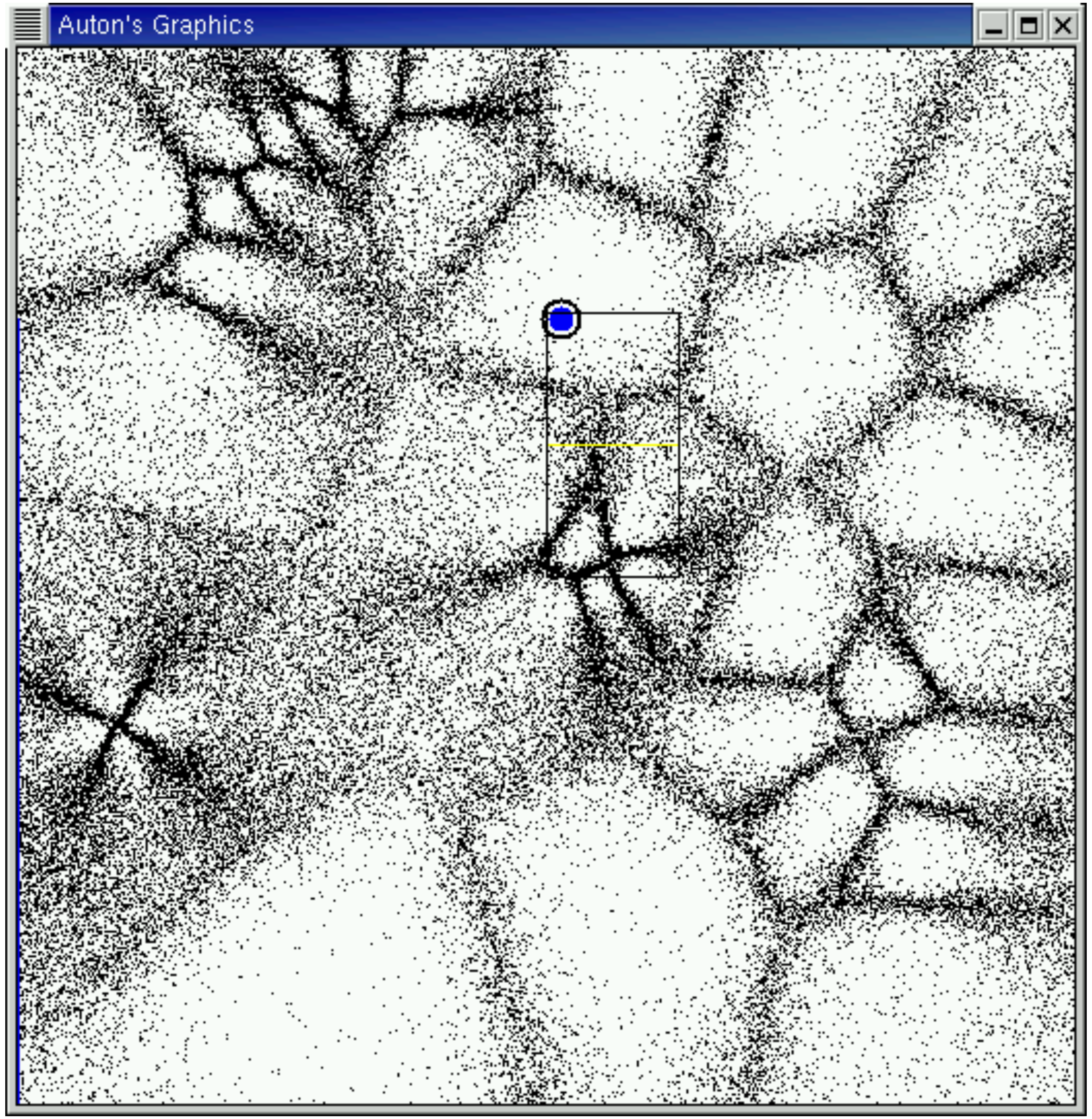


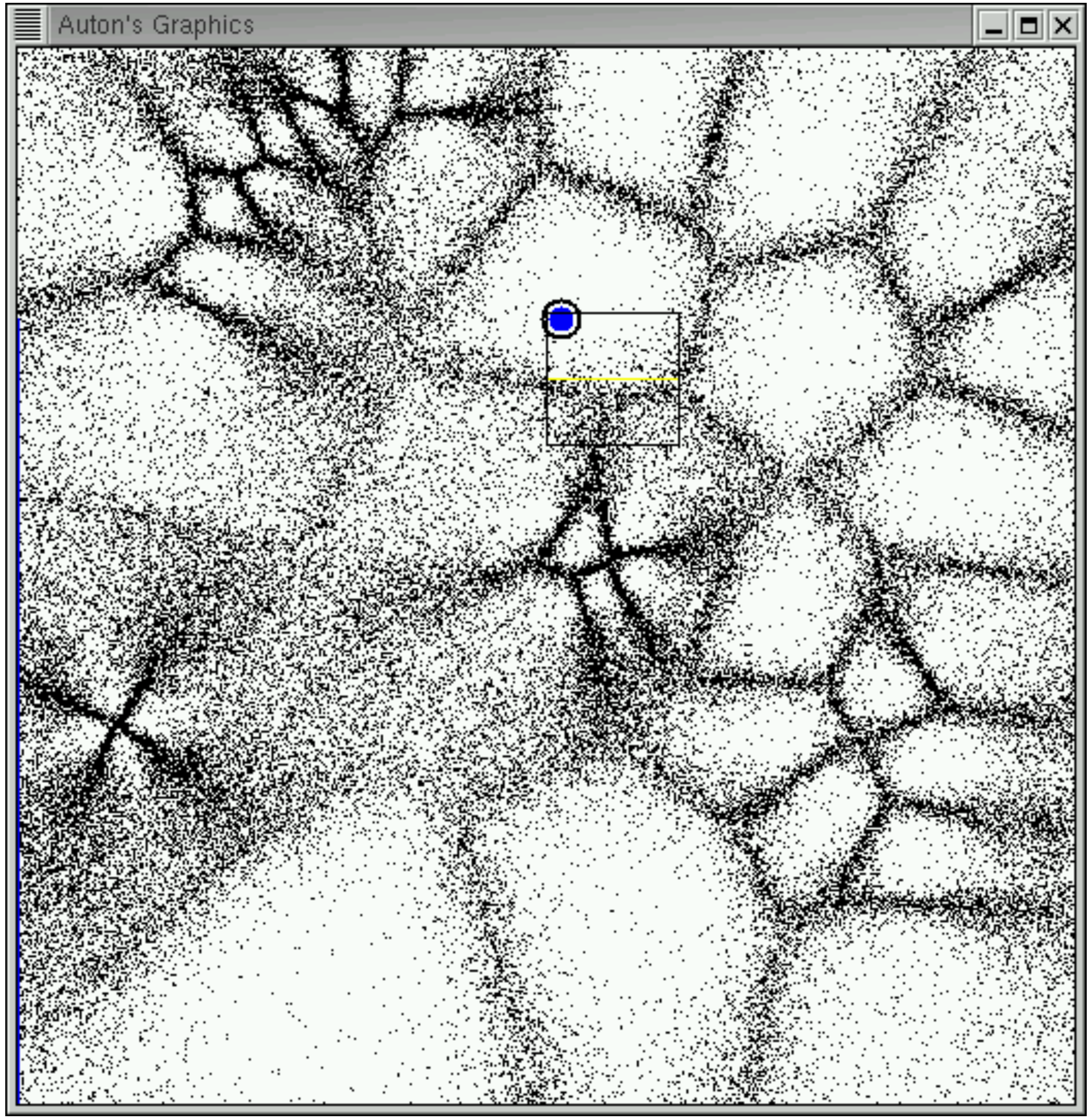

Auton's Graphics

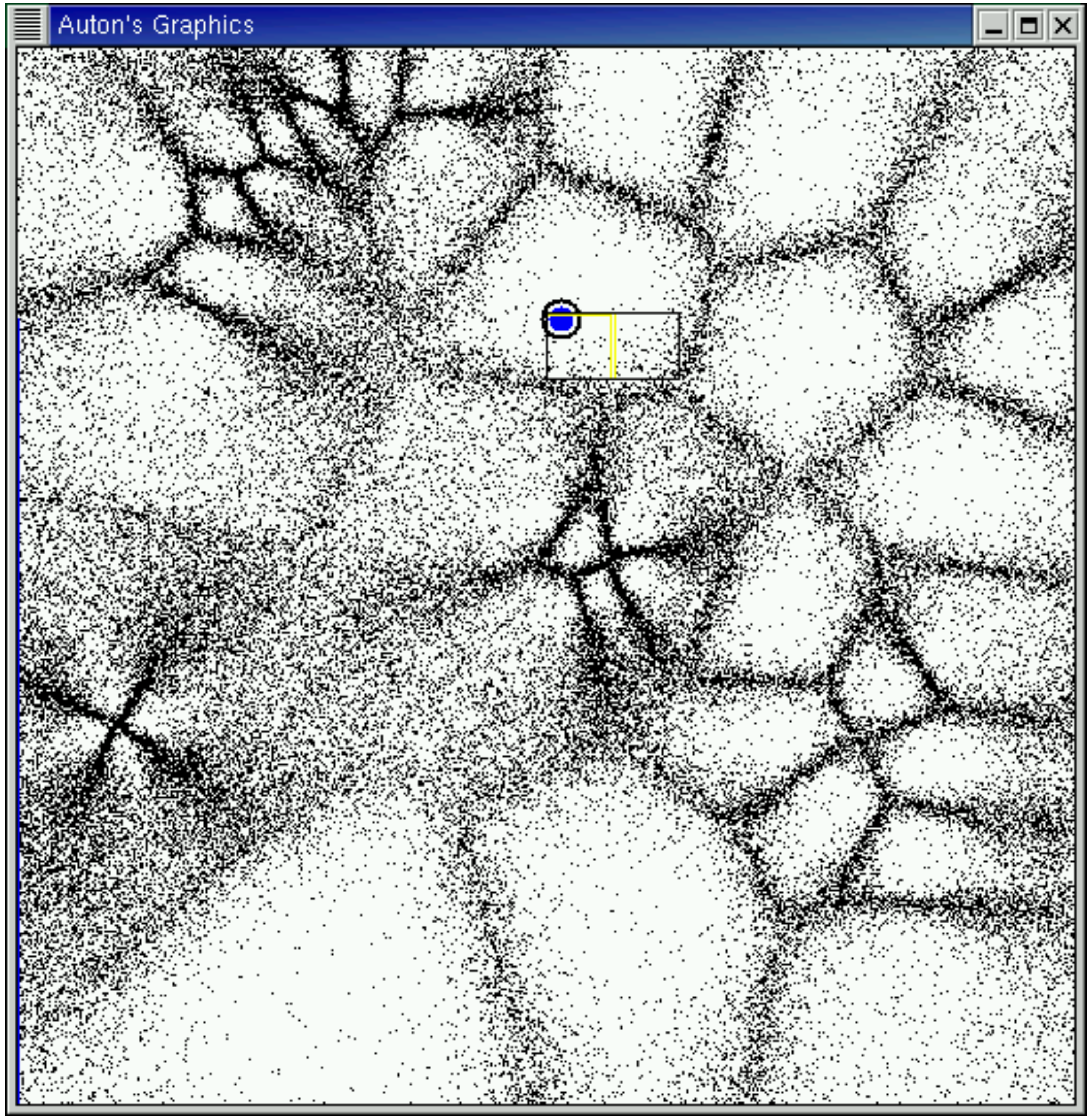

Auton's Graphics

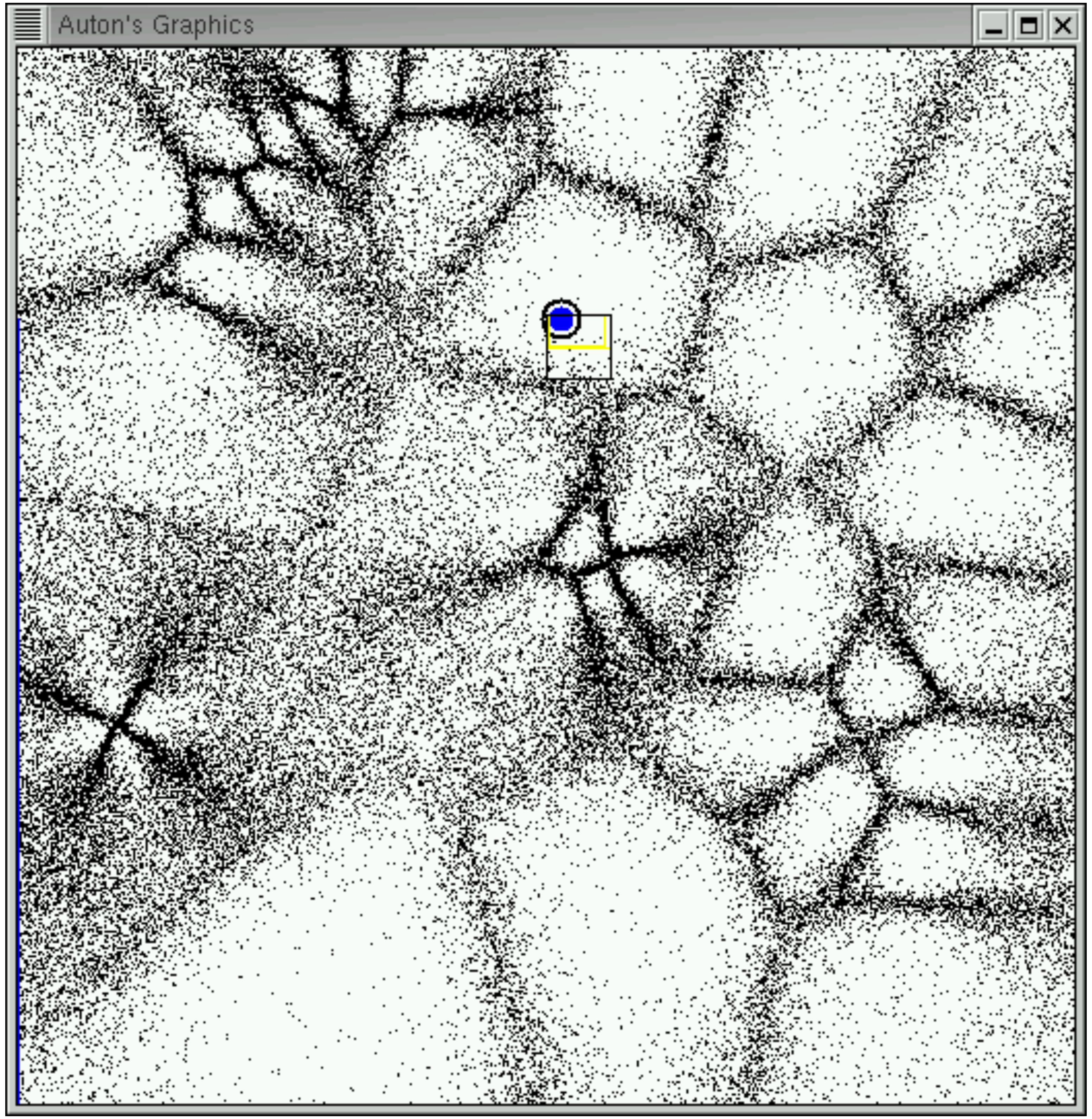

Auton's Graphics

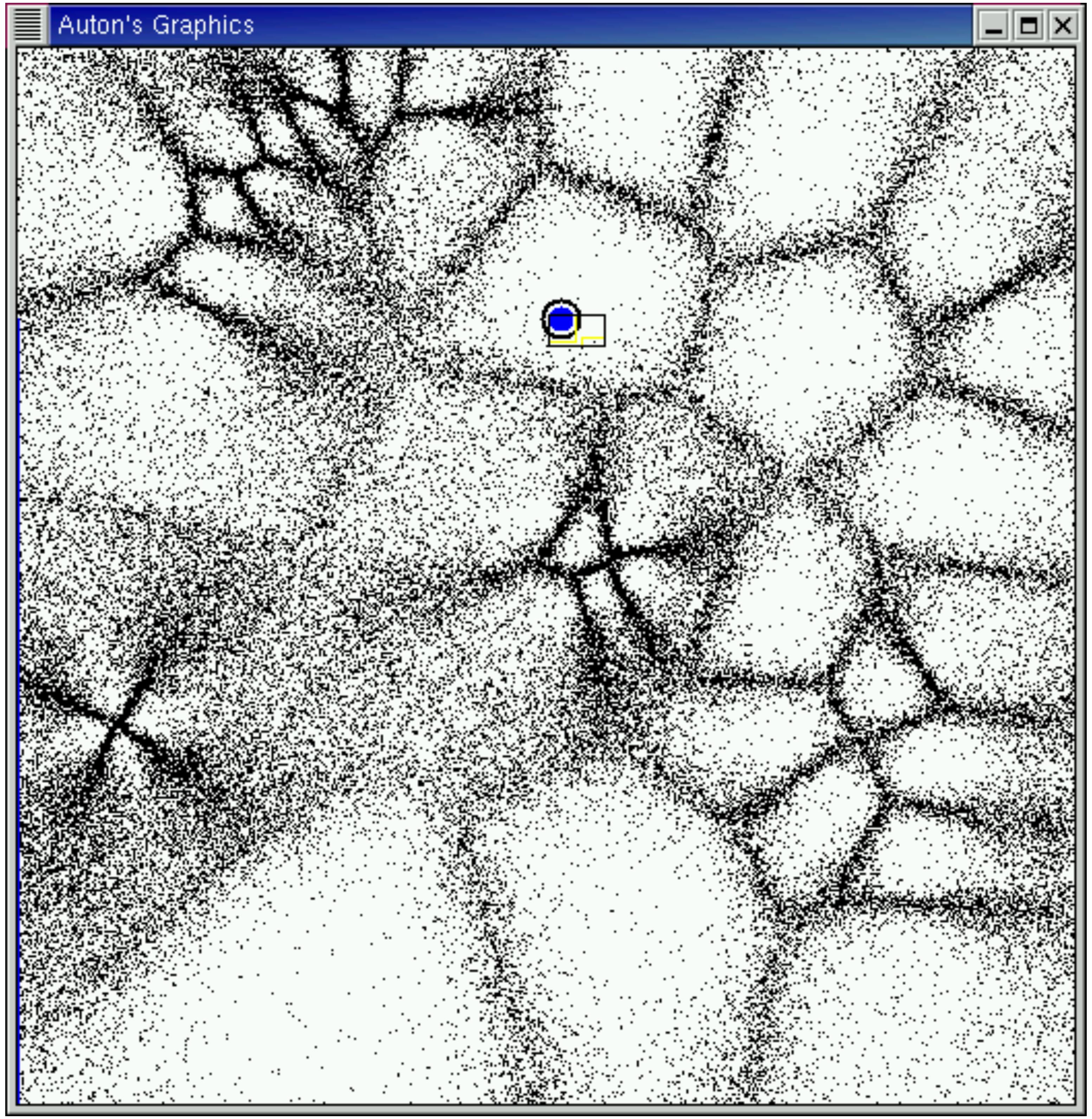

Auton's Graphics

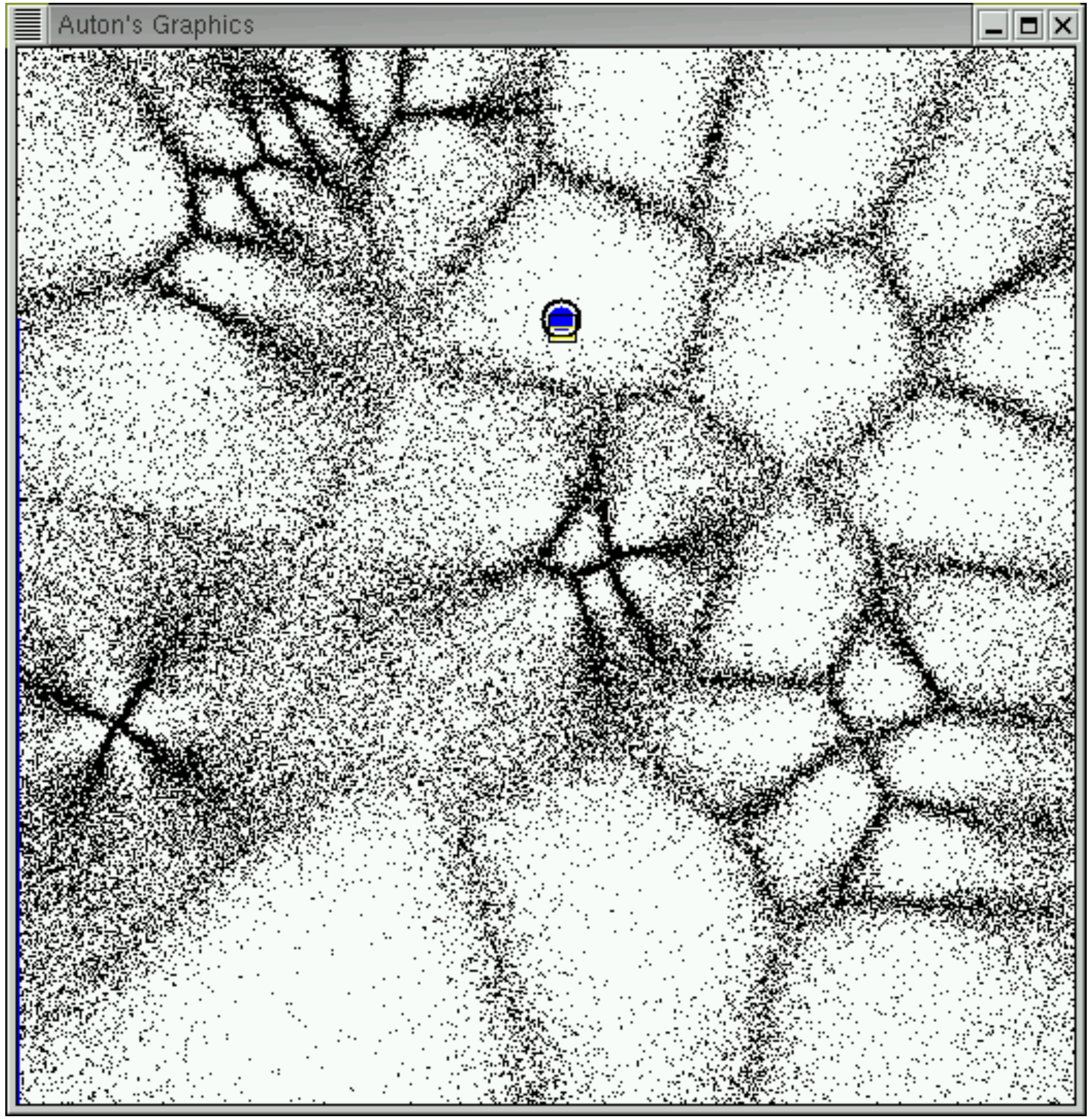

Auton's Graphics

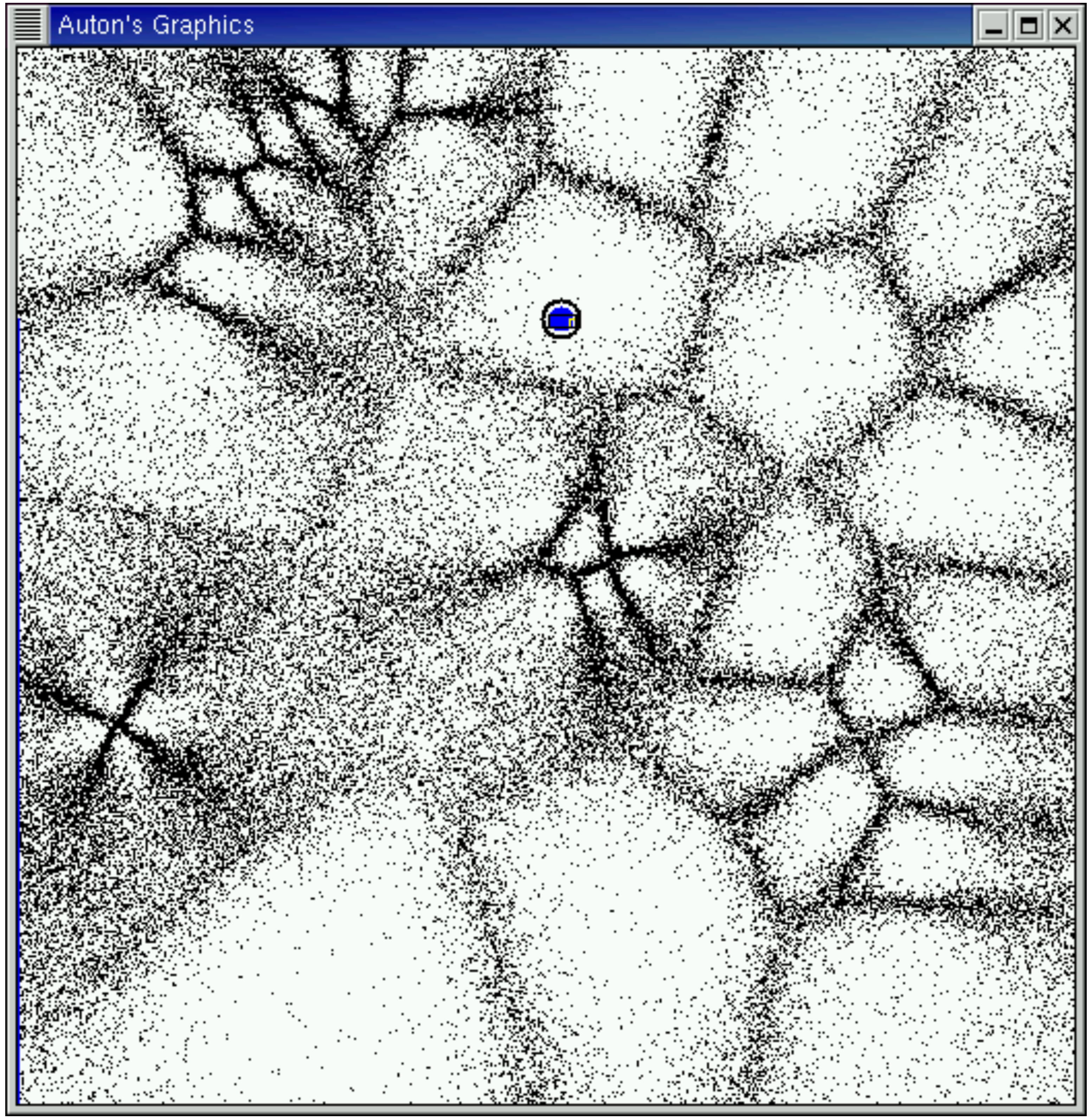

Auton's Graphics

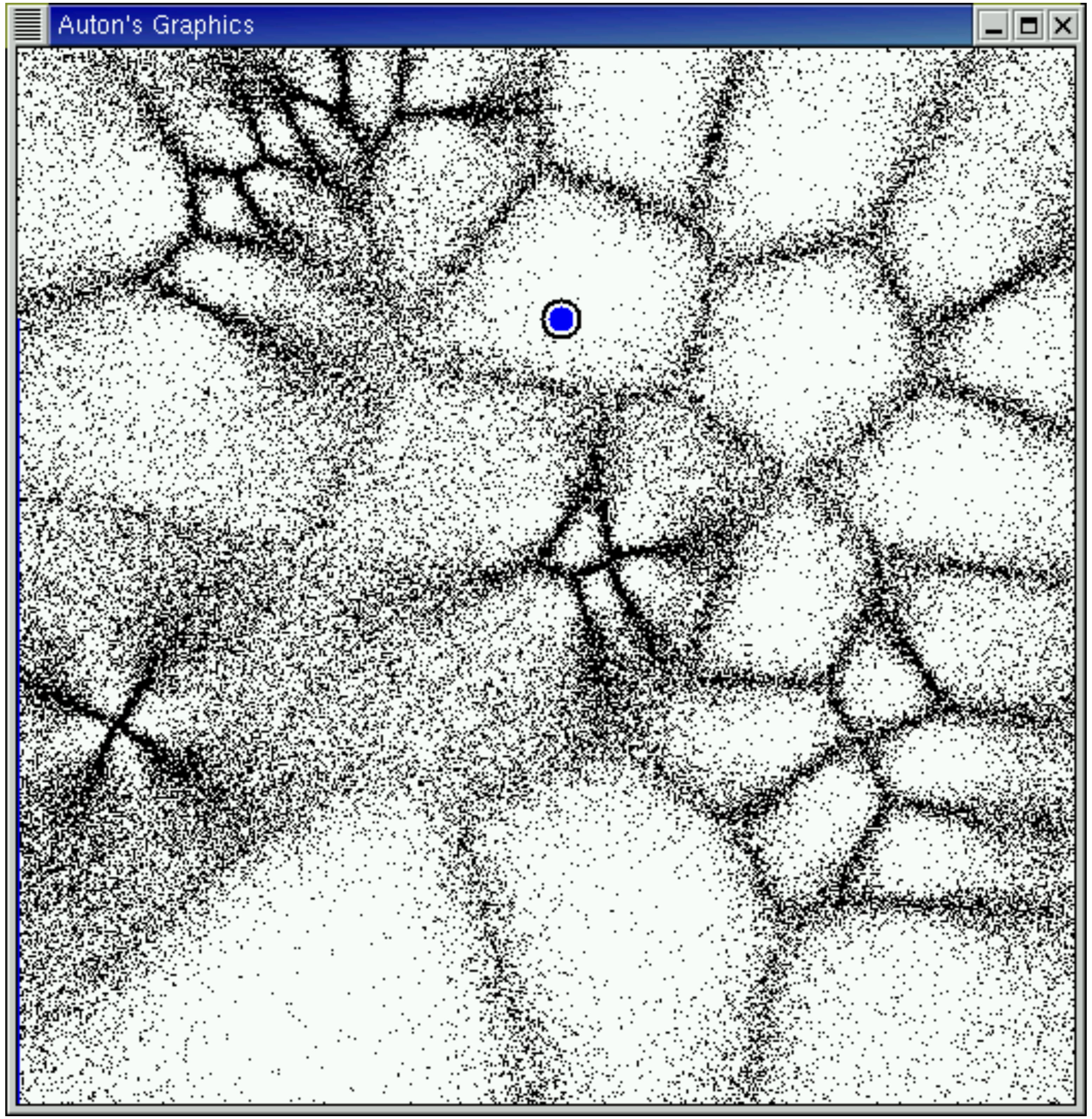

Auton's Graphics

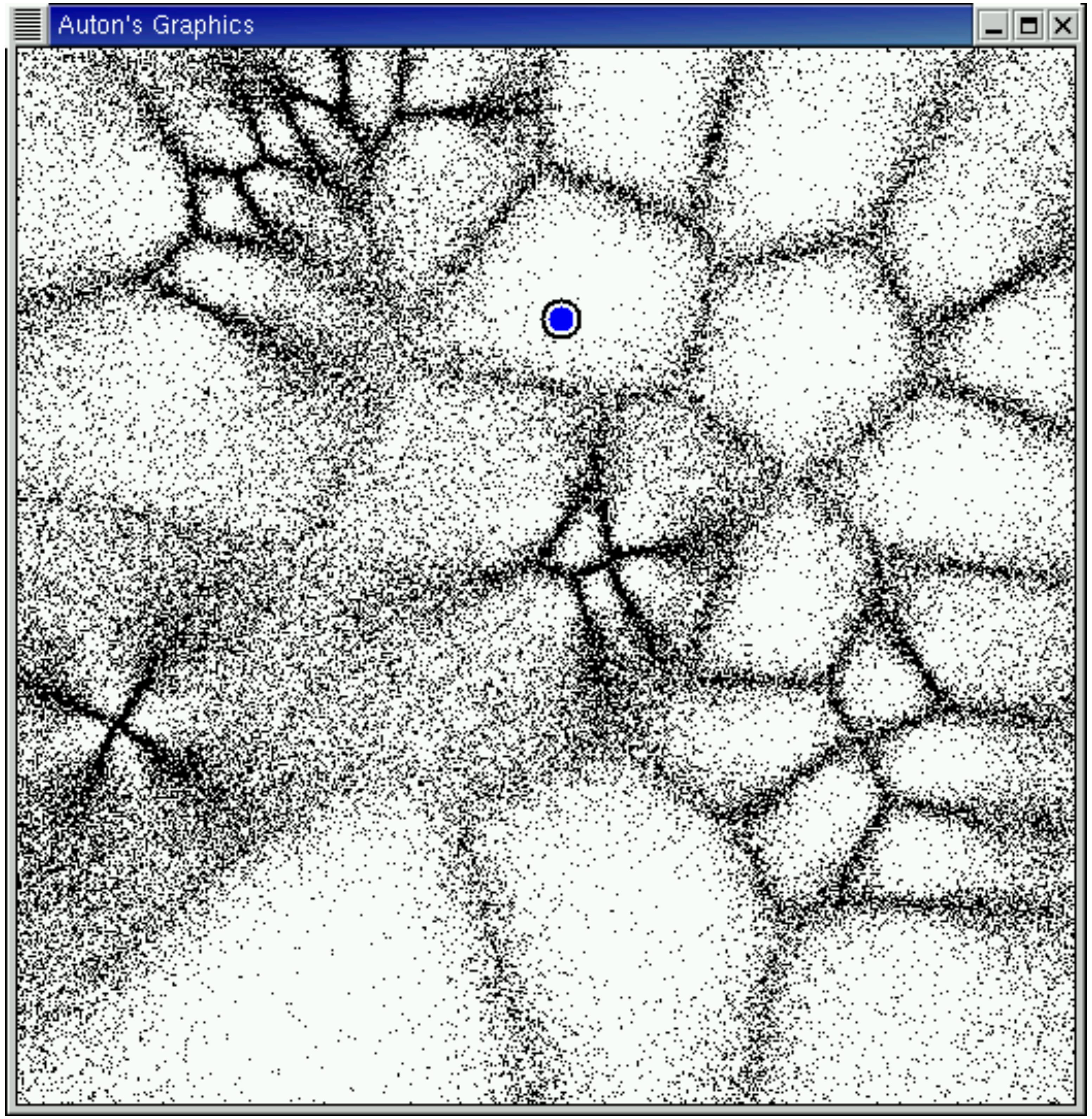

Auton's Graphics

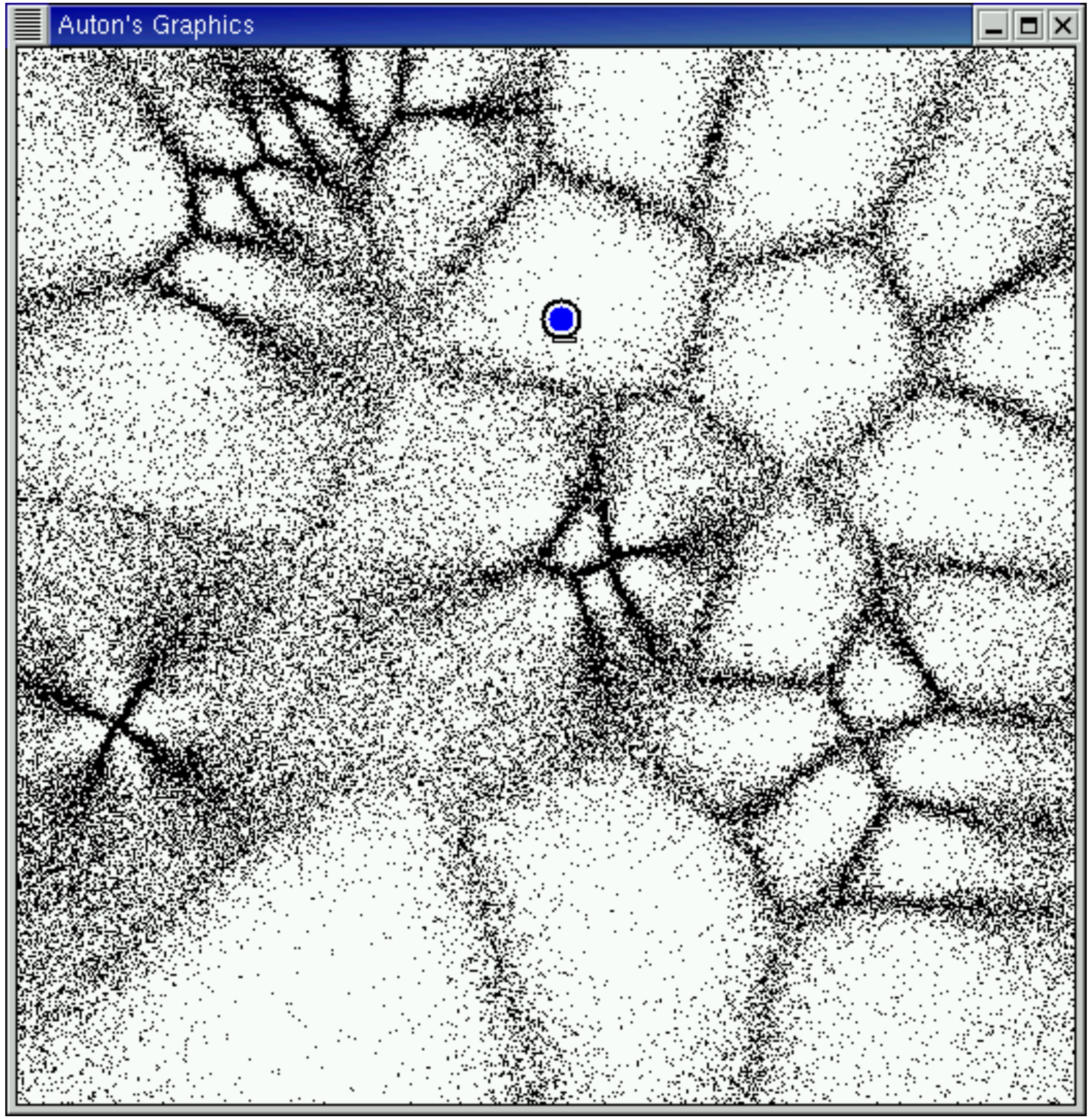

Auton's Graphics

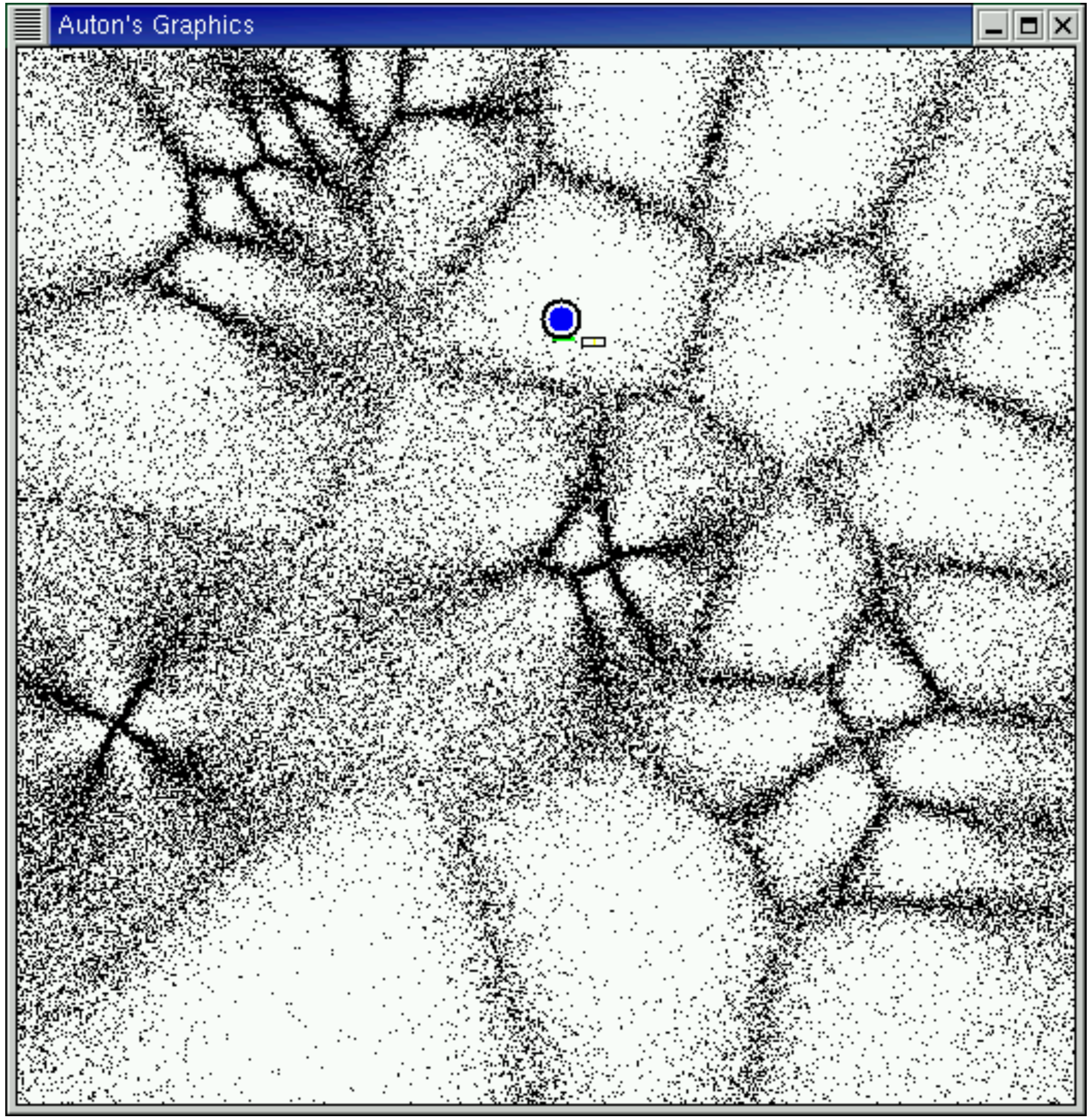

Auton's Graphics

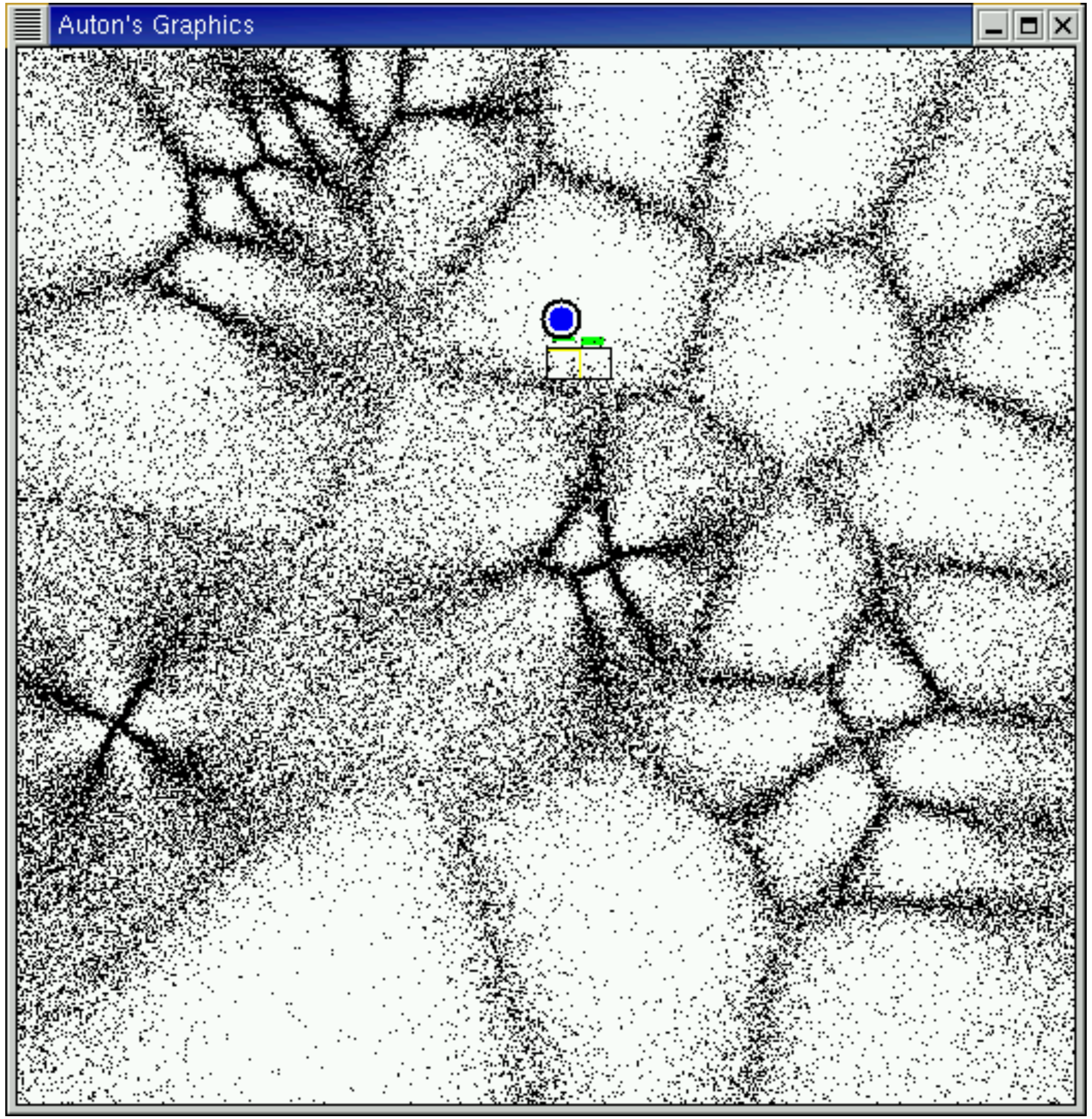

Auton's Graphics

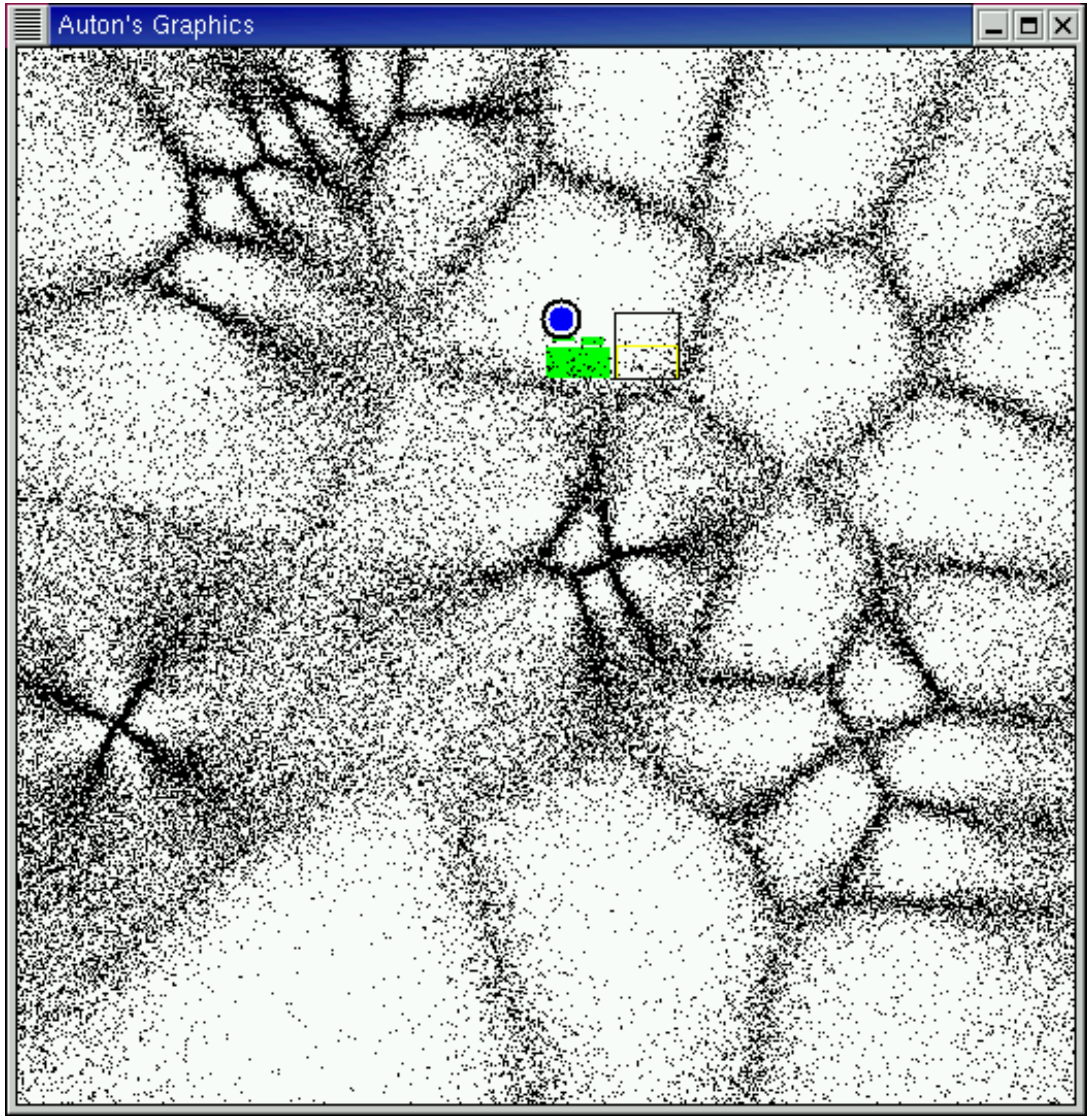

Auton's Graphics

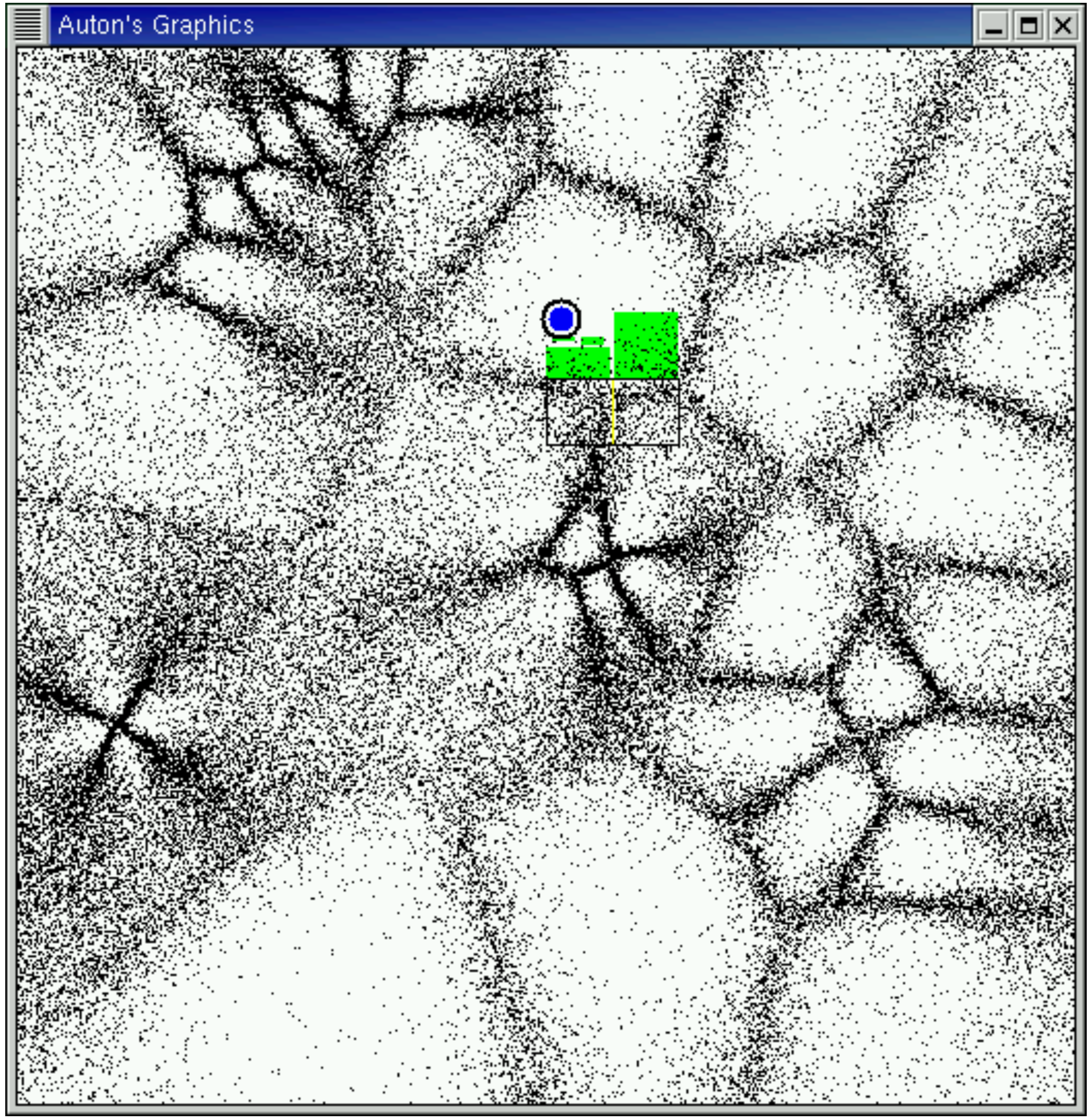

Auton's Graphics

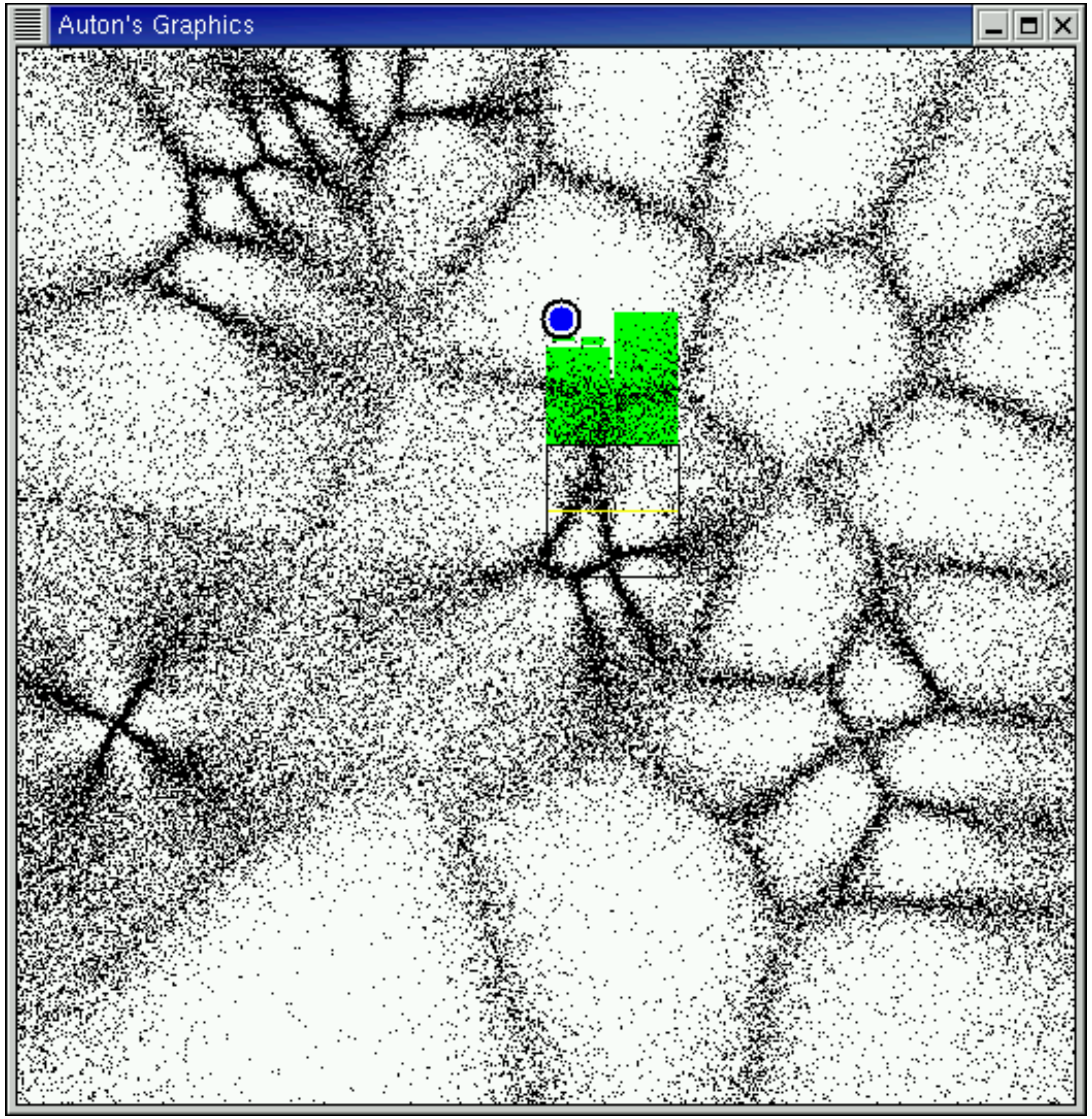

Auton's Graphics

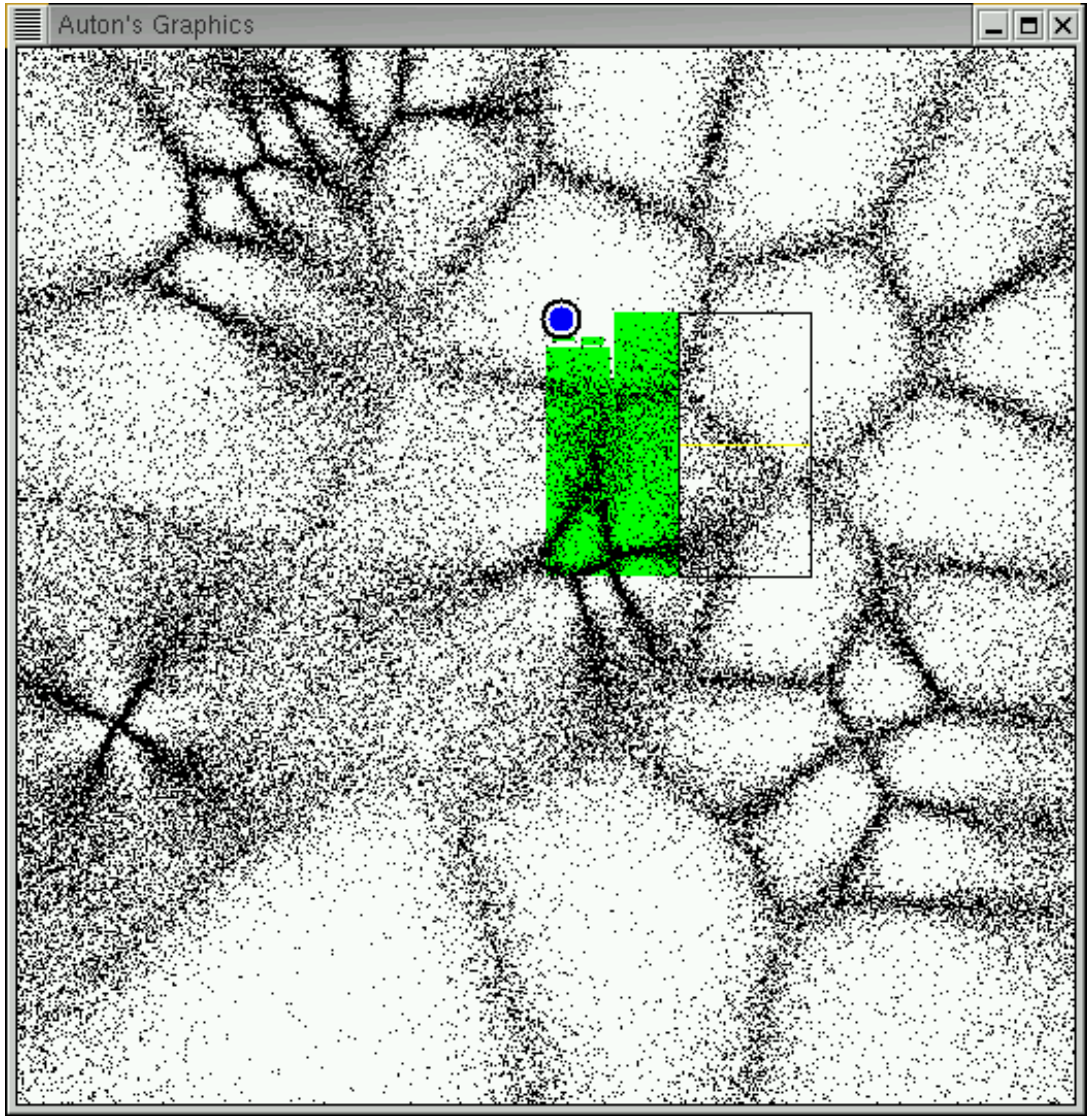

Auton's Graphics

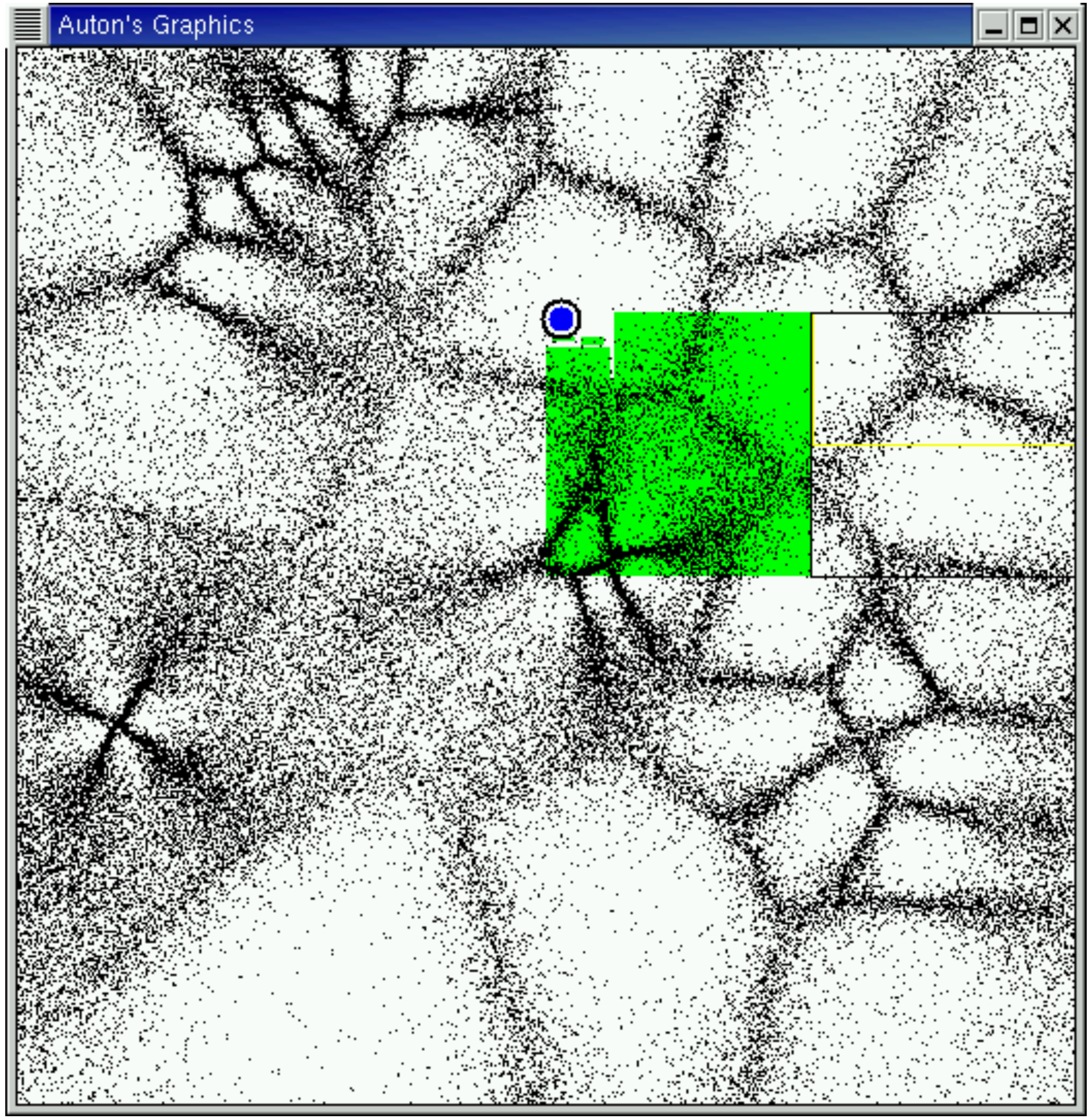

Auton's Graphics

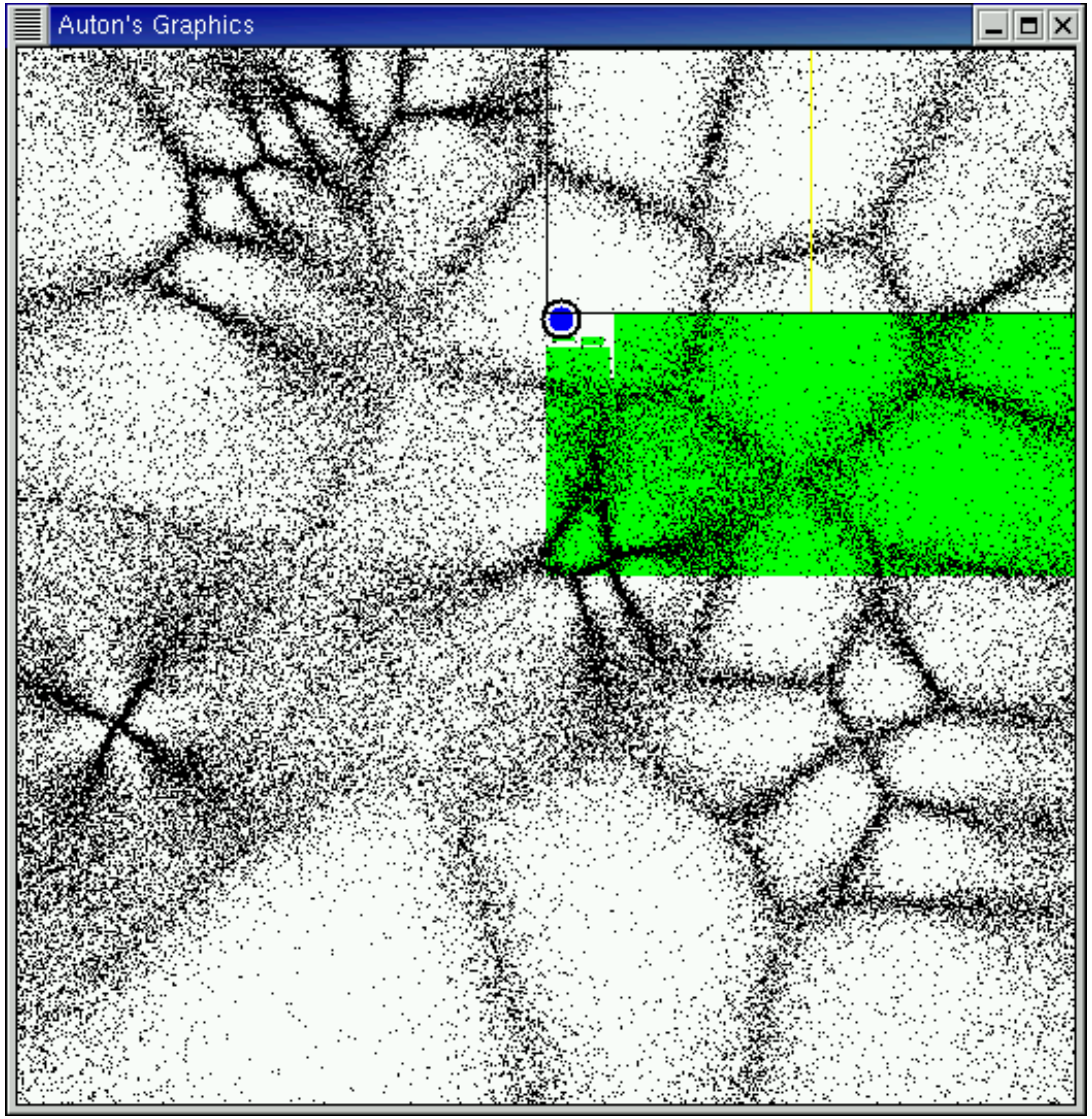

Auton's Graphics

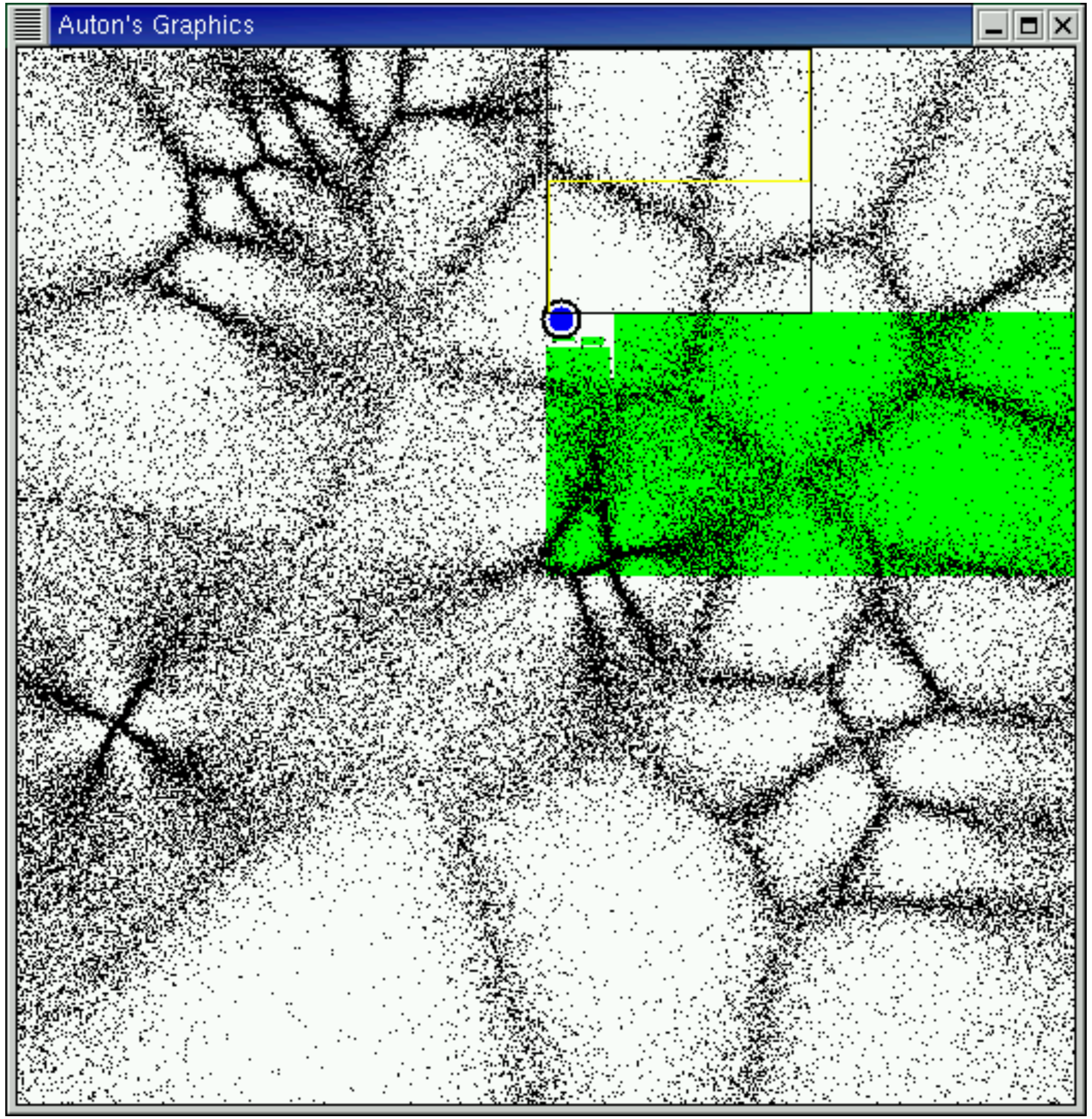

Auton's Graphics

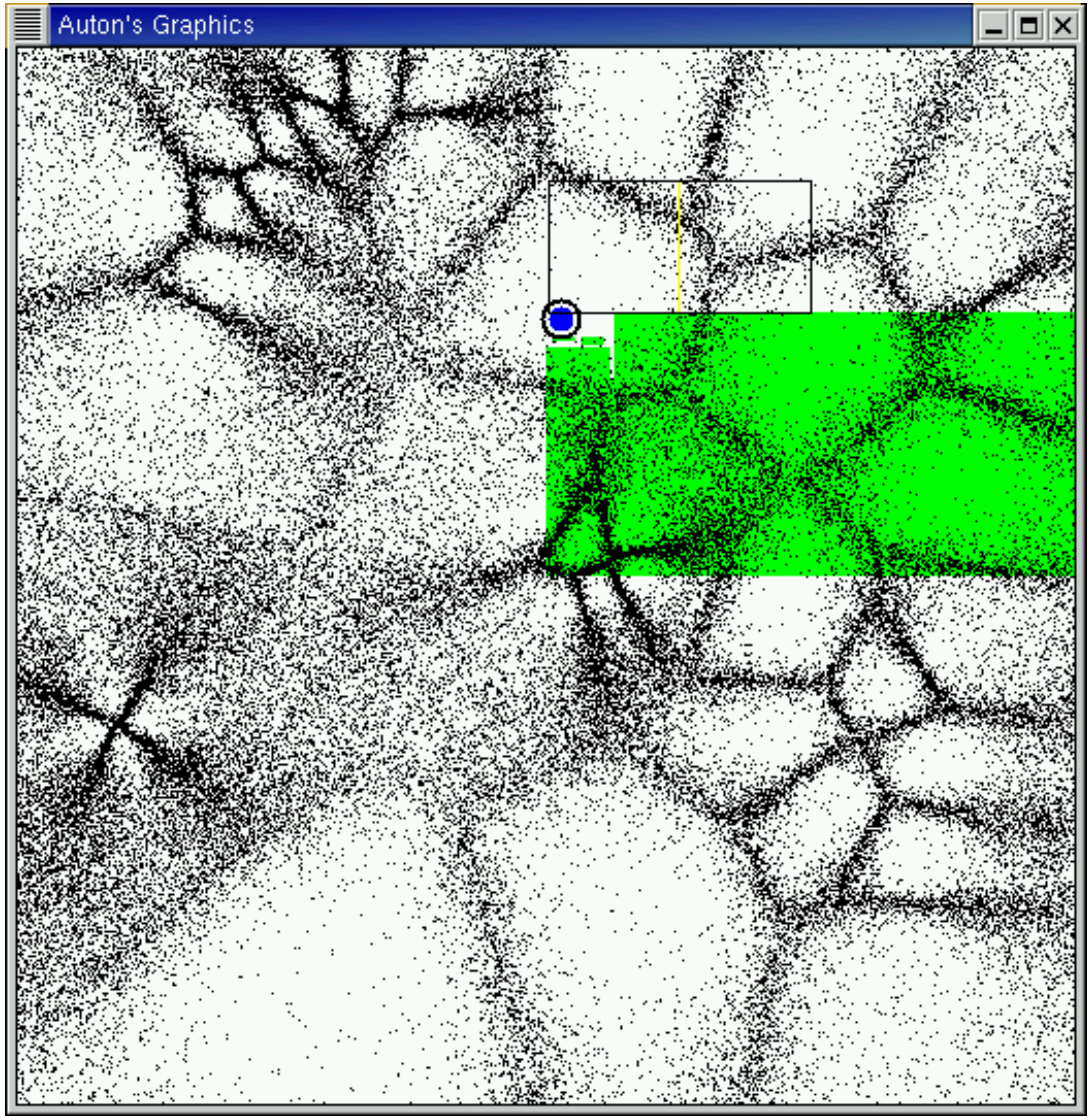

Auton's Graphics

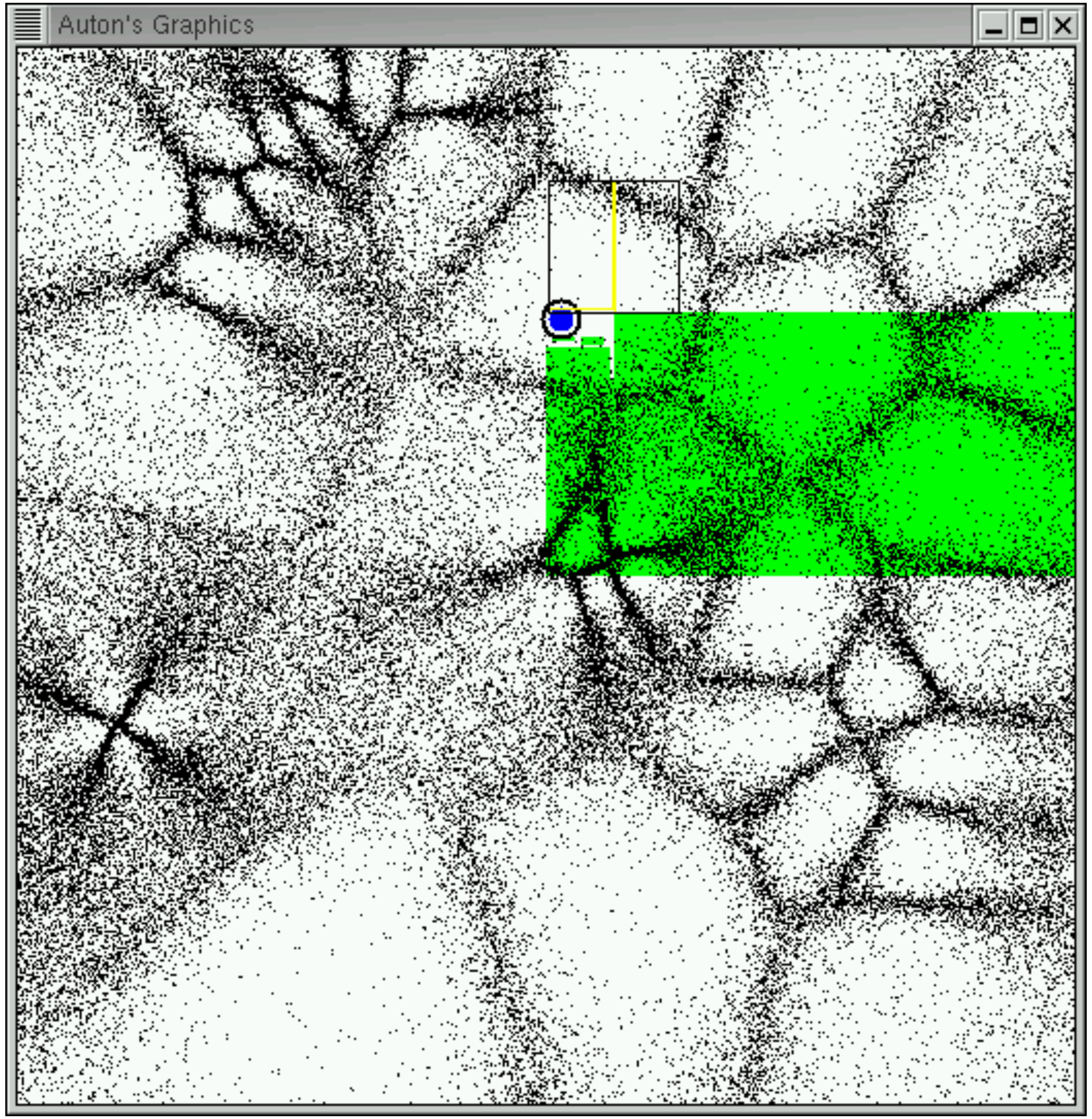

Auton's Graphics

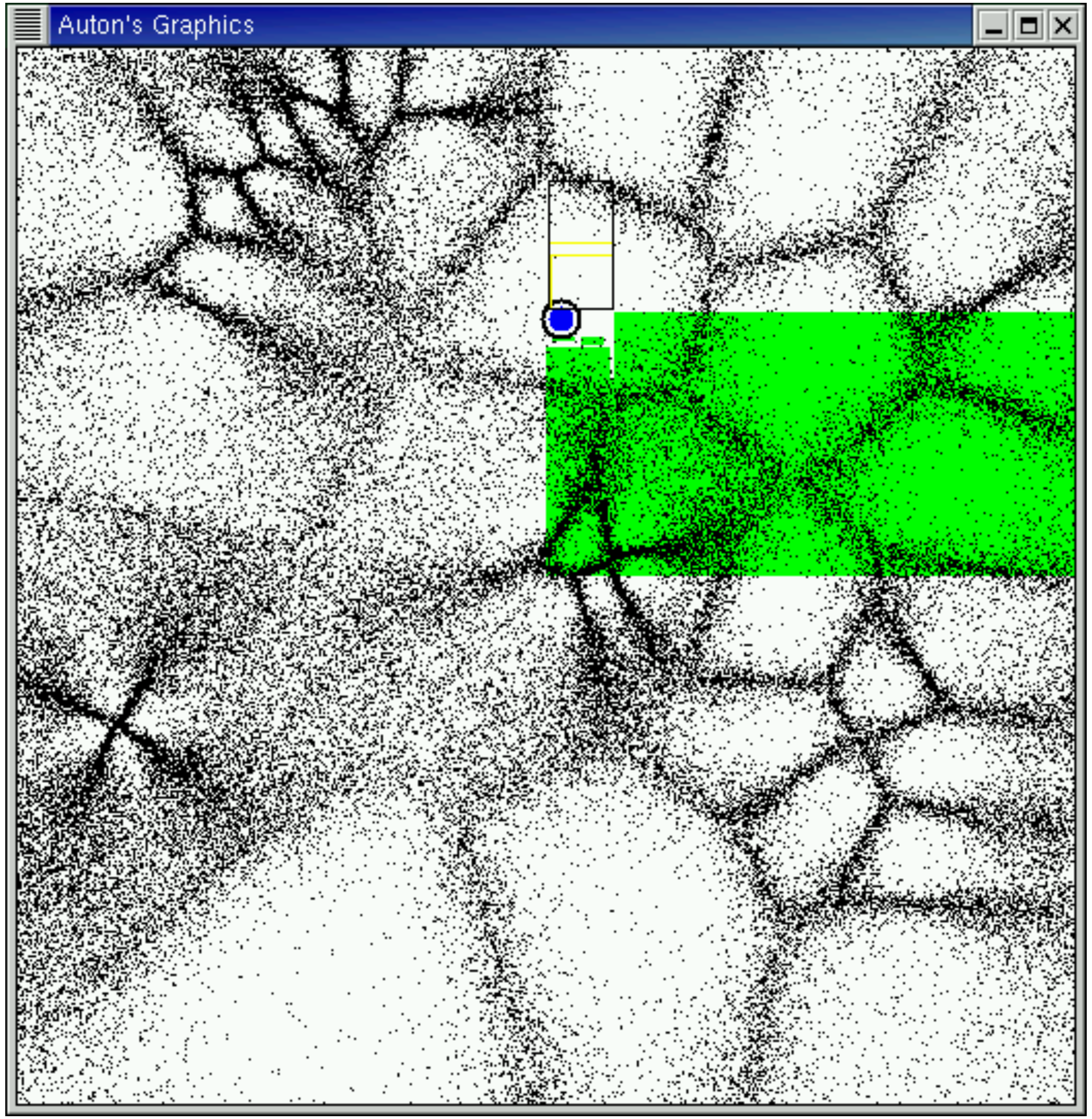

Auton's Graphics

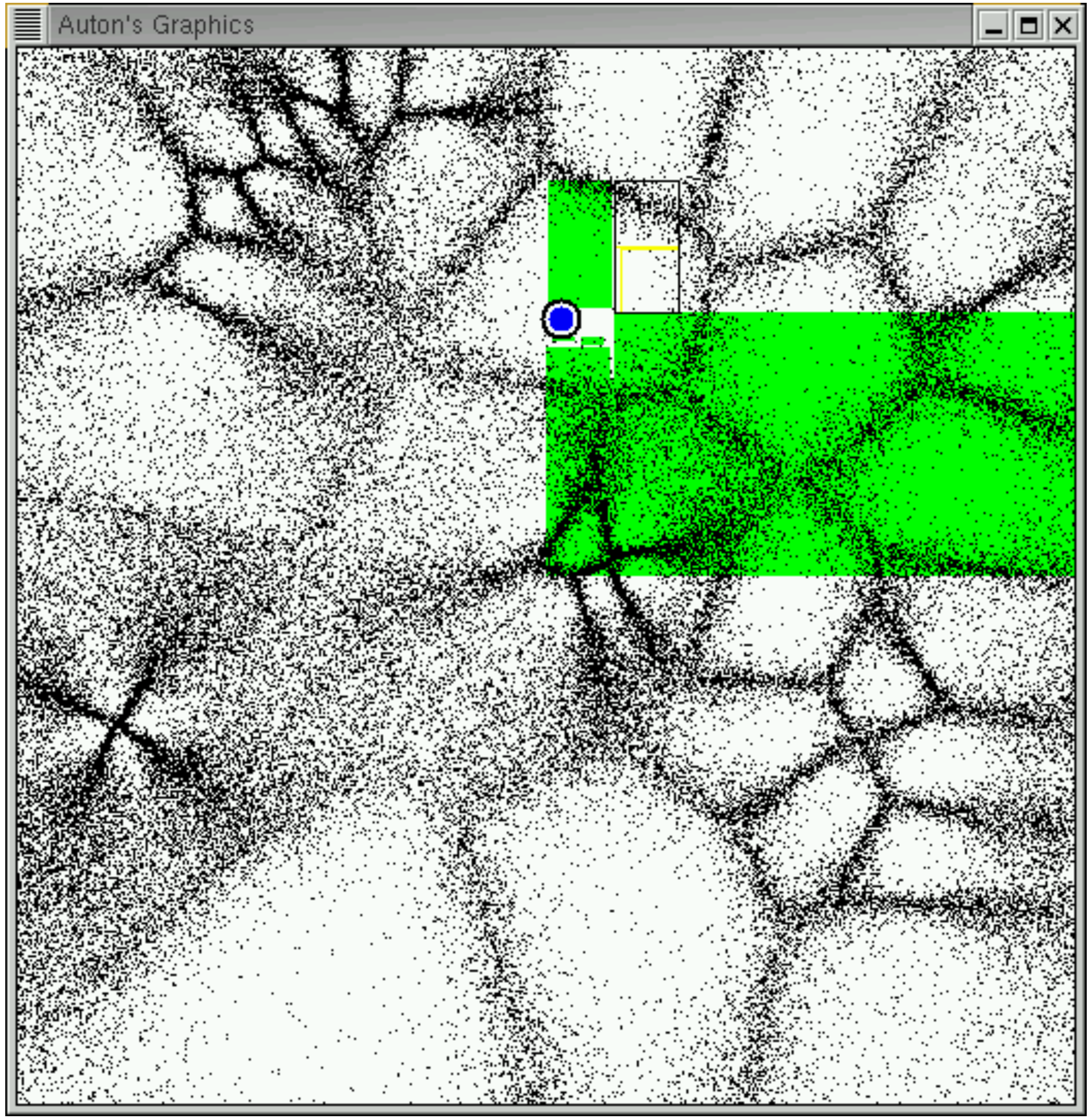

Auton's Graphics

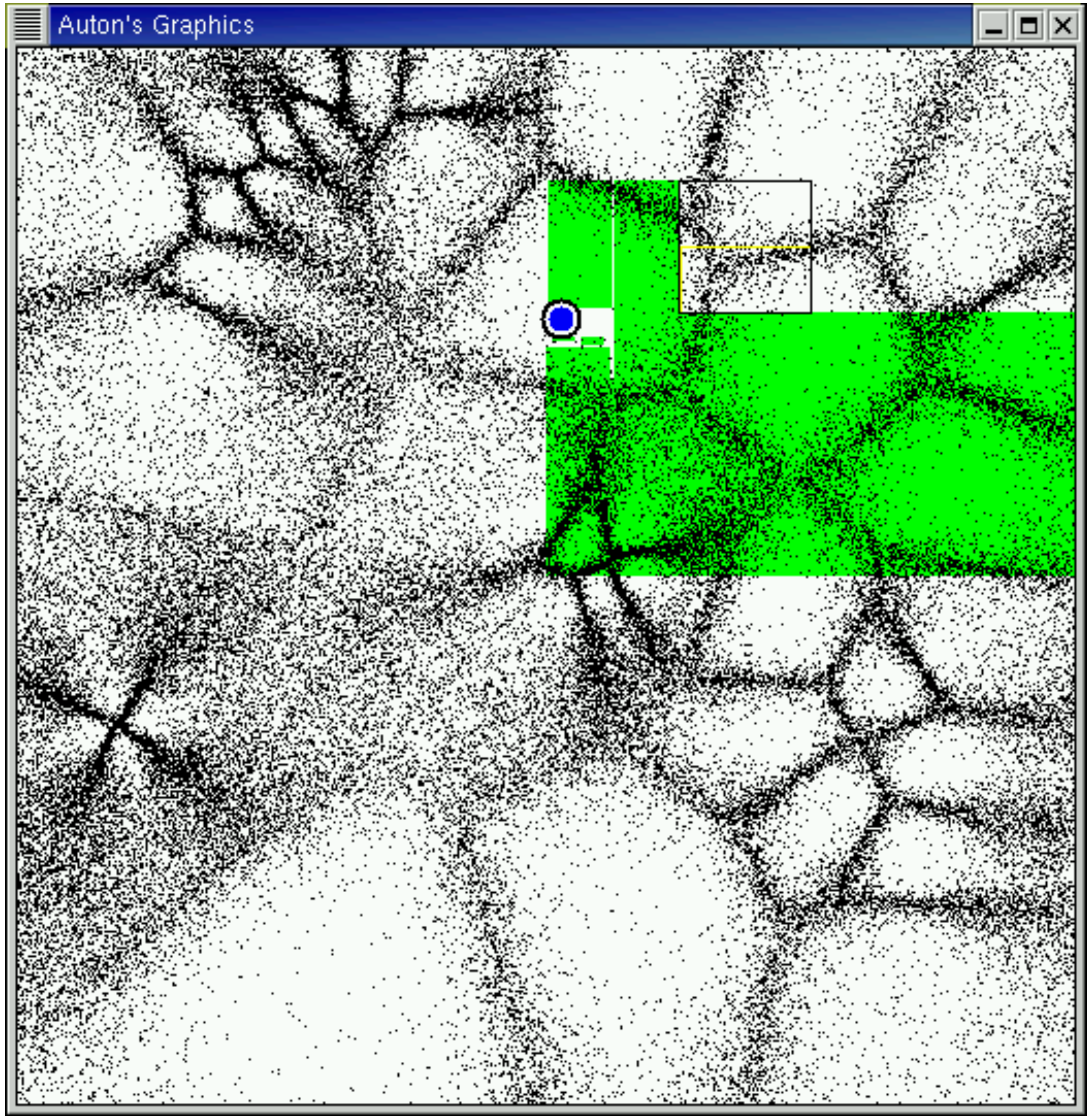

Auton's Graphics

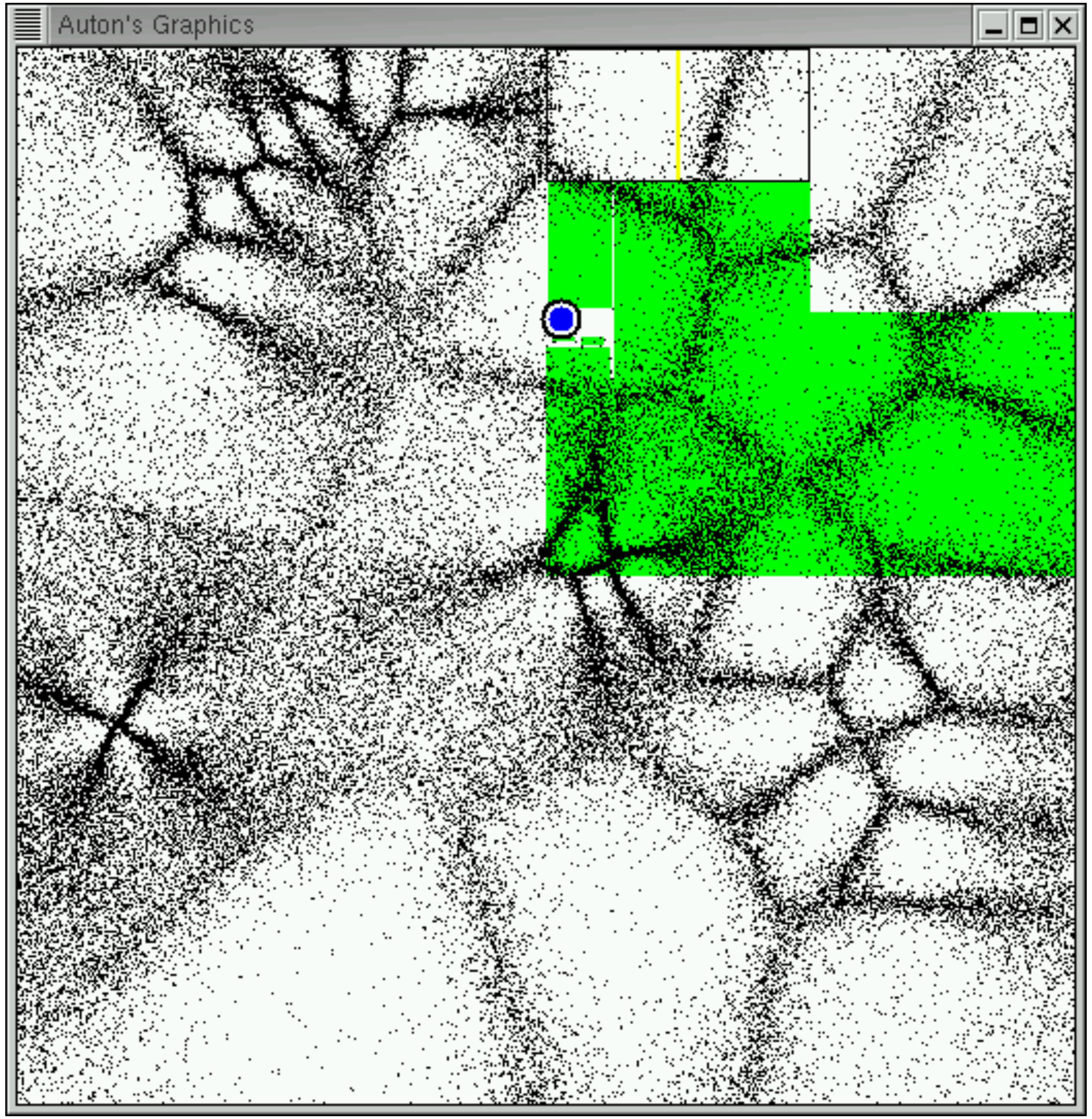

Auton's Graphics

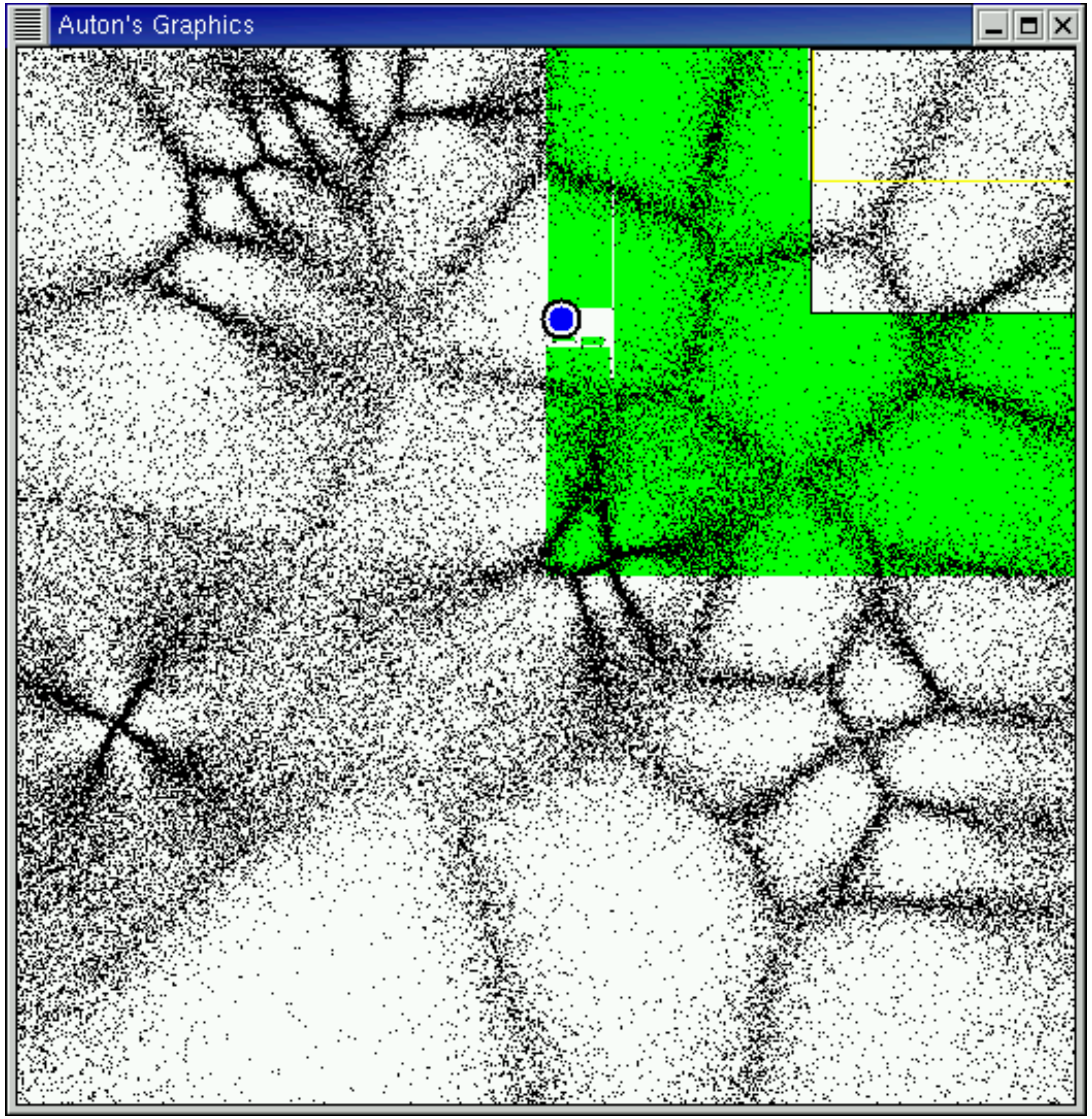

Auton's Graphics

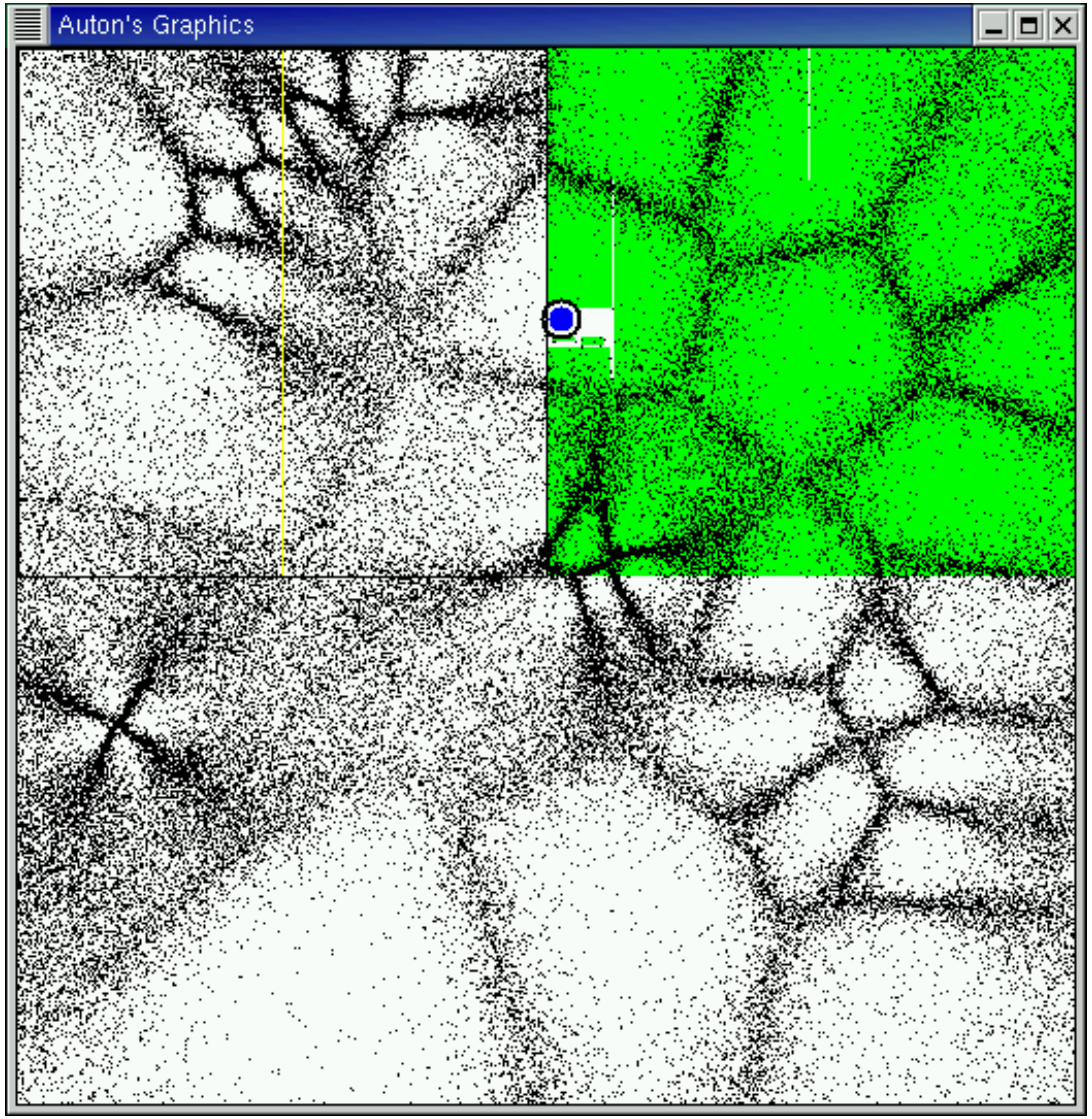

Auton's Graphics

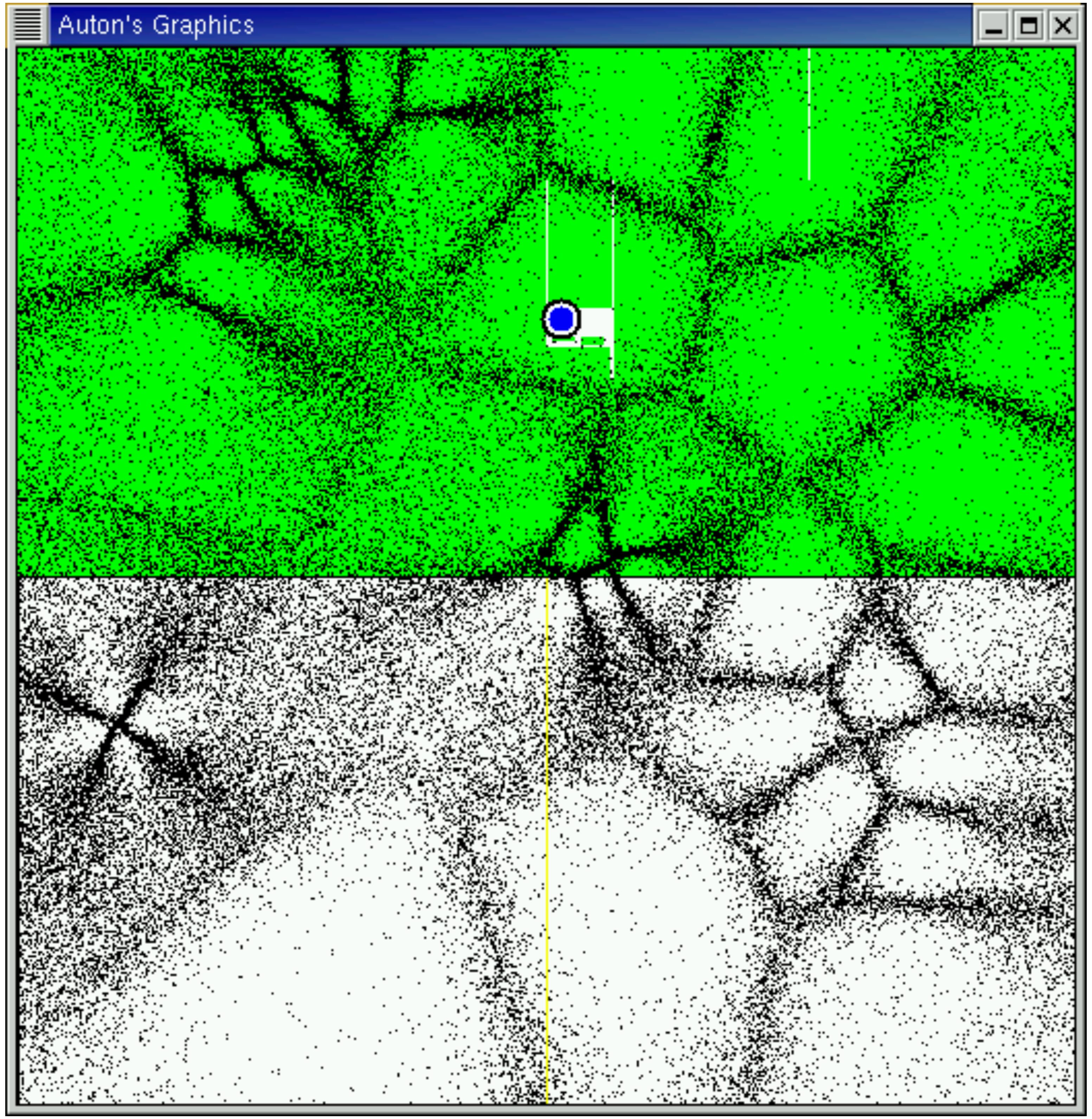

Auton's Graphics

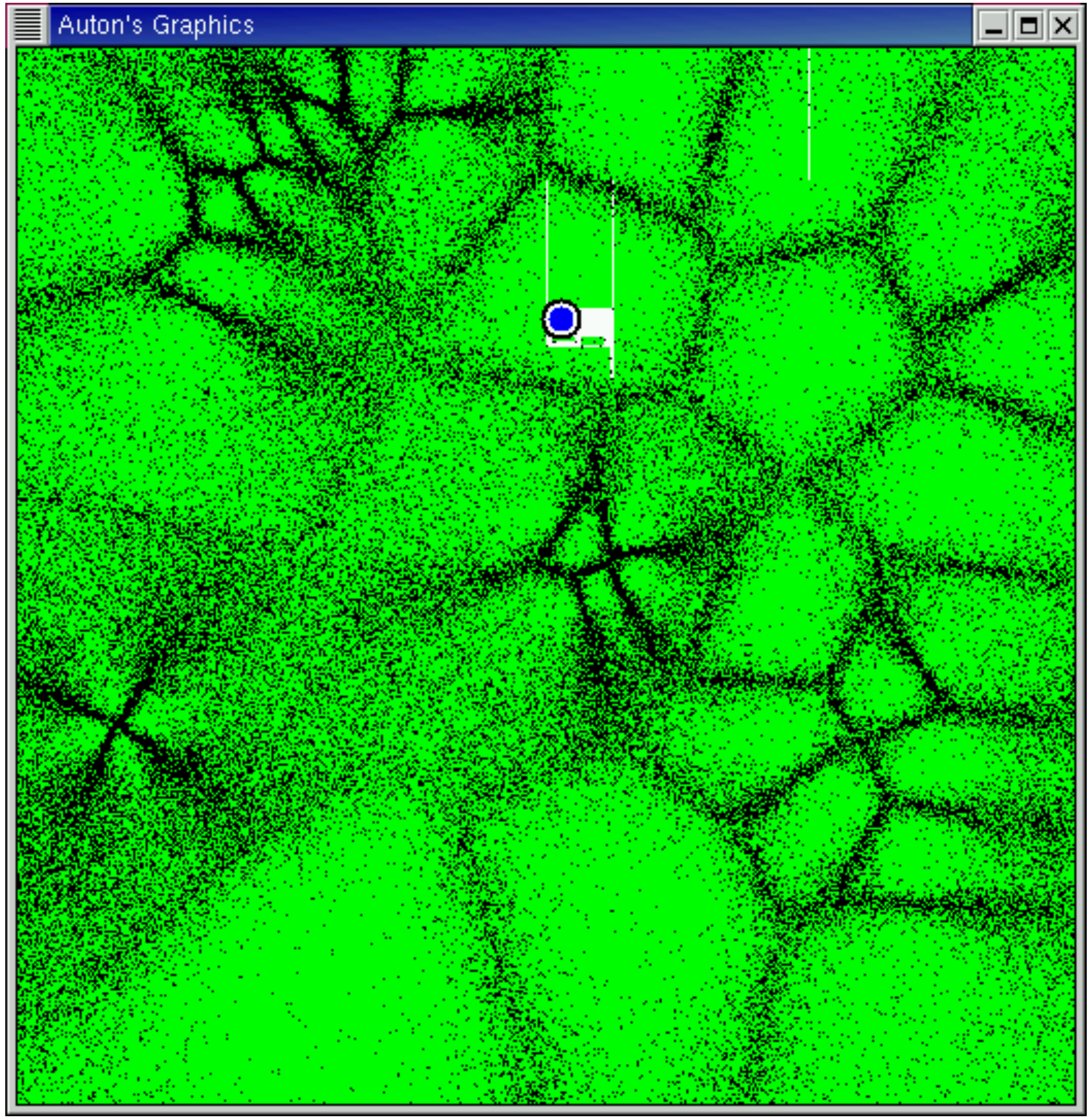

Auton's Graphics


Auton's Graphics


Auton's Graphics

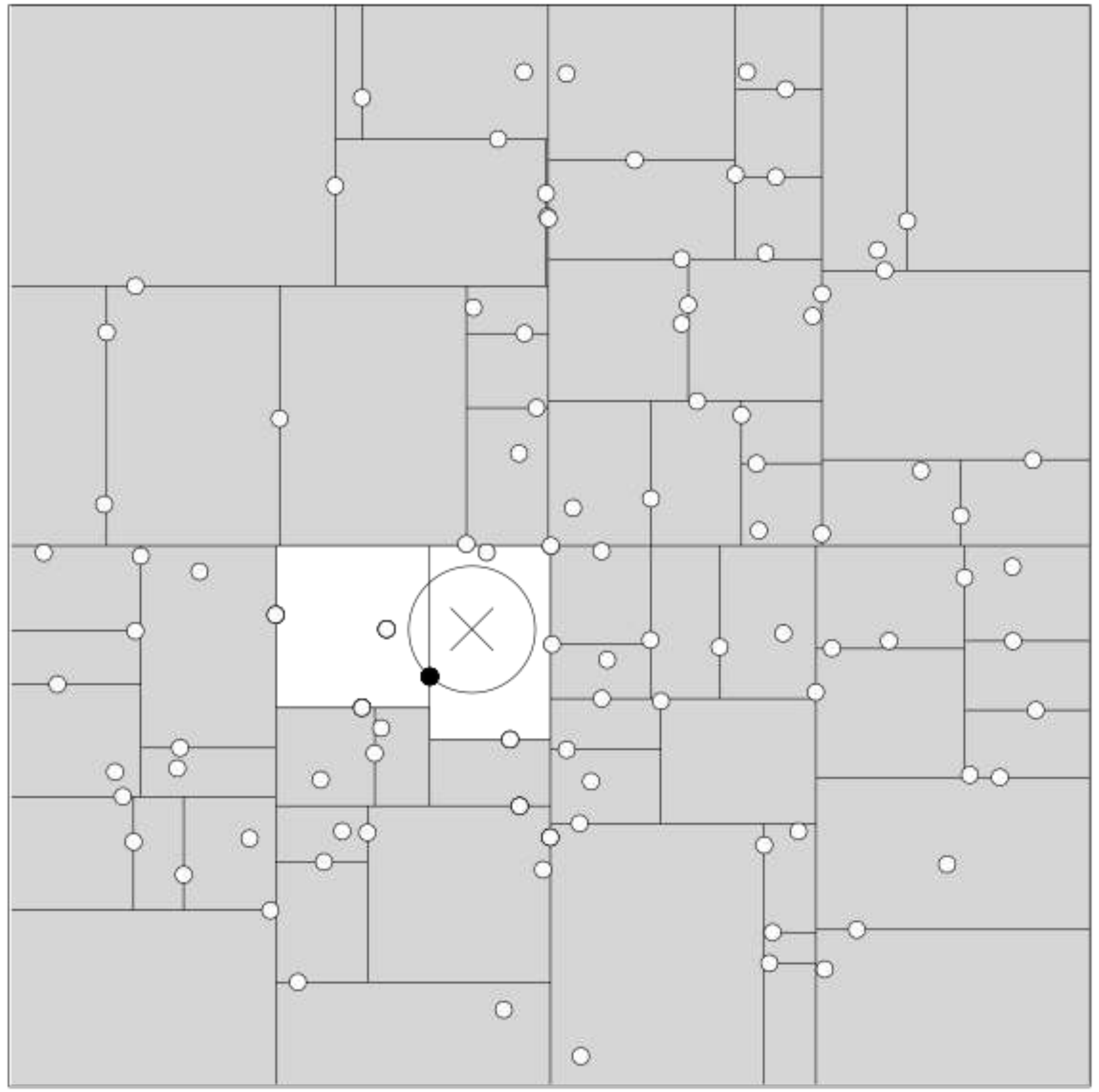

Auton's Graphics


Auton's Graphics

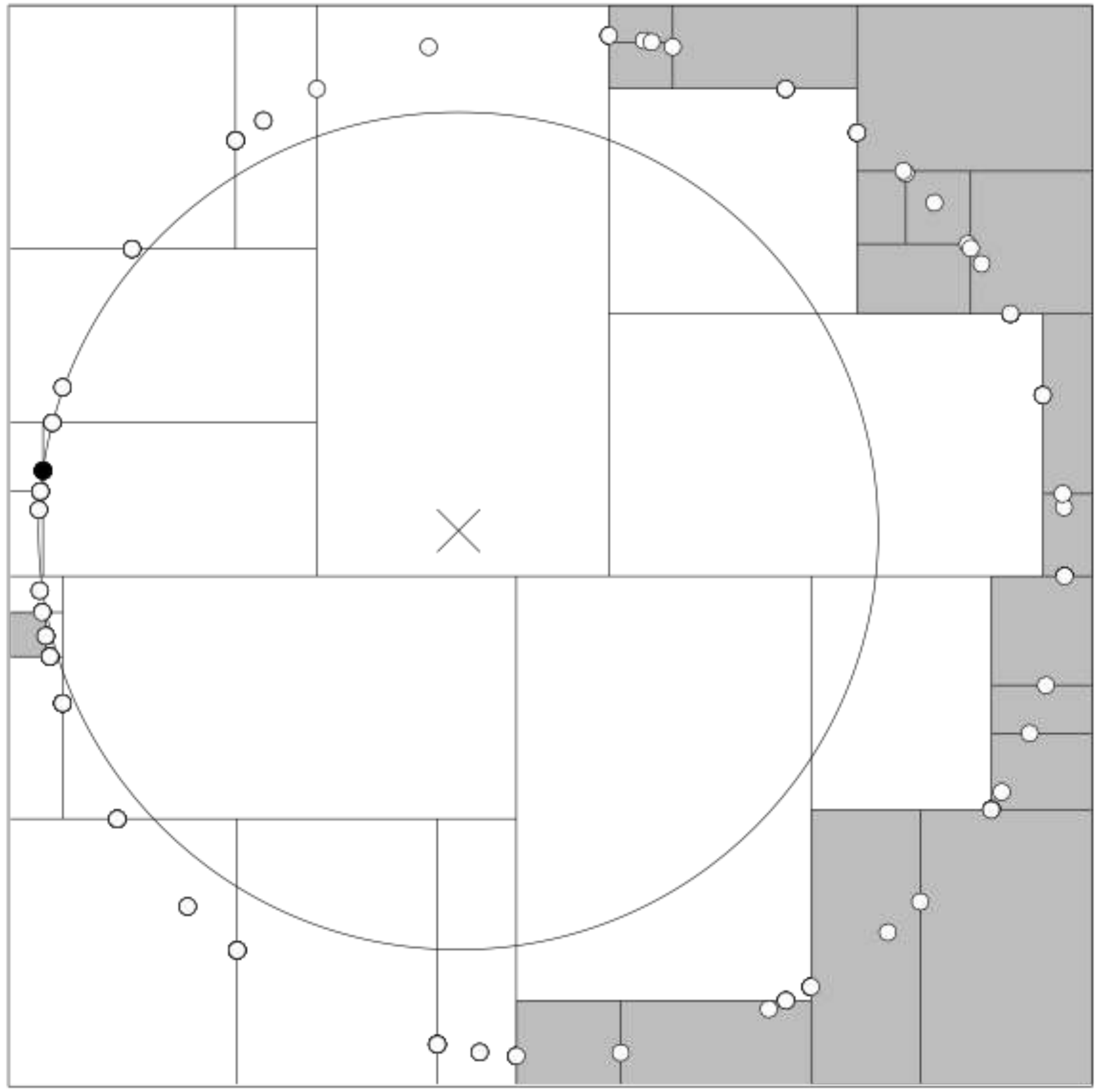

Auton's Graphics

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

Good case



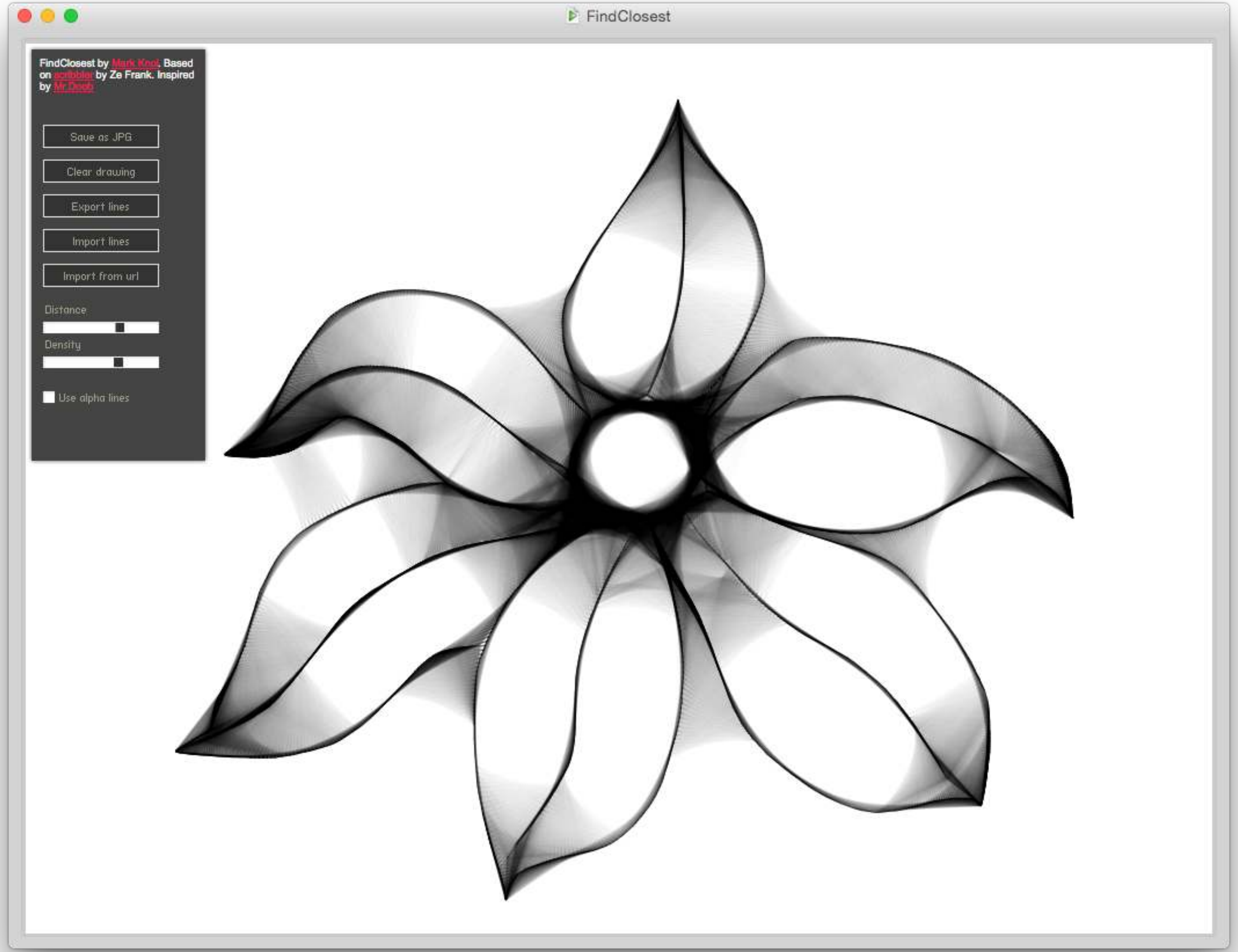
Bad case



All leaves the NN algorithm had to visit are shown in white!

In a few moments, it will get worse …

Artistic Application of k-NN Algorithm



Application: Classification

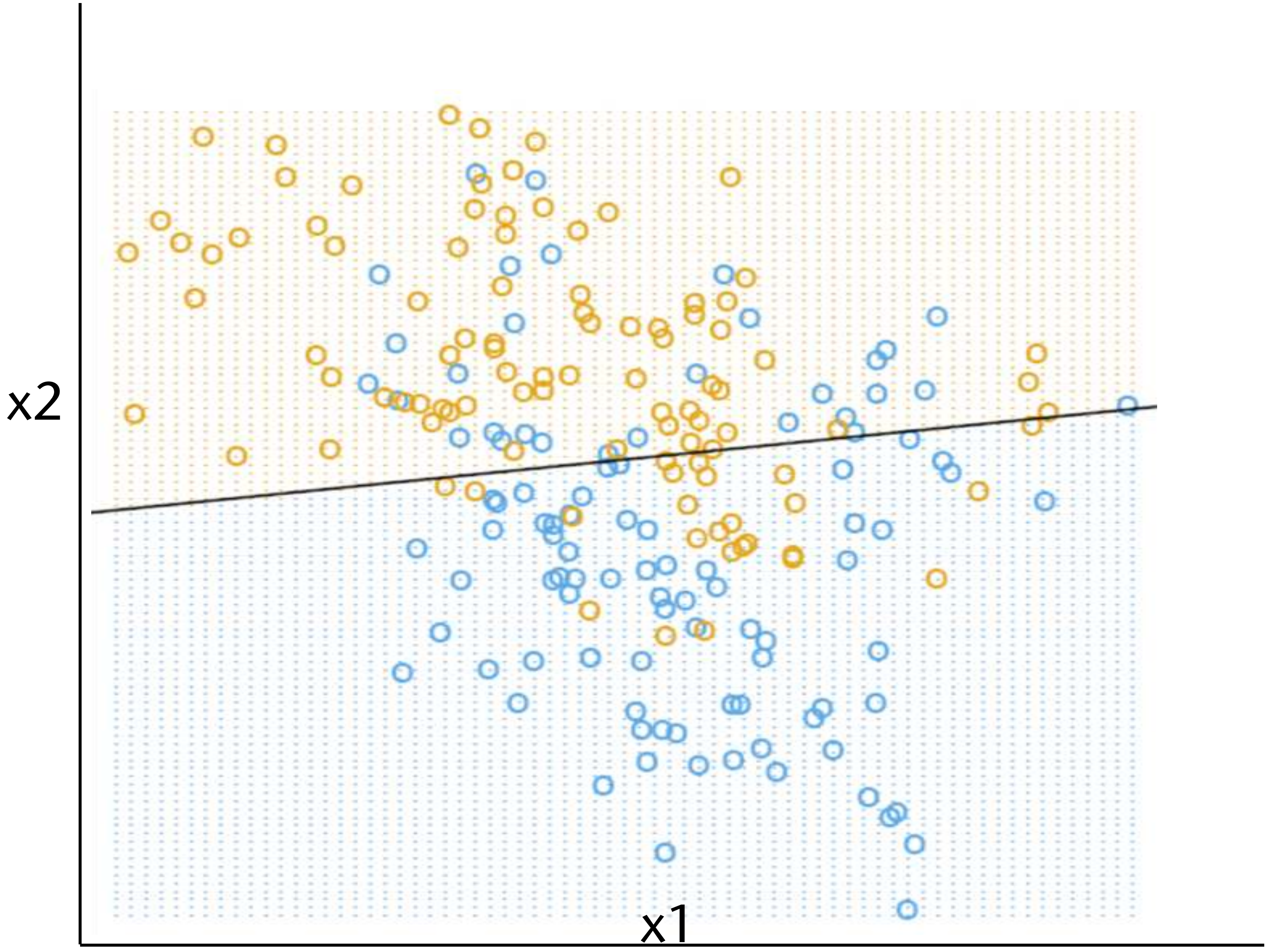
- Given a set of points $\mathcal{L} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^d$
  and for each such point a label $y_i \in \{l_1, l_2, \ldots, l_n\}$
  - Each label represents a class, all points with the same label are in the *same class*
- Wanted: a method to decide for a *not-yet-seen* point x which label it most probably has, i.e., a method to *predict class* labels
  - We say that we learn a classifier C from the training set $\mathcal{L}$:

$$C : \mathbb{R}^d \rightarrow \{l_1, l_2, \ldots, l_n\}$$

- Typical applications:
  - Computer vision (object recognition, ...)
  - Credit approval
  - Medical diagnosis

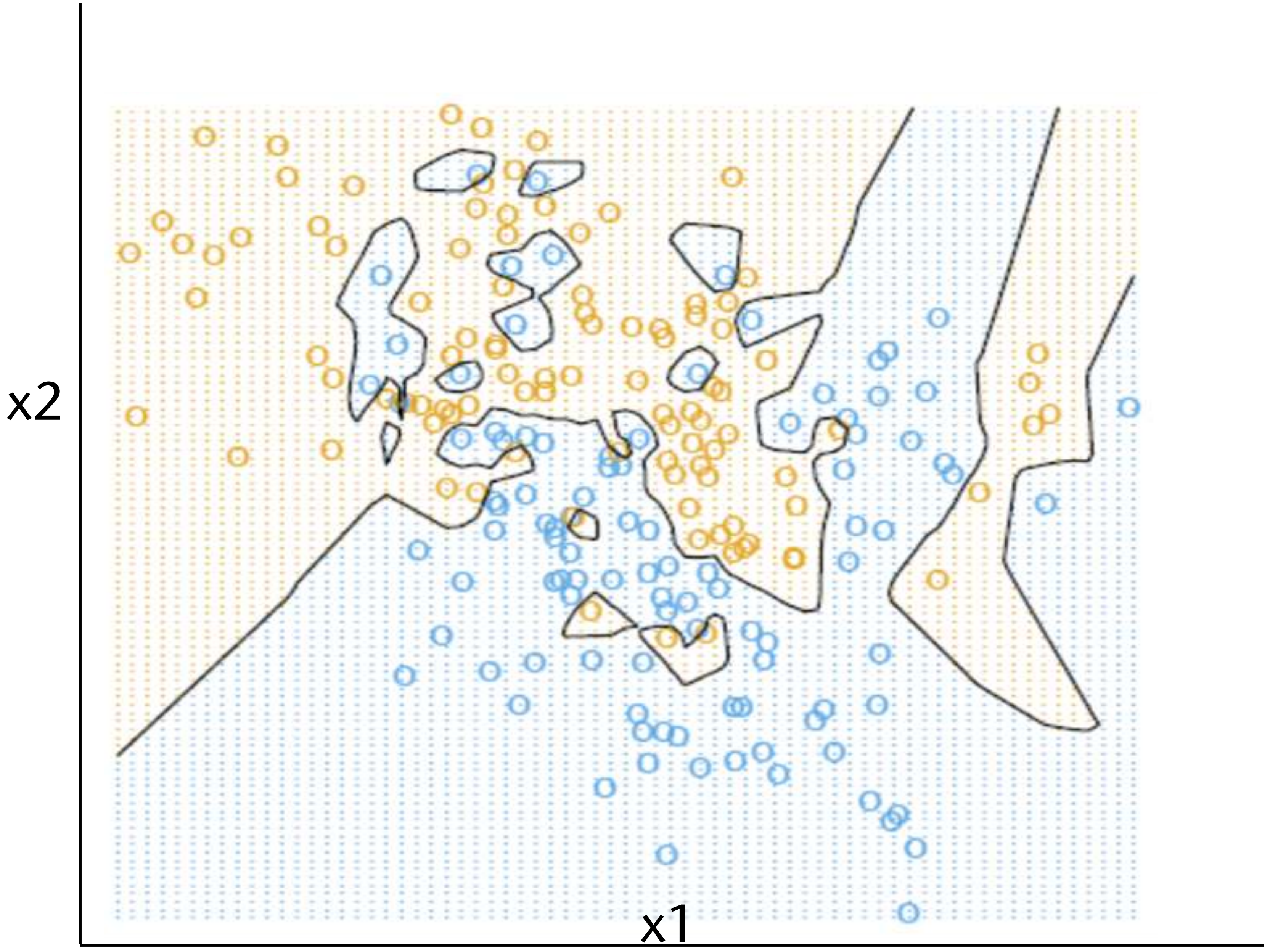
One Possible Solution: Linear Regression

- Assume we have only two classes (e.g., "blue" and "yellow")
- Fit a plane through the data



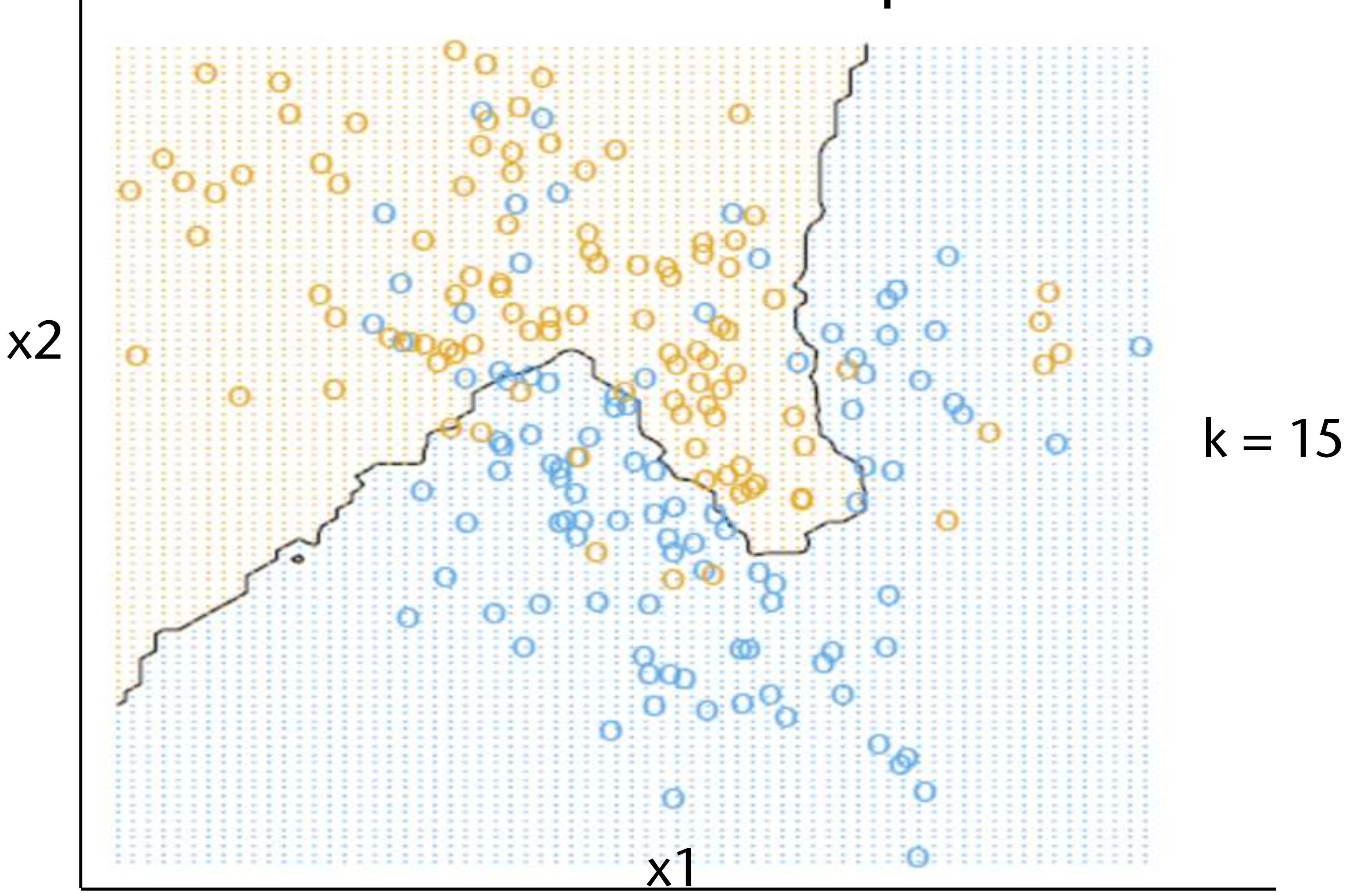
Another Solution: Nearest Neighbor (NN) Classification

- For the query point q, find the nearest neighbor $\mathbf{x}^* \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^d$
- Assign the class l^* to $\mathbf{x}$



Improvement: k-NN Classification

- Instead of the 1 nearest neighbor, find the k nearest neighbors of q, $\{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}\} \subset \mathcal{L}$
- Assign the majority of the labels $\{l_{i_1}, \ldots, l_{i_k}\}$ to q



Thinking About Higher-Dimensional Space: Slicing

"Flatland" by Edwin A. Abbott, presented by Carl Sagan

4-Dimensional Space

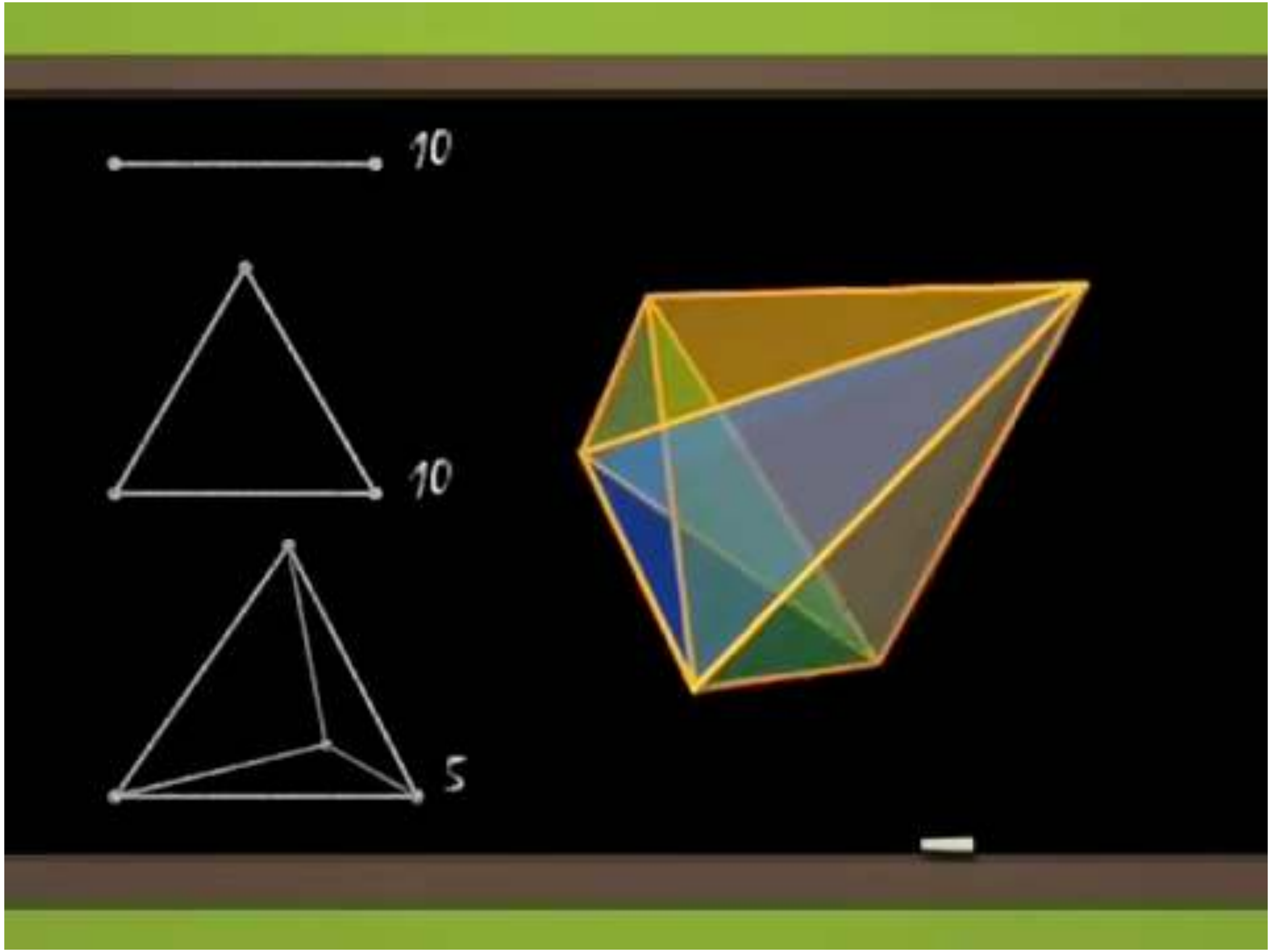
- Brain teaser: what does a cube that slowly "floats" through Flatland look like, starting with a corner?
- What can a higher-dimensional being do with lower-dimensional beings:

Thinking About 4D: Analogy and Slicing, Example: 4-Dim. Tetrahedron

1-simplex

2-simplex

3-simplex



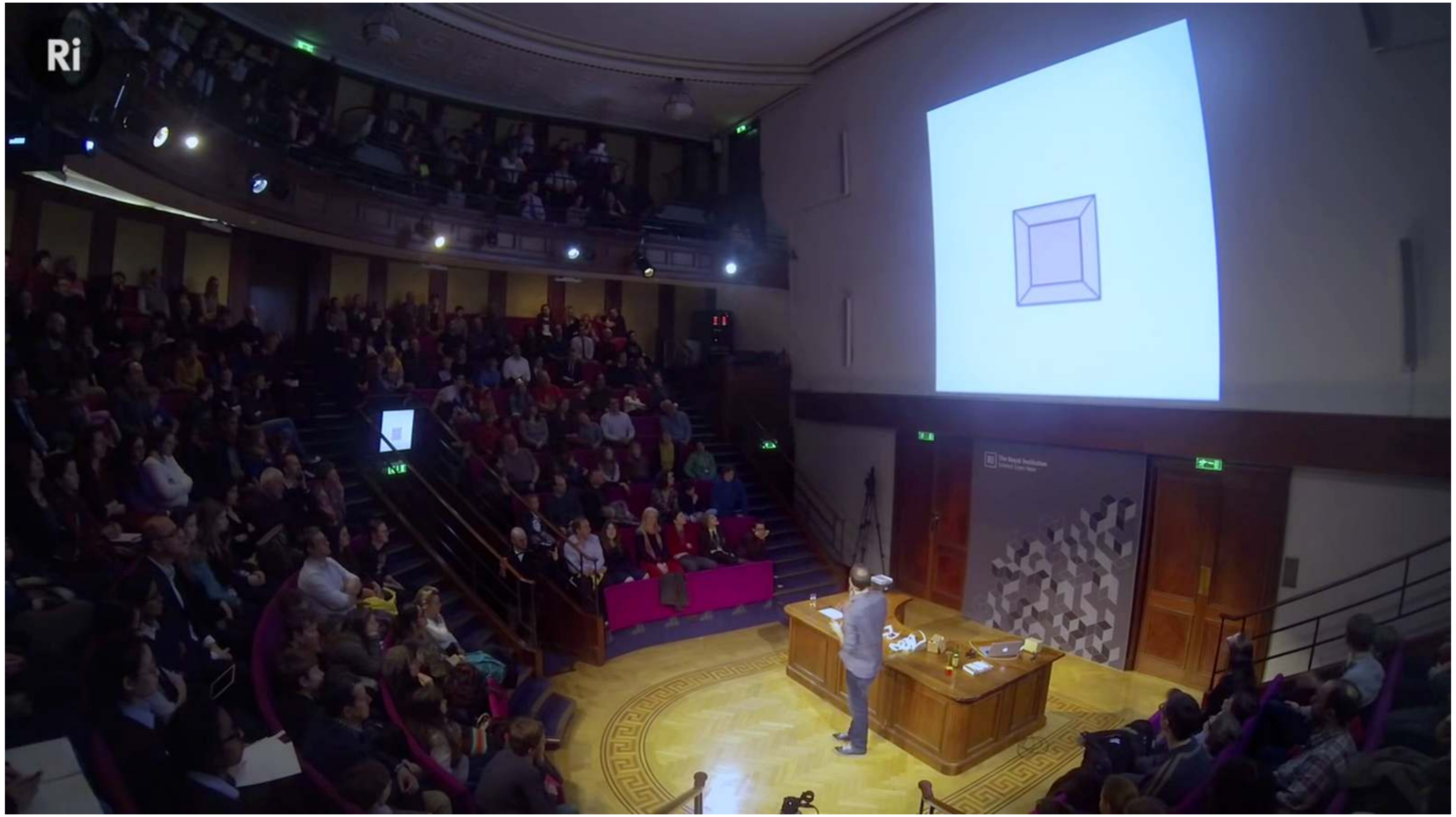
4-simplex

http://www.dimensions-math.org

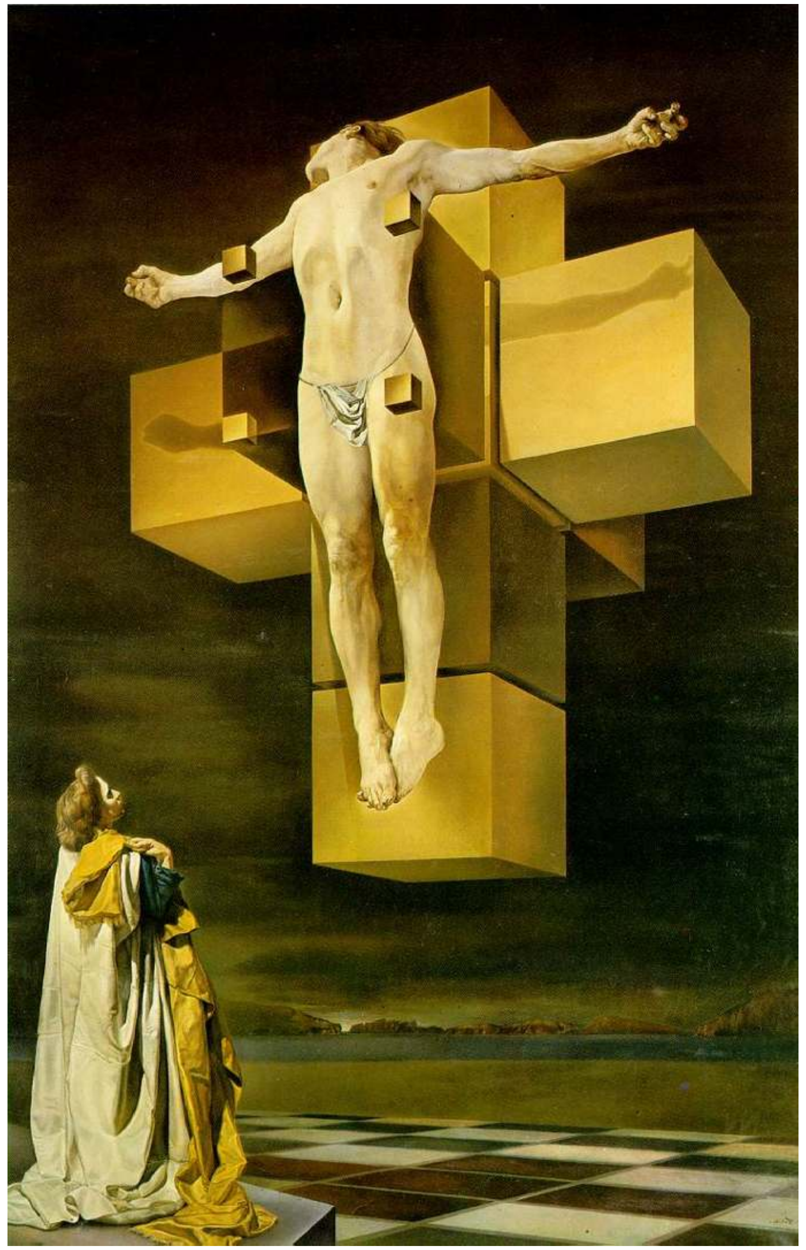
Thinking About 4D: the Projection Method, Example Hypercube (Tesseract)

Thinking About 4D: the Unwrapping Method

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



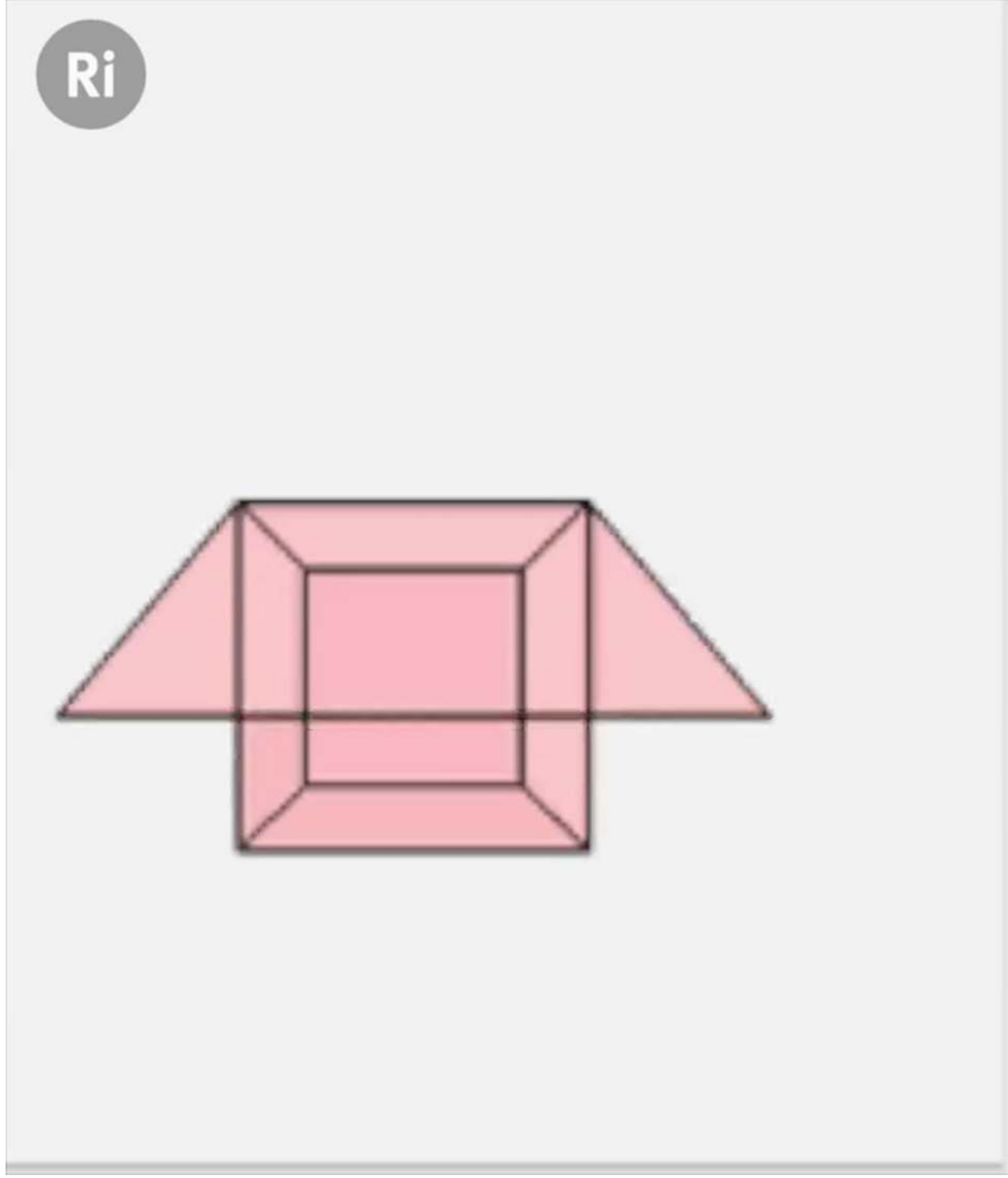
Matt Parker

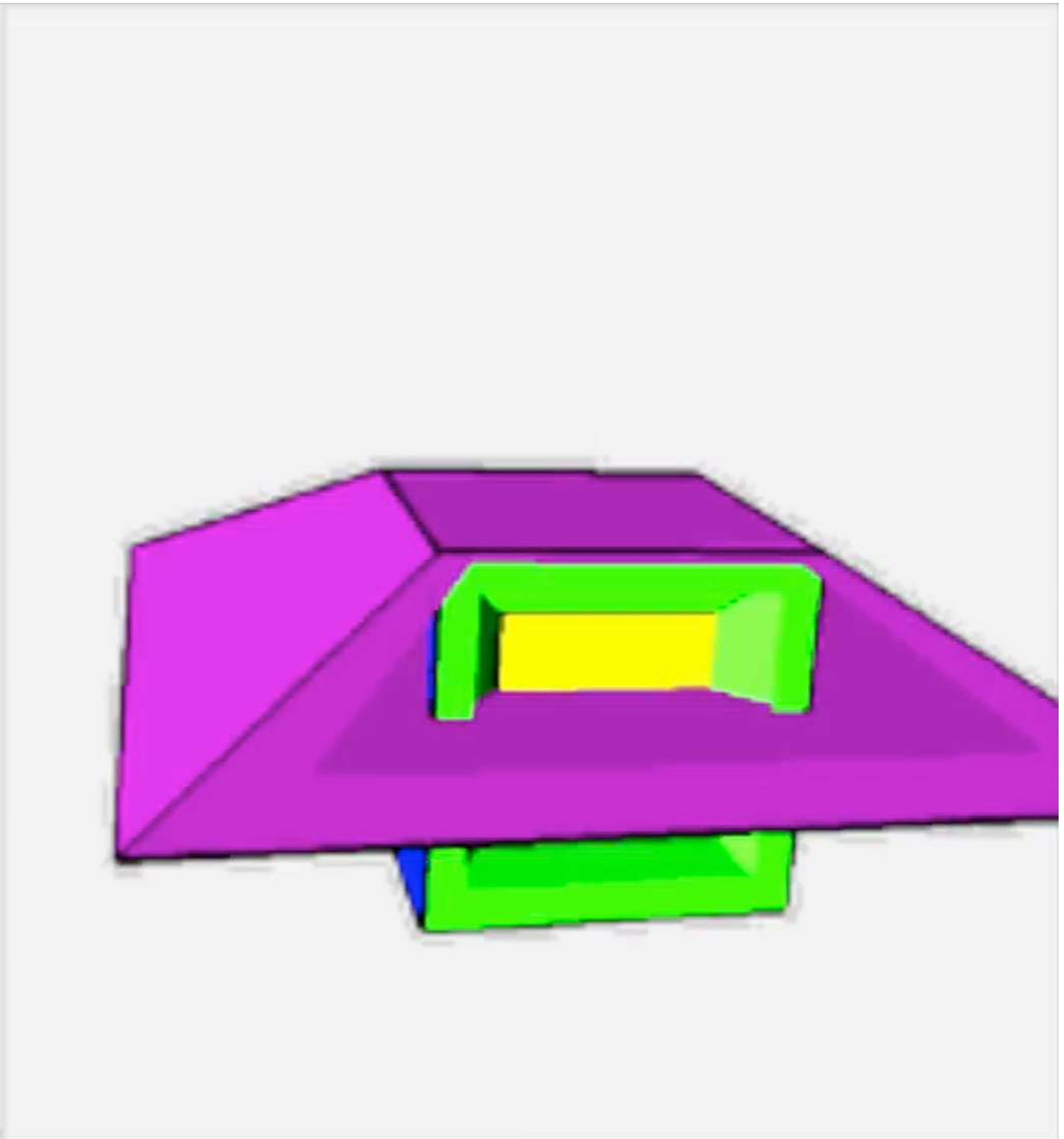


Crucifixion
(Corpus Hypercubus),
1954, Salvador Dali

Projection of a 3D cube unfolding into its 2D net

Projection of a 4D cube unfolding into its 3D net

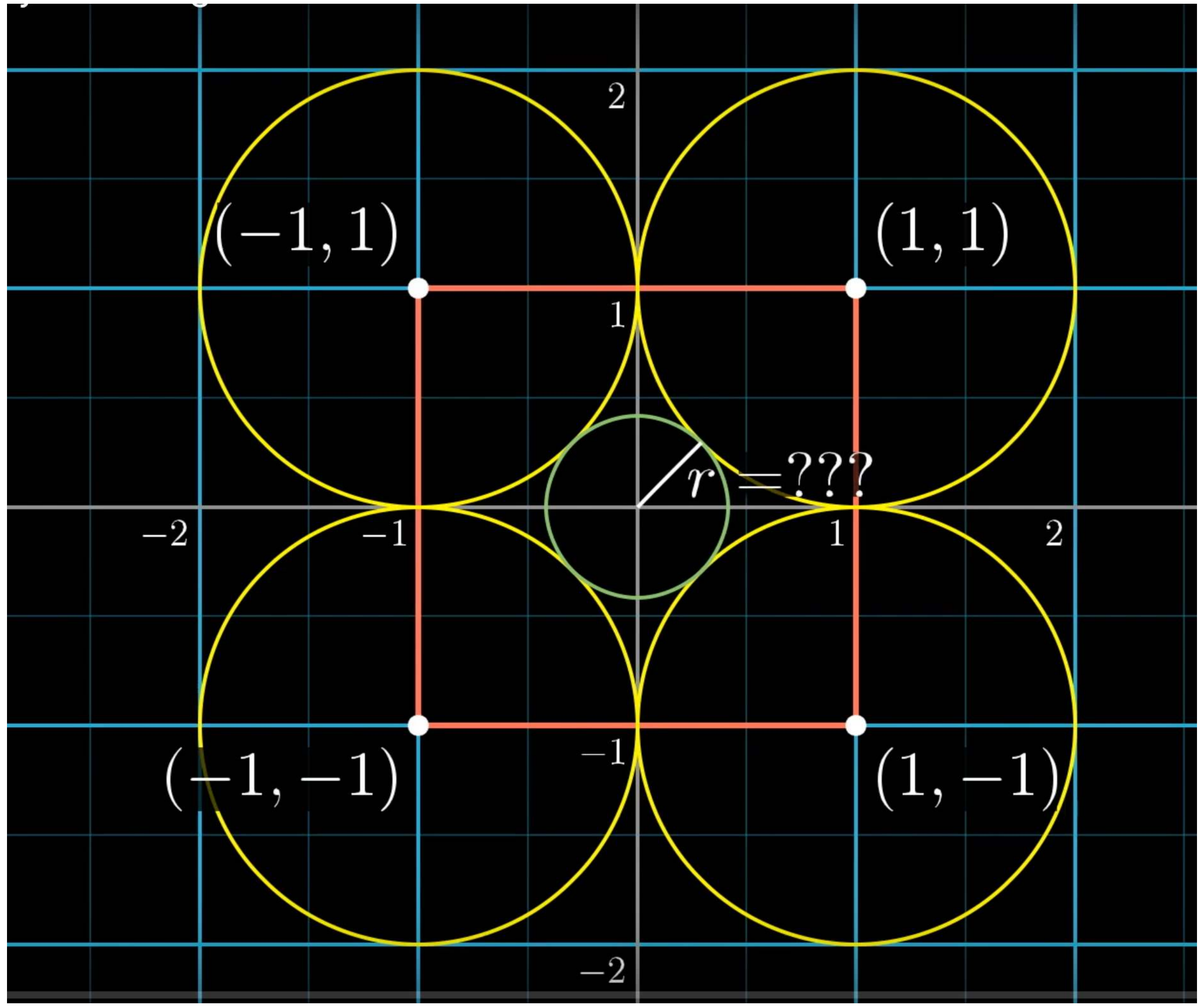


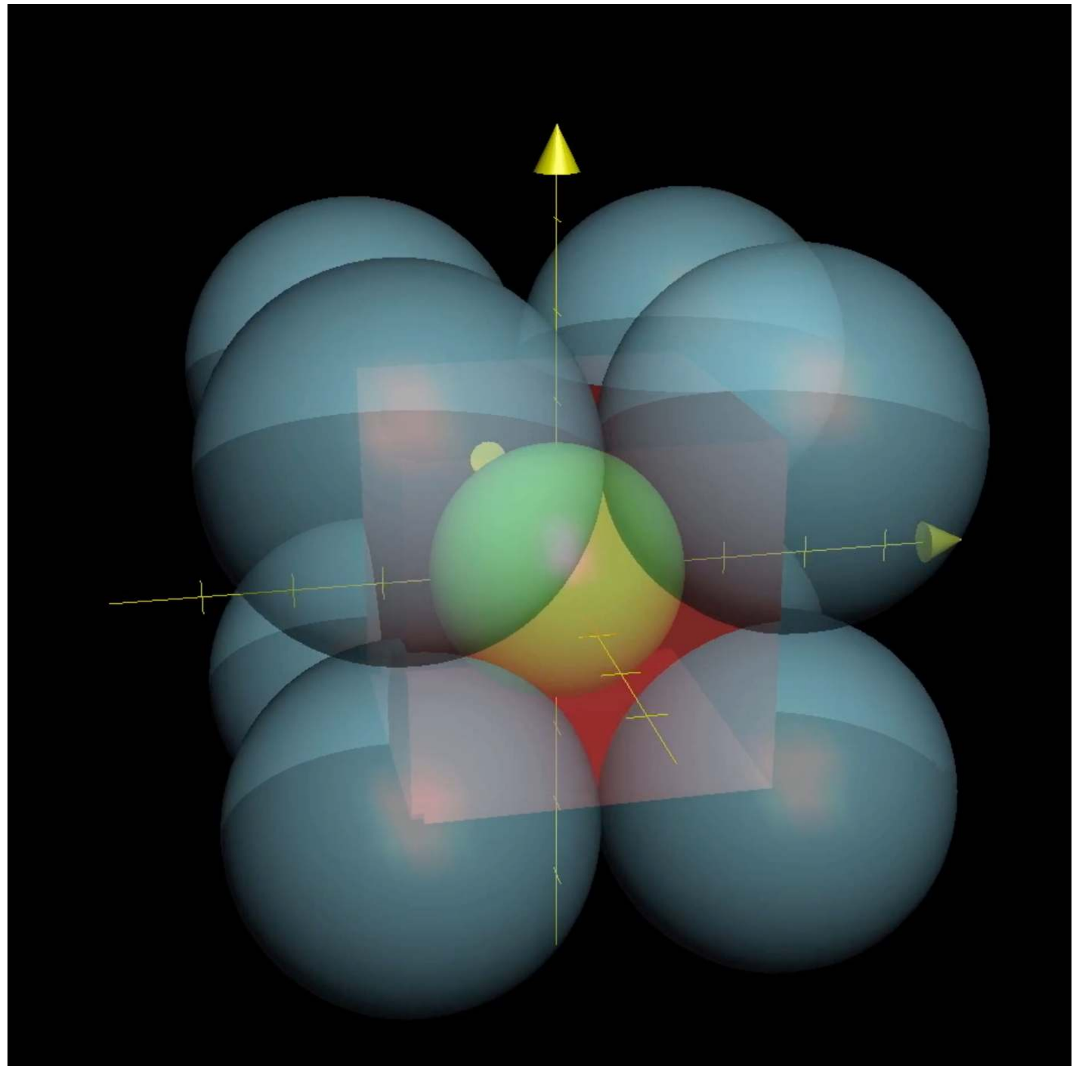


The "lid" of the 4D cube (what is it?) does not deform, of course; that is just an artefact of the projection into 3D, just like the lid of the 3D cube when projected into 2D.

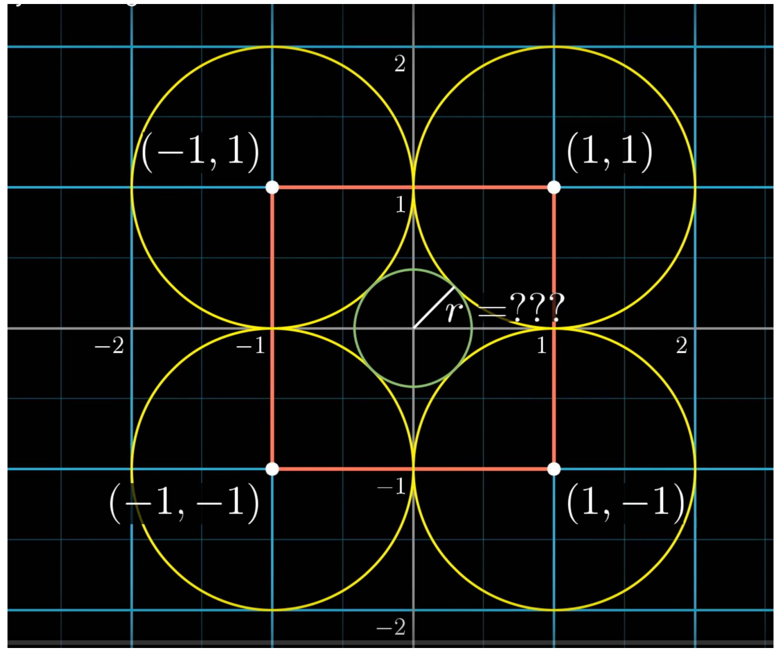
Strange Things Happen in Higher-Dimensional Space

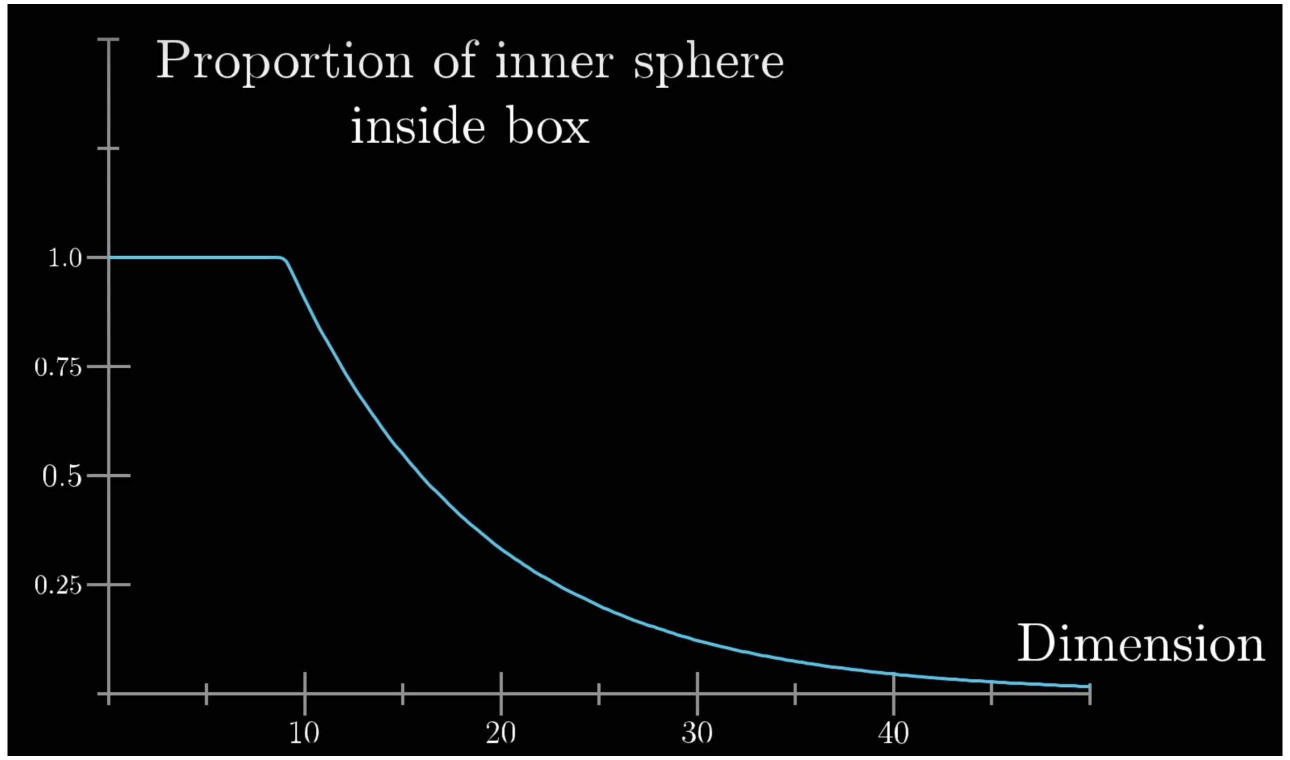
- Consider a cube $[-1,+1]^d$ with unit spheres centered at each corner
- What is the radius, *r*, of a sphere centered at the origin and just touching the corner spheres?



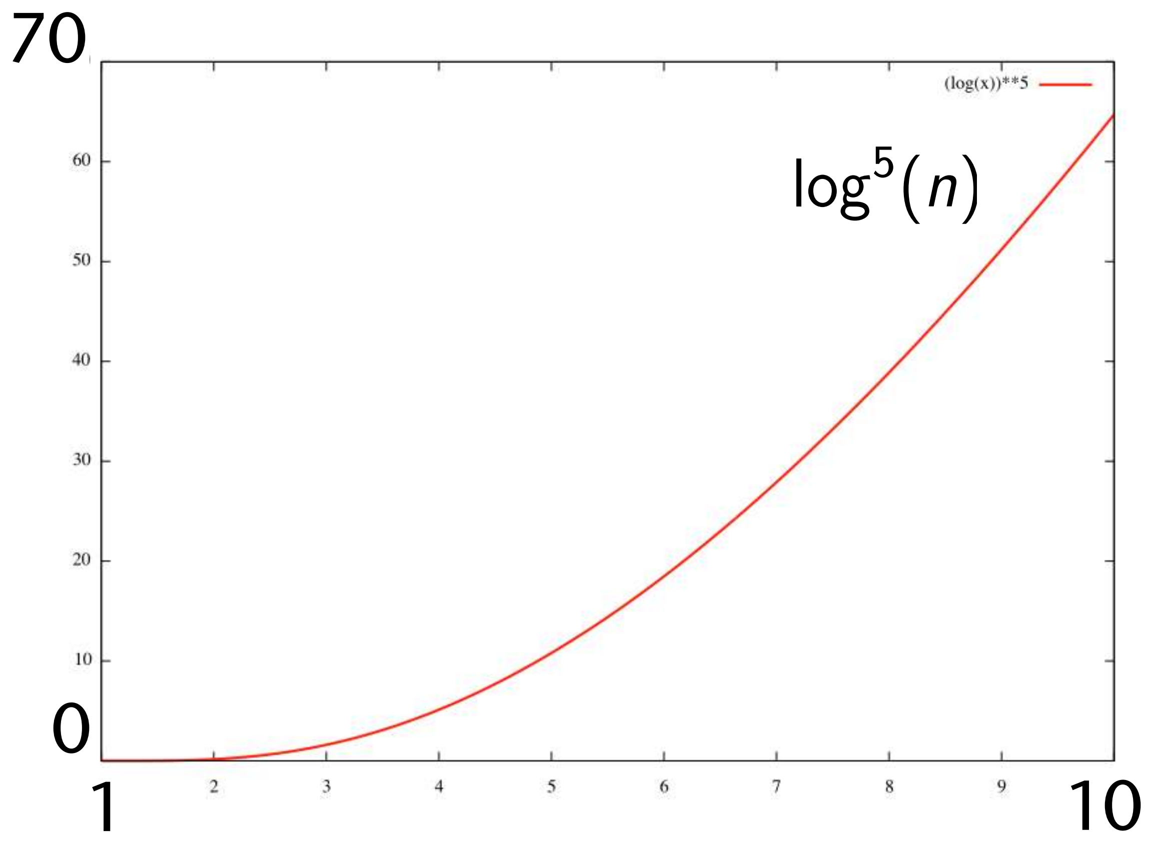


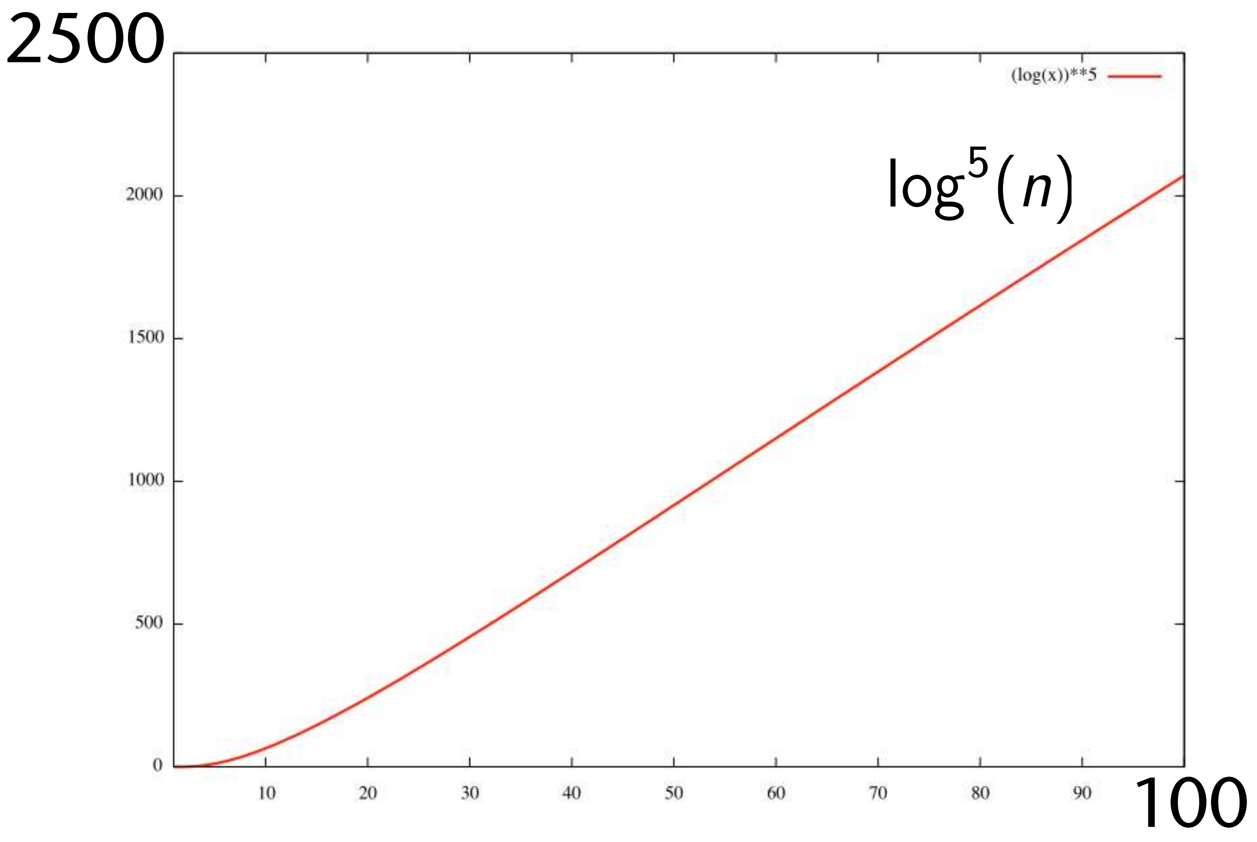
- Radius $r = \sqrt{d} - 1$
- In 2D: $r = 0.414$, in 3D: $r = 0.73$
- In 4D: $r = 1 \quad \longrightarrow$
  inner sphere touches box at the face centers!
- In 5D: $r = 1.24 \longrightarrow$ inner spheres sticks outside!
- In higher dimensions, more and more of the inner sphere is *outside* the box:

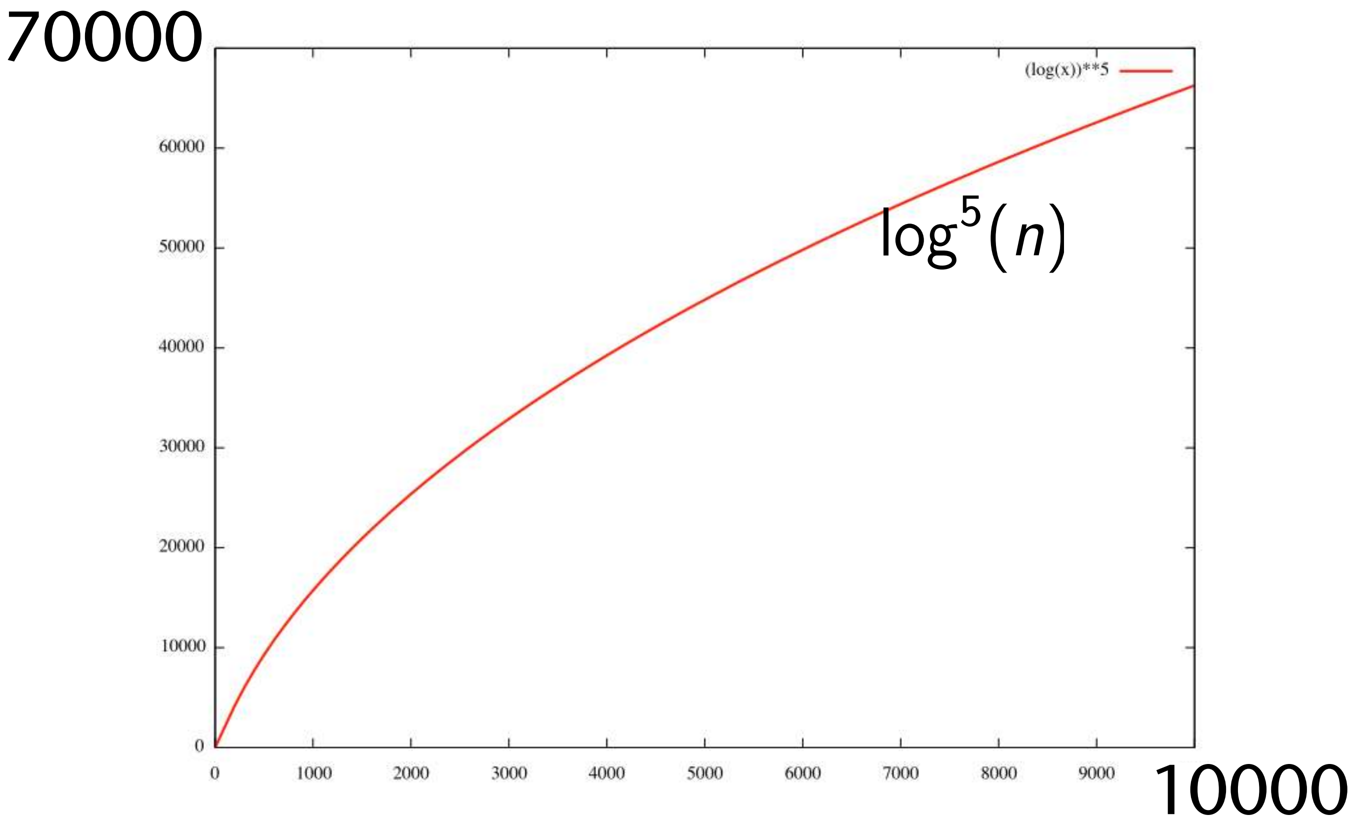




Zum Verhalten von $\log^d(n)$

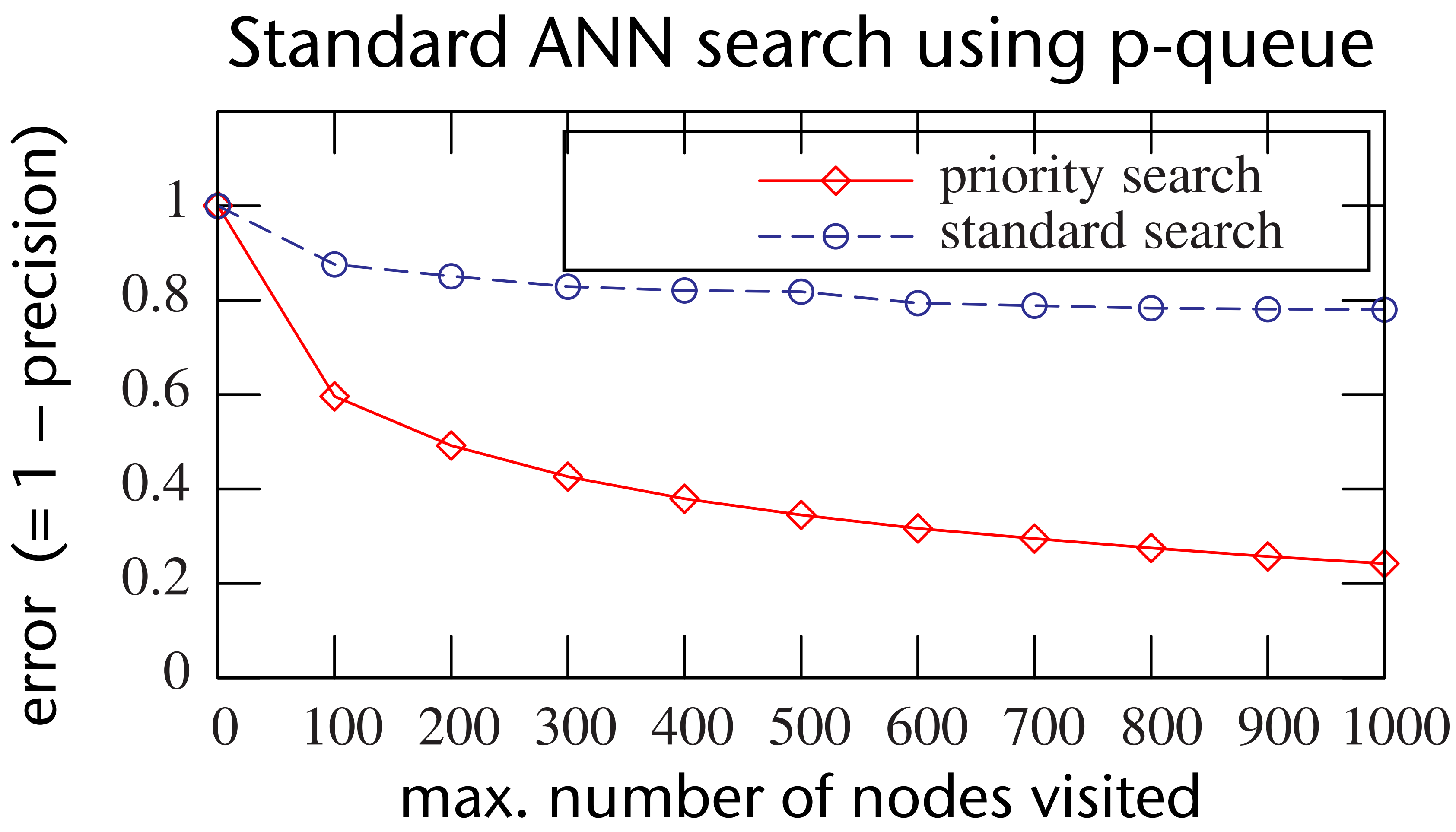


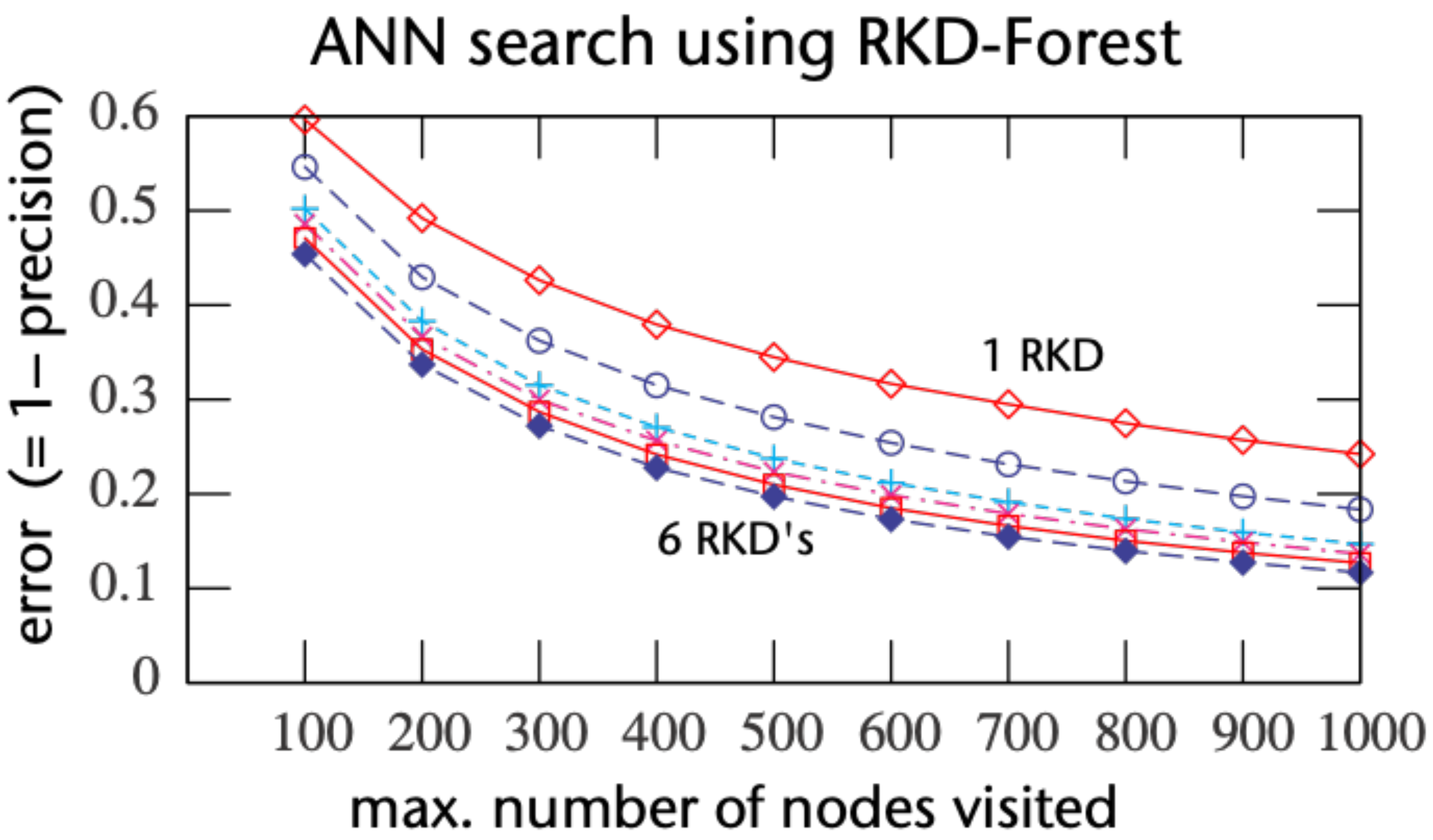




Therefore the algorithm for the ANN-search is better (asymptotically) than brute-force searching for all points and calculating their distance from the query point q.

Experimental Results on kd-Tree / RKD Forest [Silpa-Anan 2008]

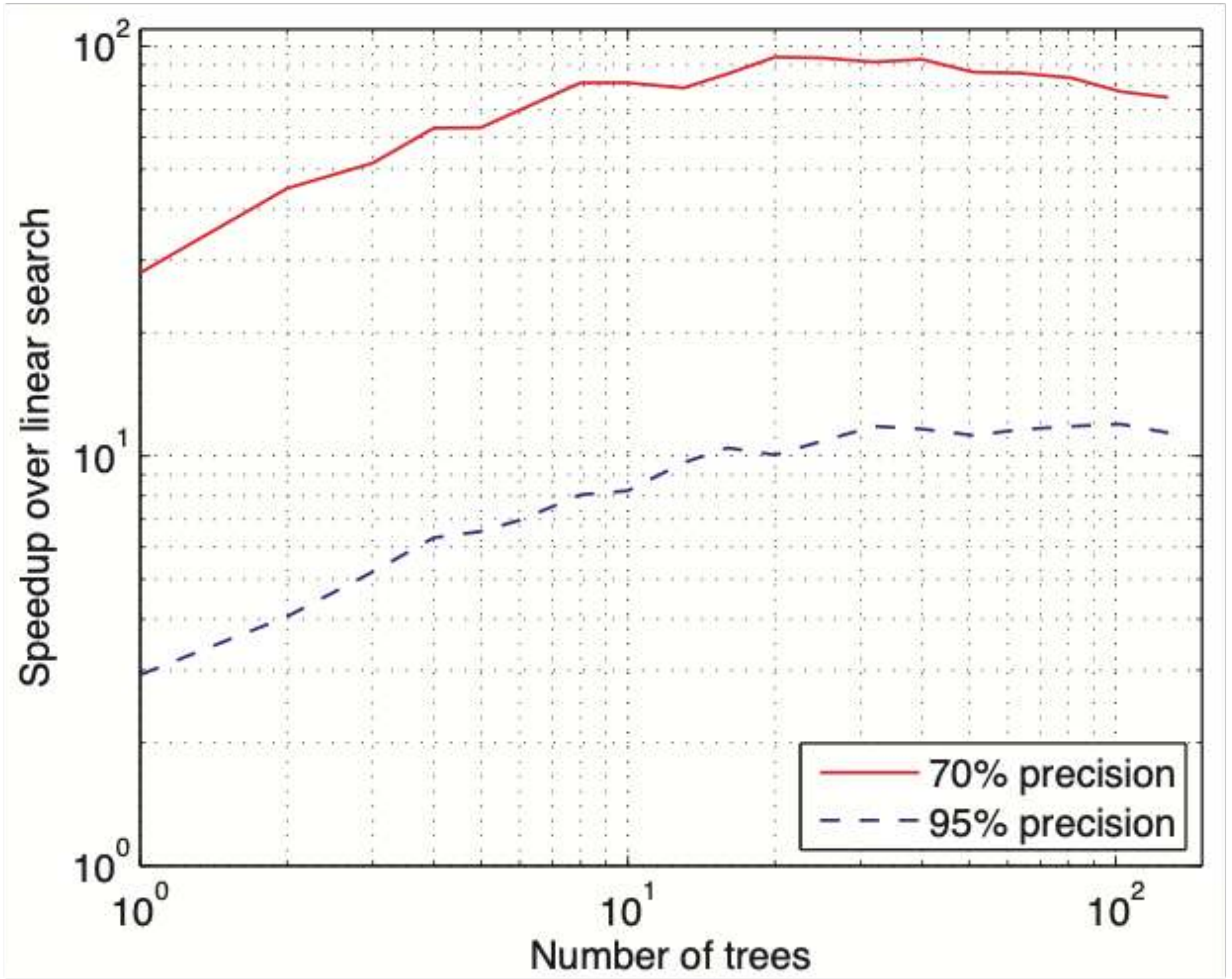


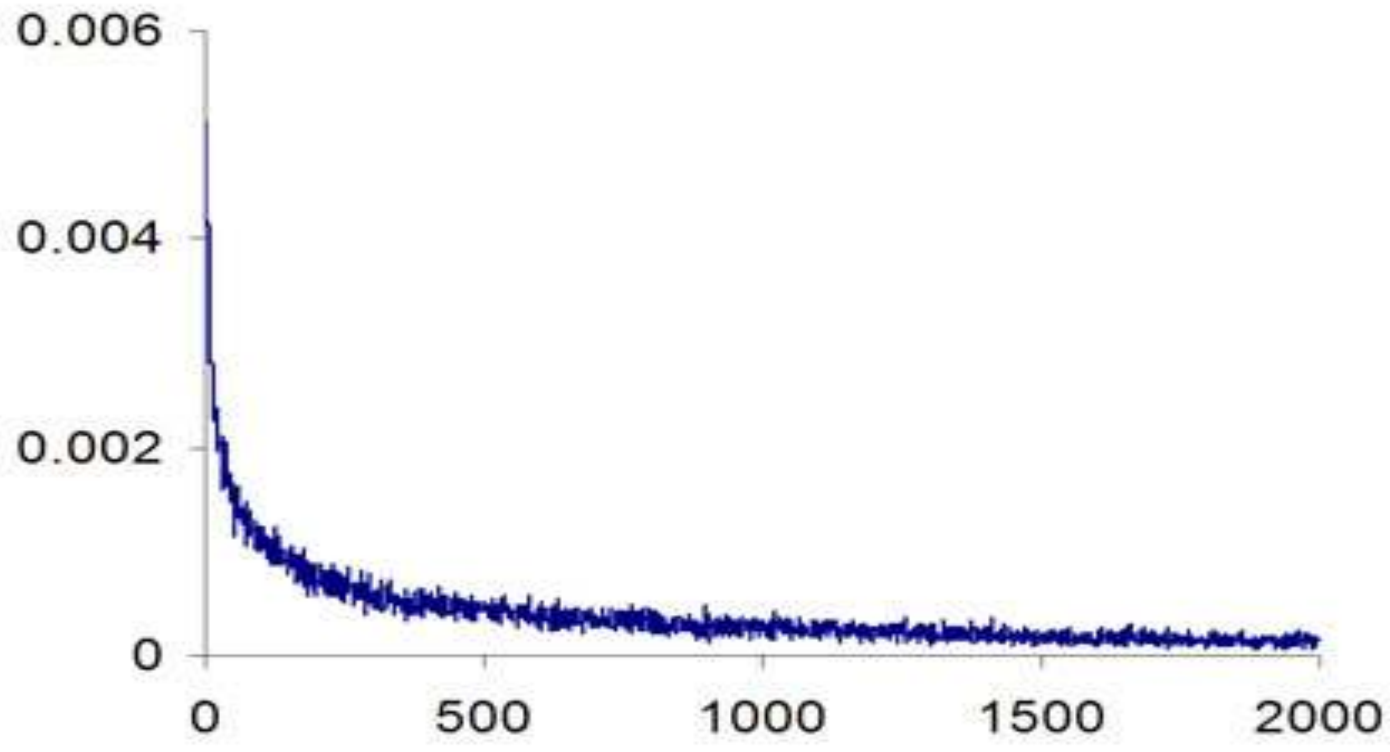


Set of 500,000 points
in 128-dimensional space

Distribution of rank of NN after projection to low dimensions

Optimal Number of Trees in RKD-Forest



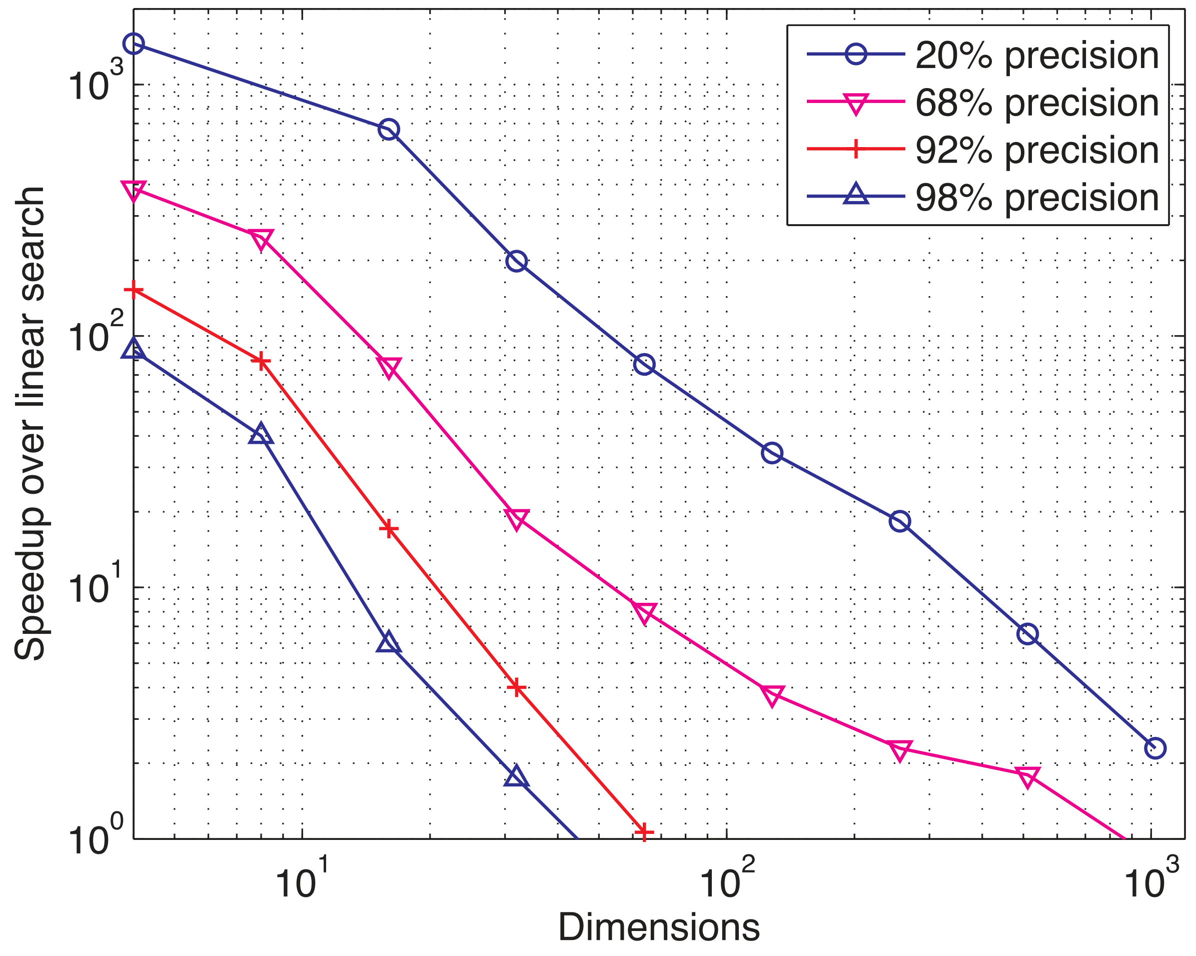


100,000 random points in a 128-dim. cube are projected by a projection π into 20 dim. space. The NN p* to a query q in 128D may become n-th NN after projection onto 20D. The x-axis is the ranking n and the y-axis shows the probability that the projection π(p*) will be n-th NN to π(q) in 20D. The graph shows a long tail.

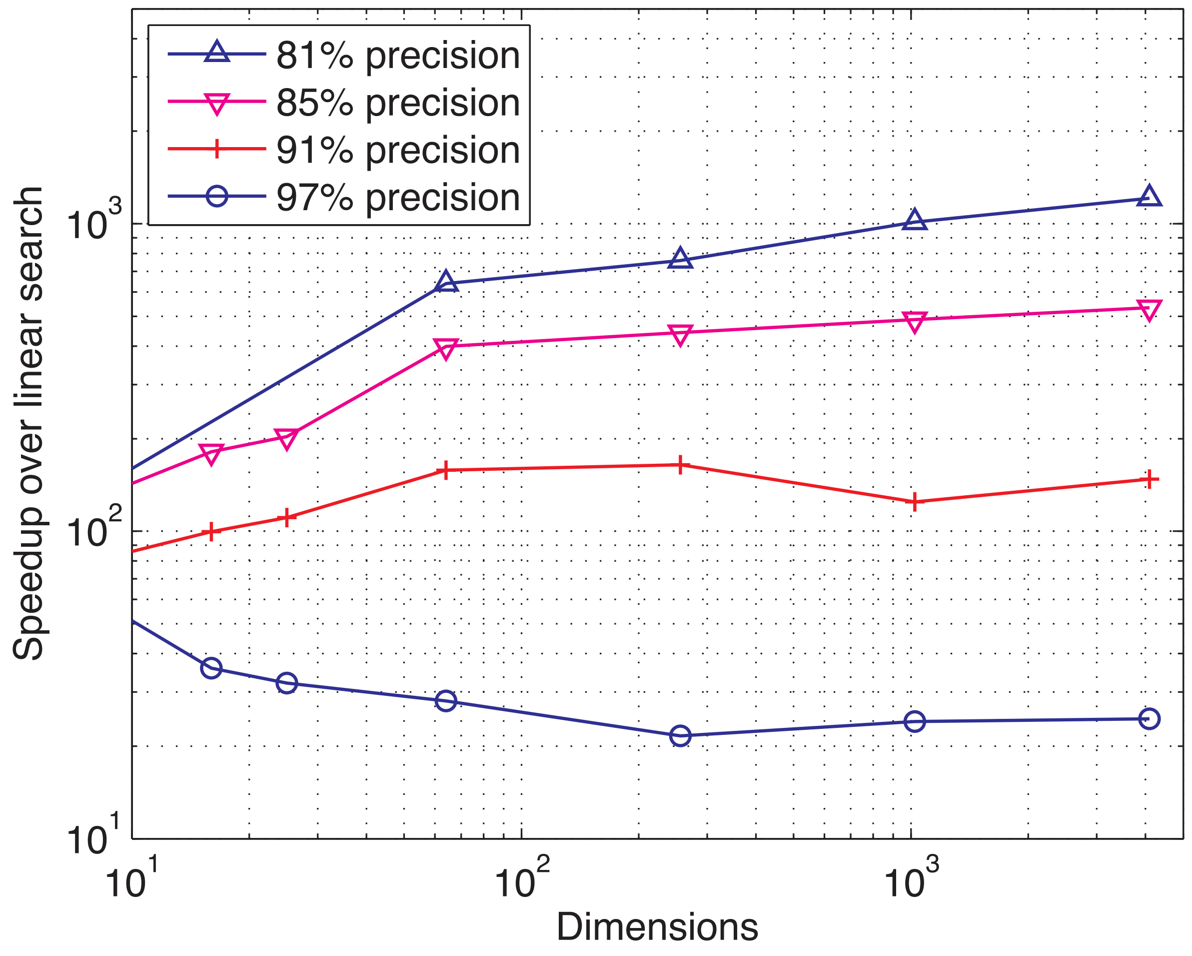
⟶ Another reason why the RKD-forest works better.

Performance depends highly on distribution of input point set

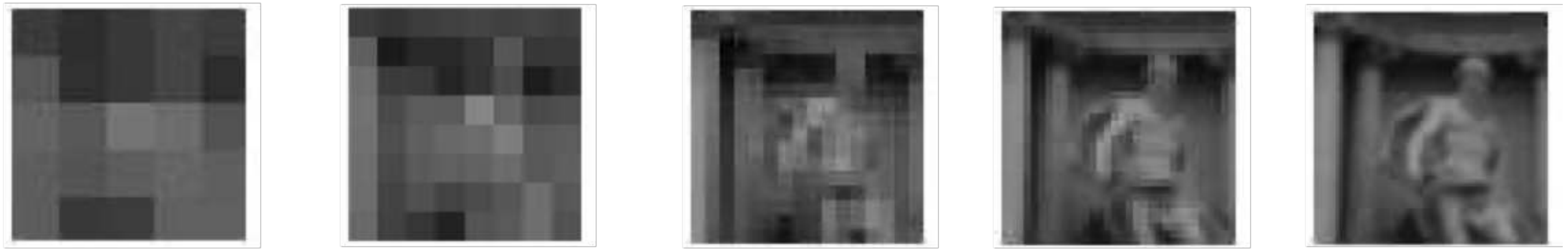
Search efficiency for data of varying dimensionality



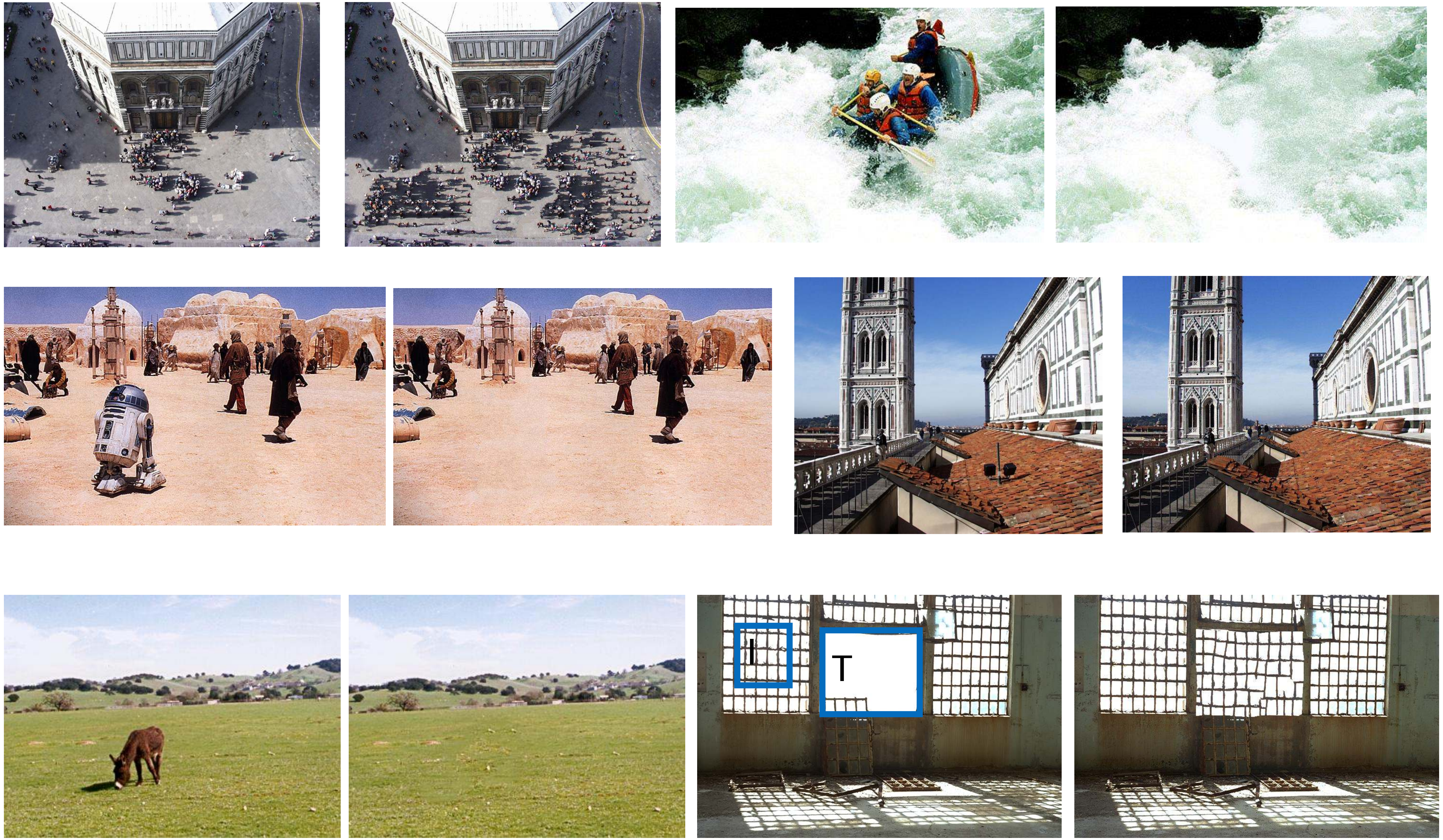
(a) Random vectors



(b) Image patches



Texture Synthesis

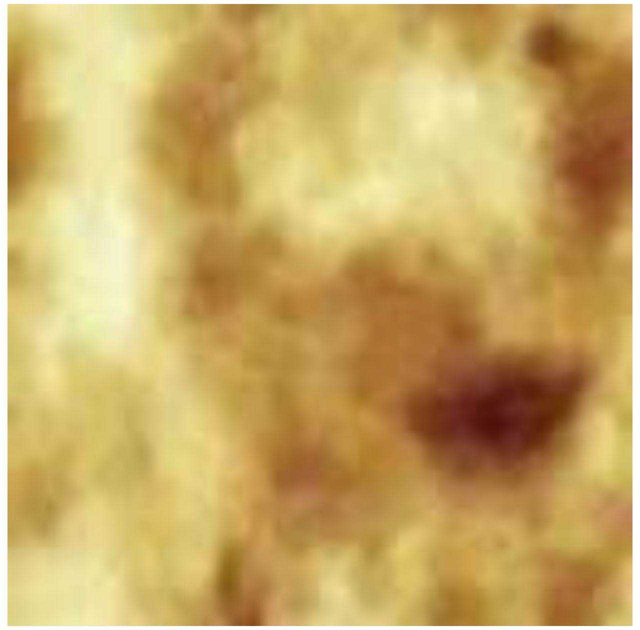


Wei & Levoy

Wei & Levoy

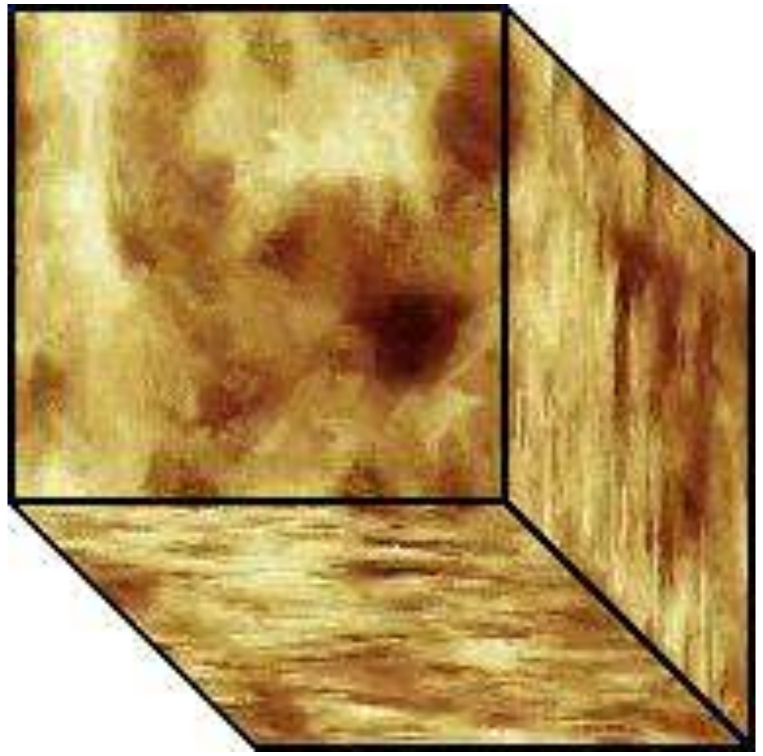

Aspen Trees
Harris and Love, Inc.


Aspen Trees
Harris and Love, Inc.



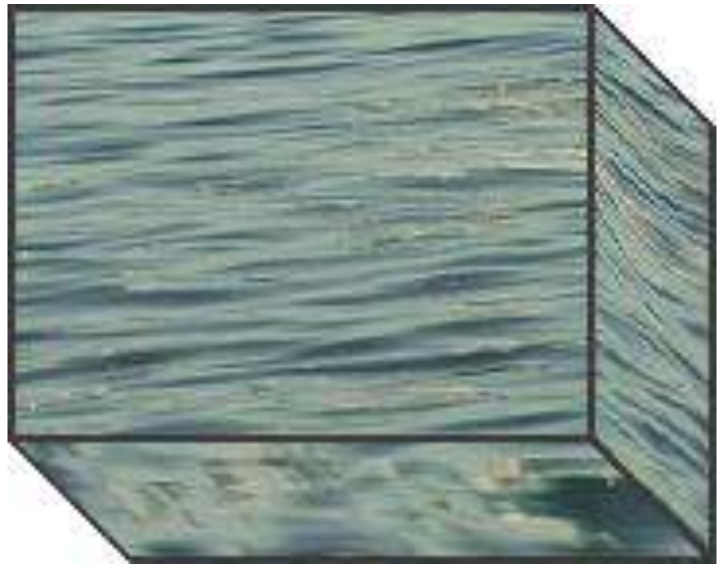
original

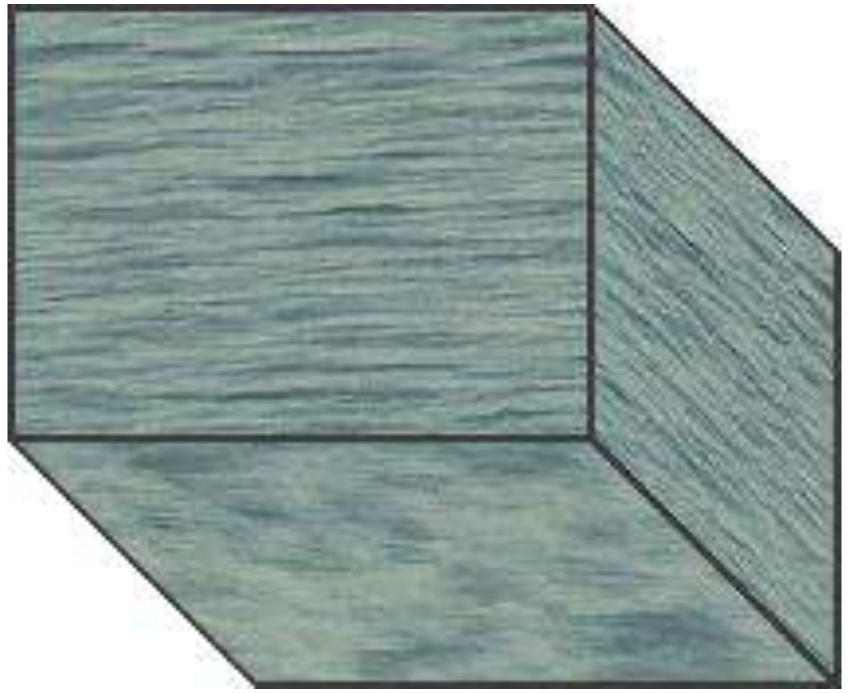
synthesized



original

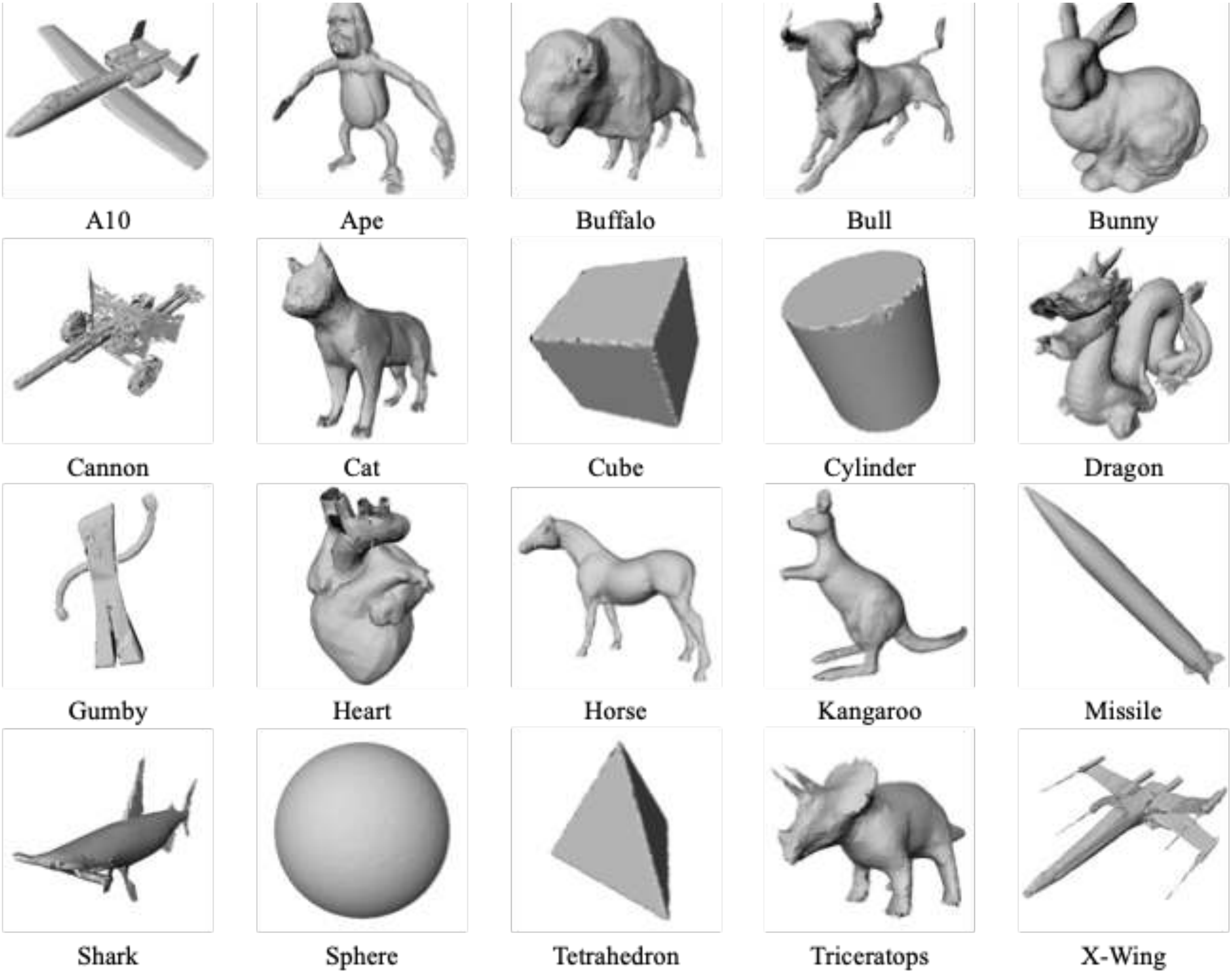
synthesized

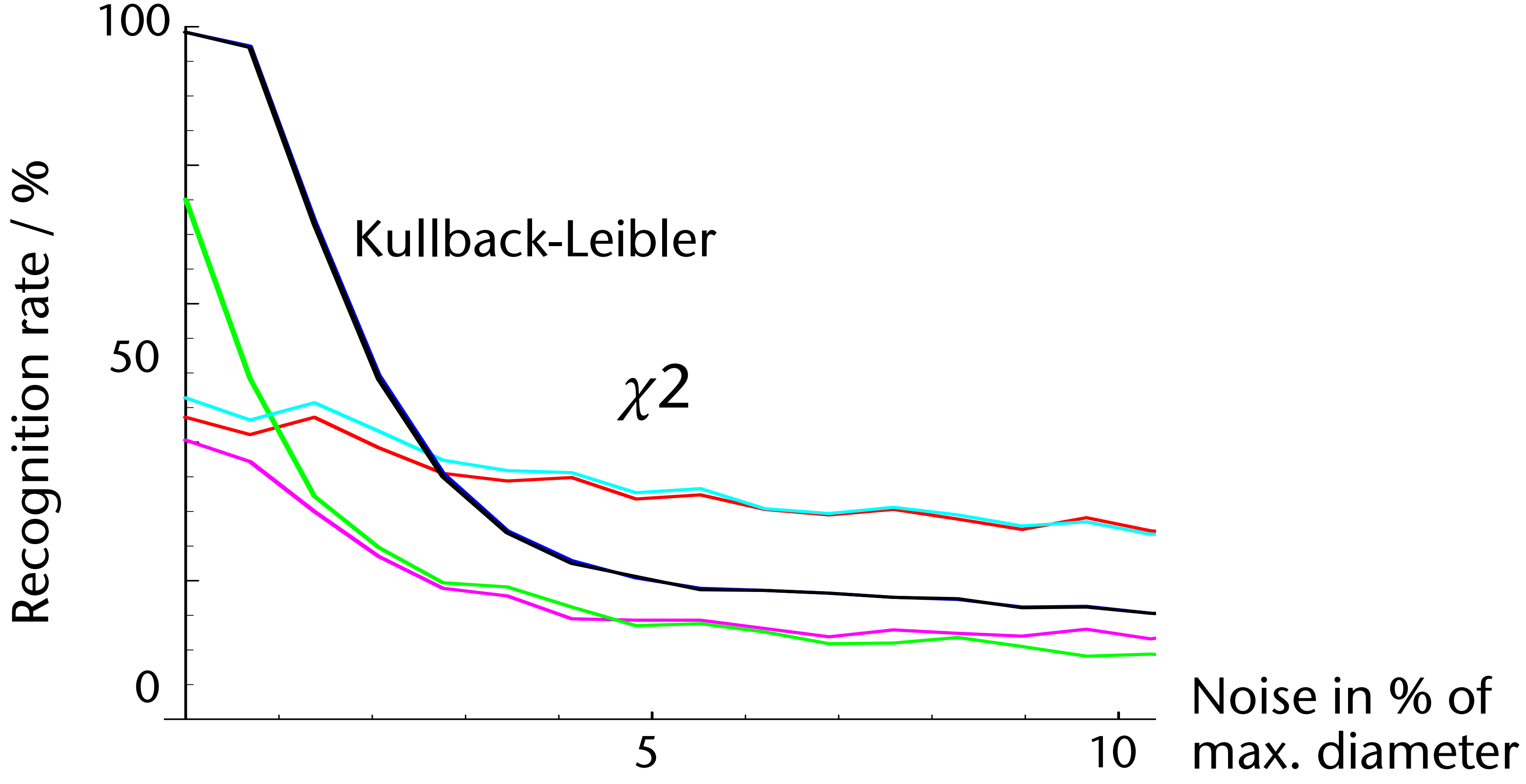


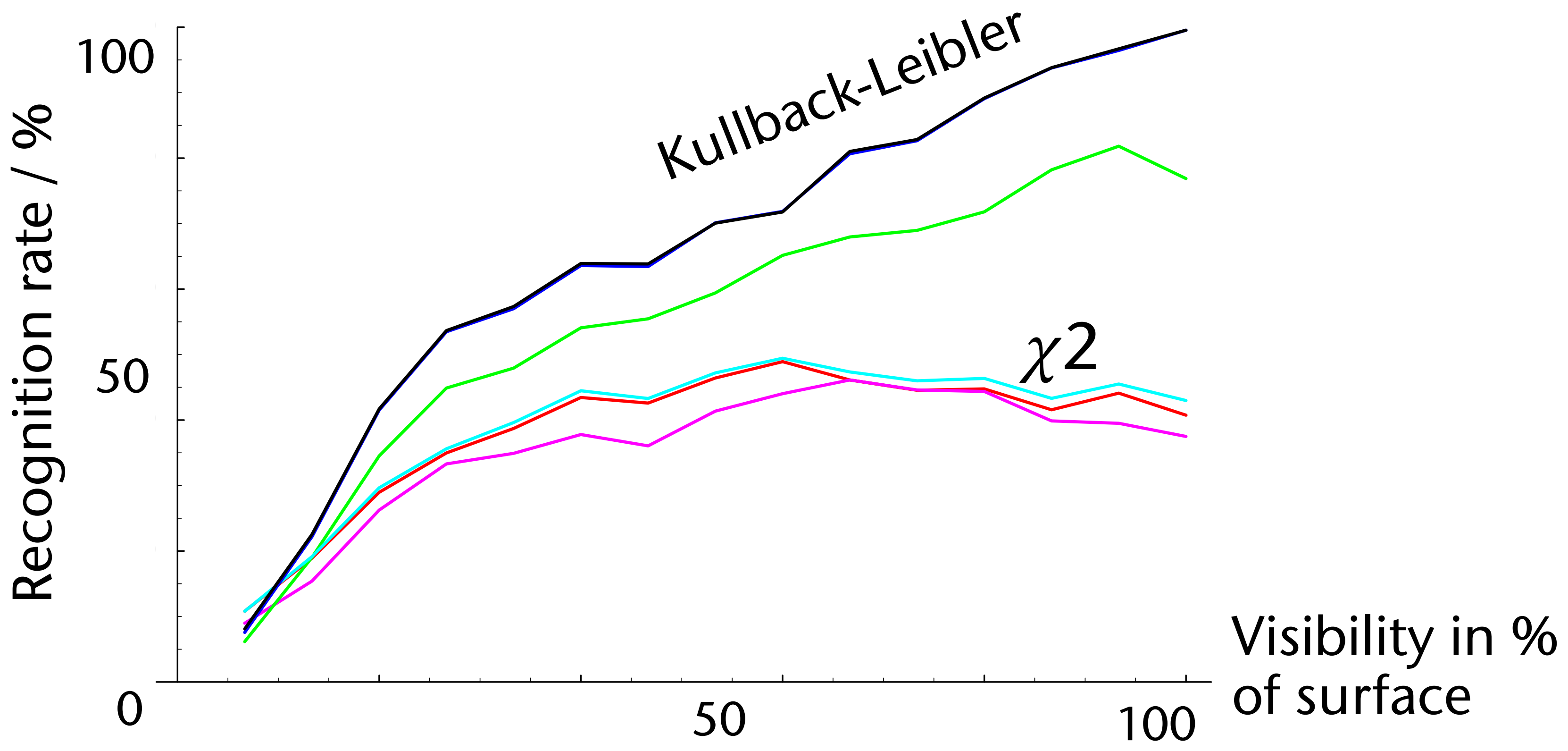


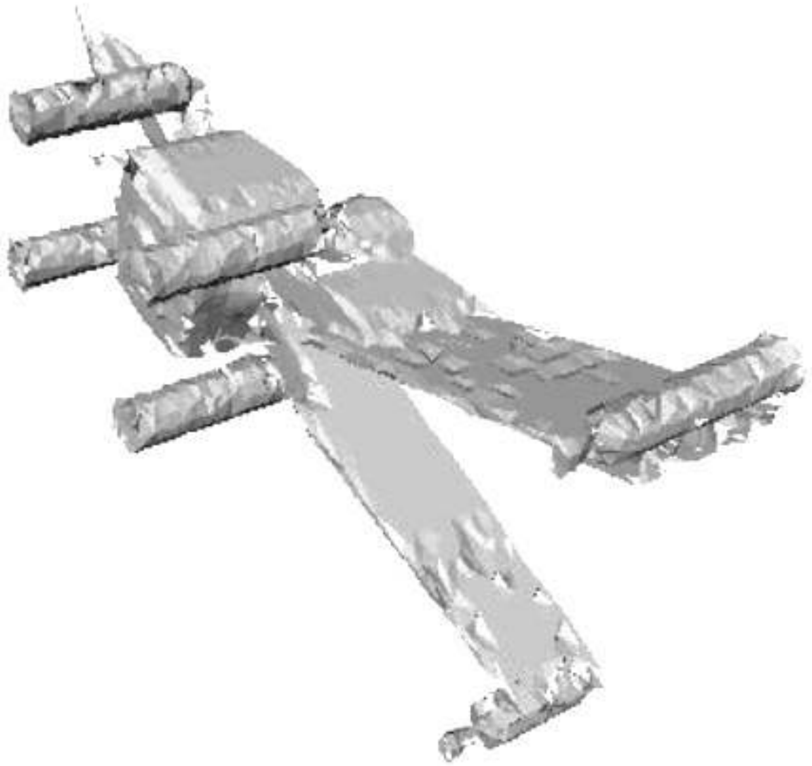
Experiments and Results Regarding Surflet-Pair Histograms

Test Objects




100
50
0
Recognition rate / %
Kullback-Leibler
$\chi 2$
5
10
Noise in % of
max. diameter

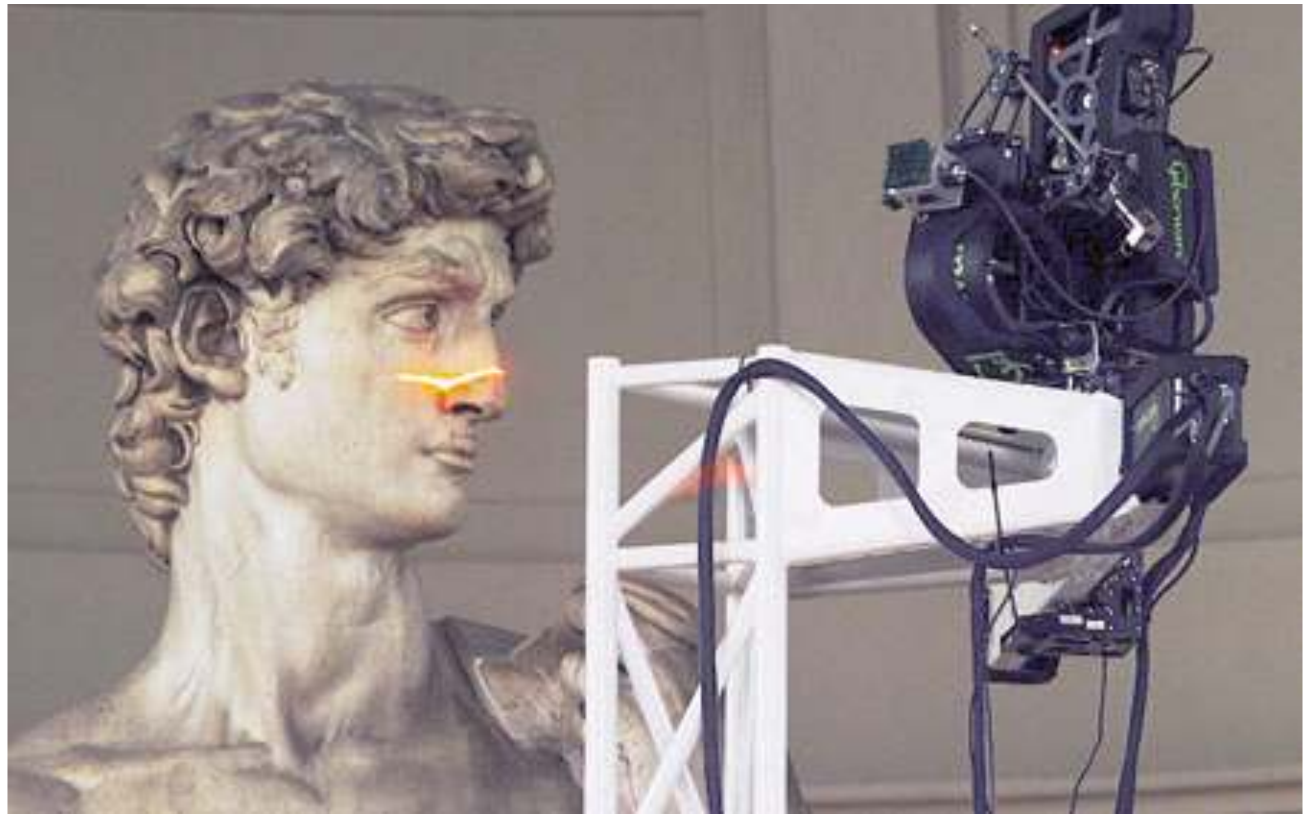

100
50
0
Recognition rate / %
Kullback-Leibler
$\chi 2$
50
100
Visibility in %
of surface

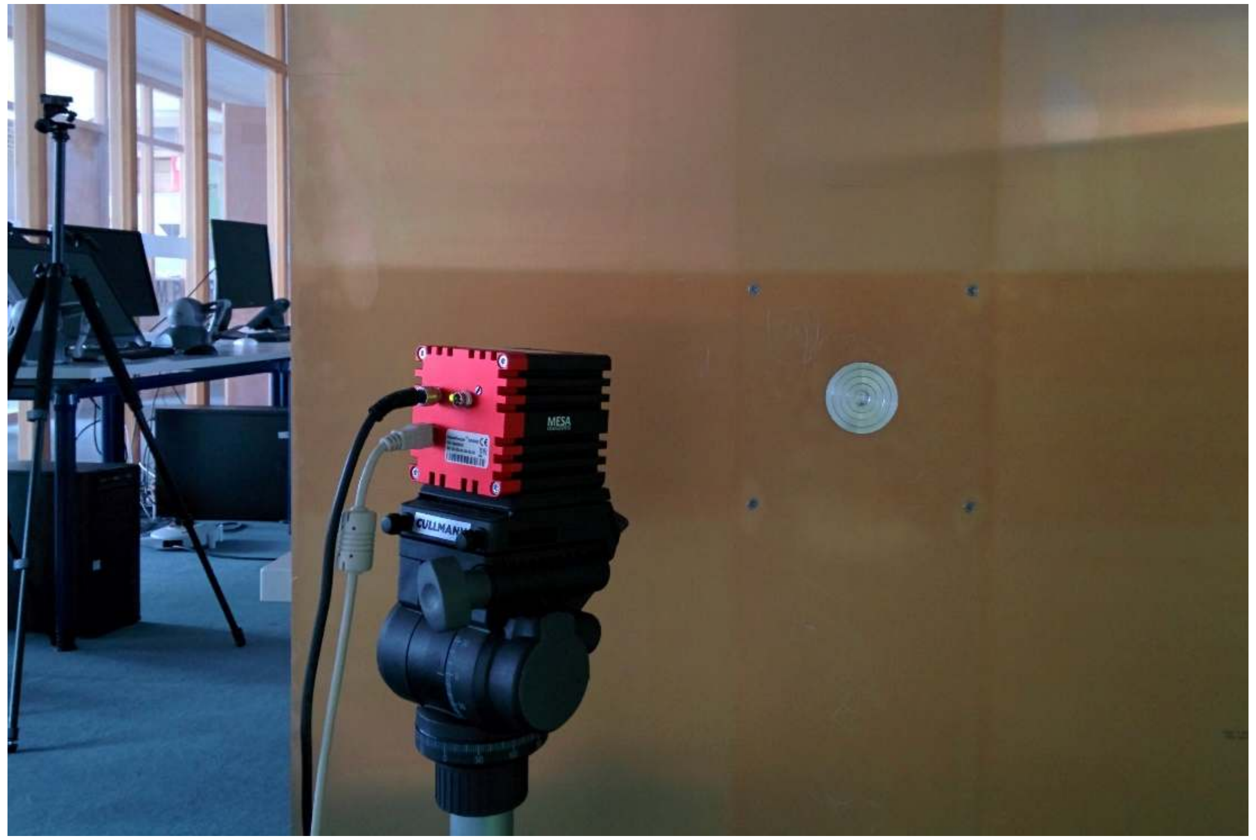


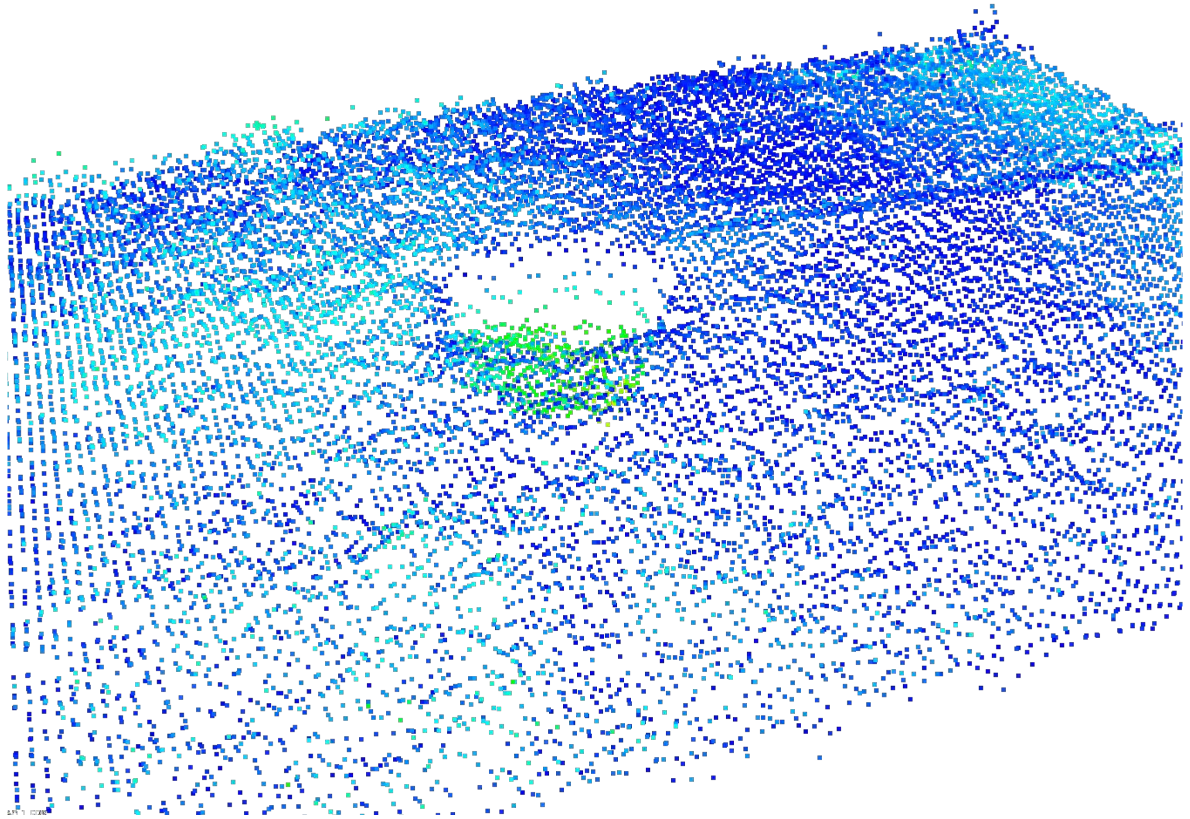
Point Cloud Surfaces

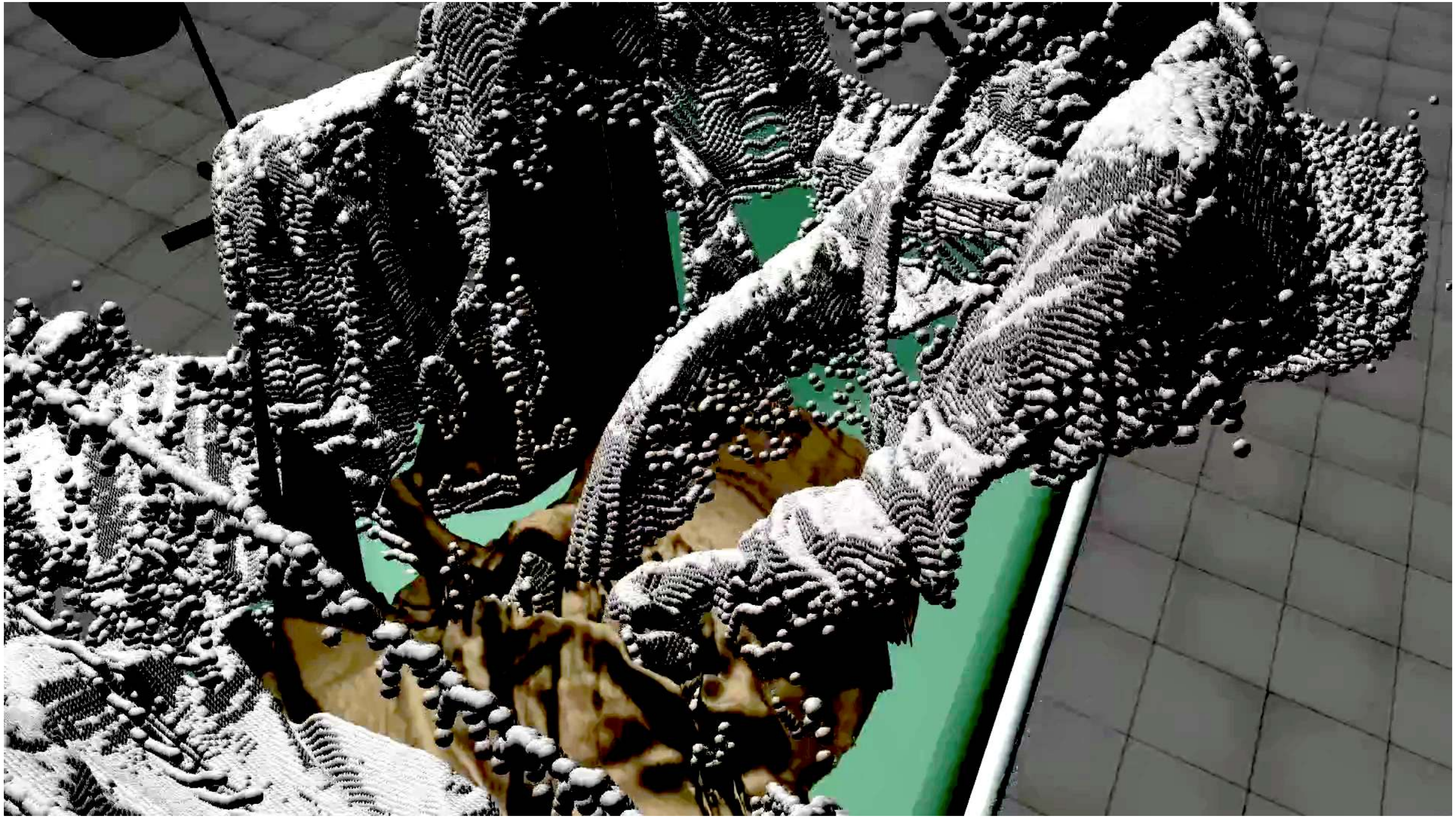
- Increasingly popular geometry representation
- Lots of sources of point clouds (laser scanners, Kinect et al., ...)

Applications



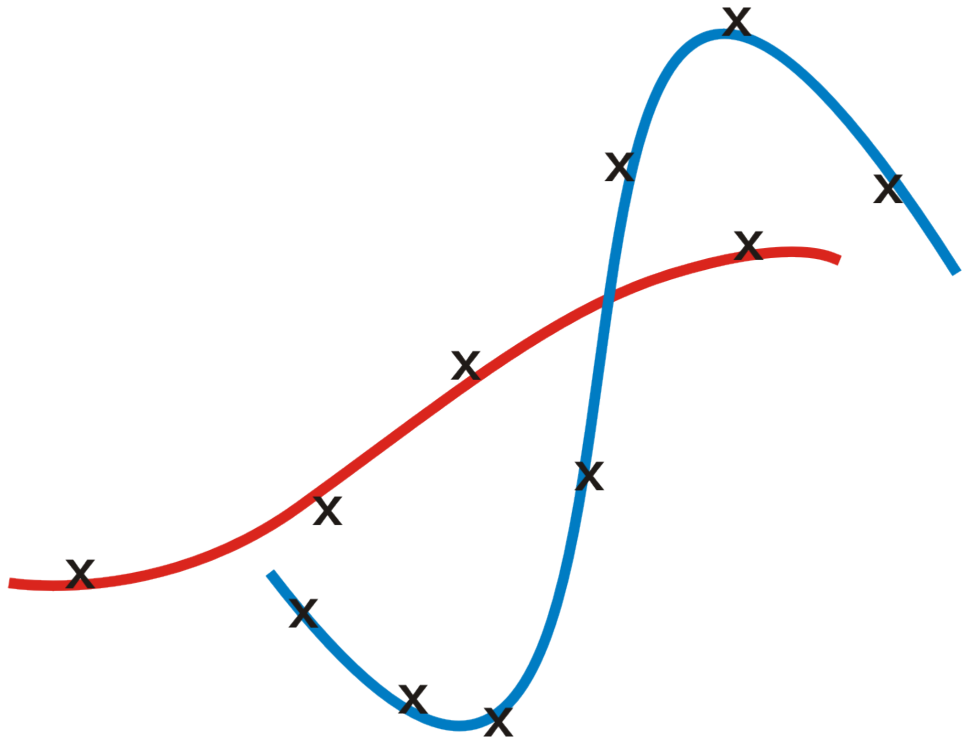






Goal

- Surface definition that is ..
  - Quick to evaluate
  - Robust against noise
  - Smooth
- The surface definition / representation should be well suited for:
  - Ray tracing (rendering)
  - Collision detection (physics)

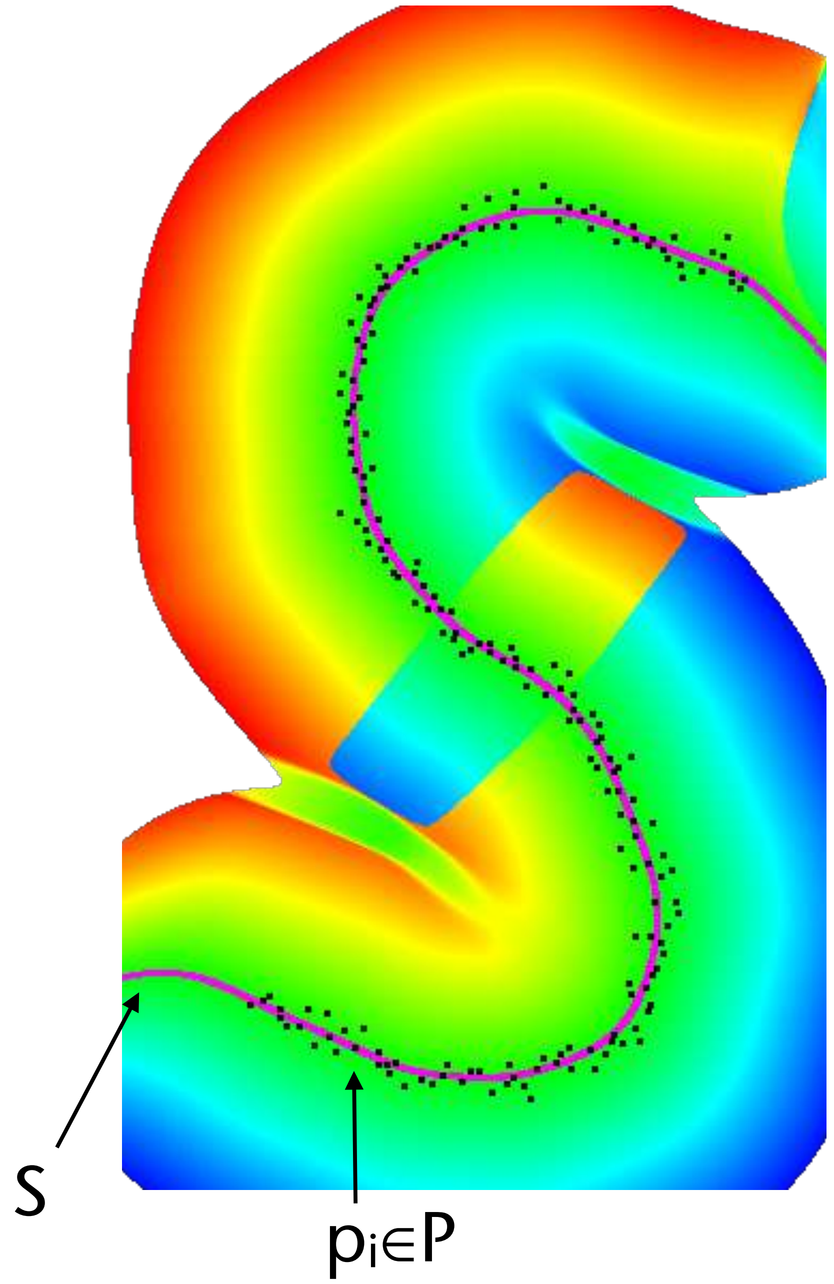


Weighted Moving Least Squares: an Implicit Surface Definition

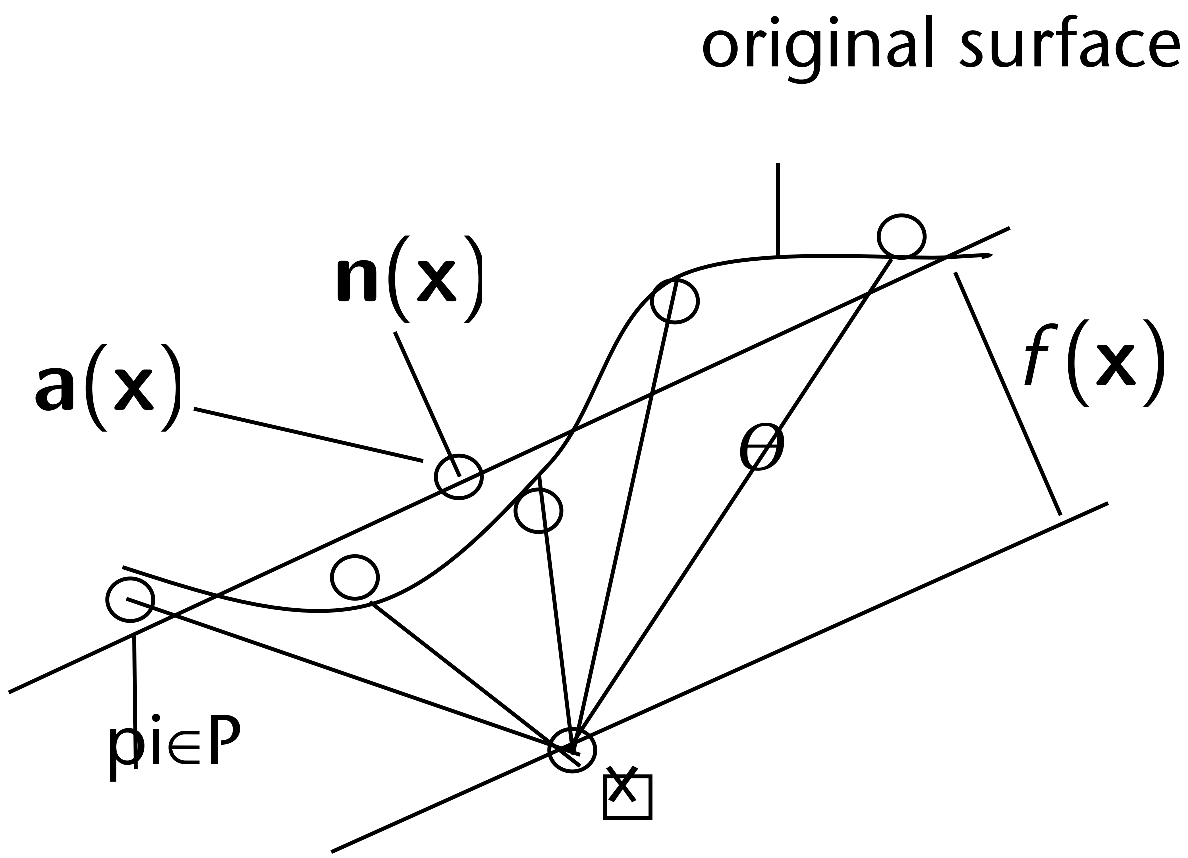
- Consider a point cloud P as noisy sampling of a smooth surface
    - Consequence: reconstructed surface should *not interpolate* the points
- Define the surface as an implicit surface over a smooth distance function f, determined by the point cloud P:

$$S = \{\mathbf{x} \mid f(\mathbf{x}; P) = 0\}$$

where f is the distance to the yet unknown surface S



- Define f using weighted moving least squares over k nearest neighbors



- The surface is approximated locally by a plane through

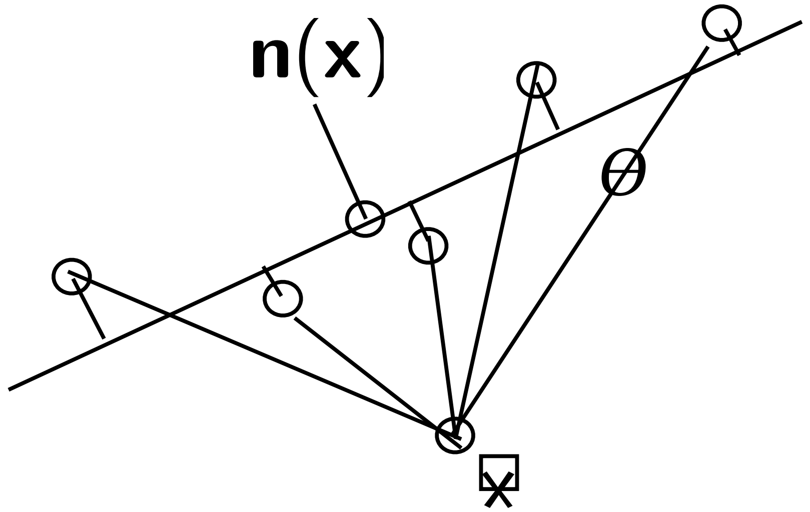
$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{i=1}^{k} \theta(\|\mathbf{x} - \mathbf{p}_i\|)\mathbf{p}_i}{\sum_{i=1}^{N} \theta(\|\mathbf{x} - \mathbf{p}_i\|)}$$

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

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

- Choose n as

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

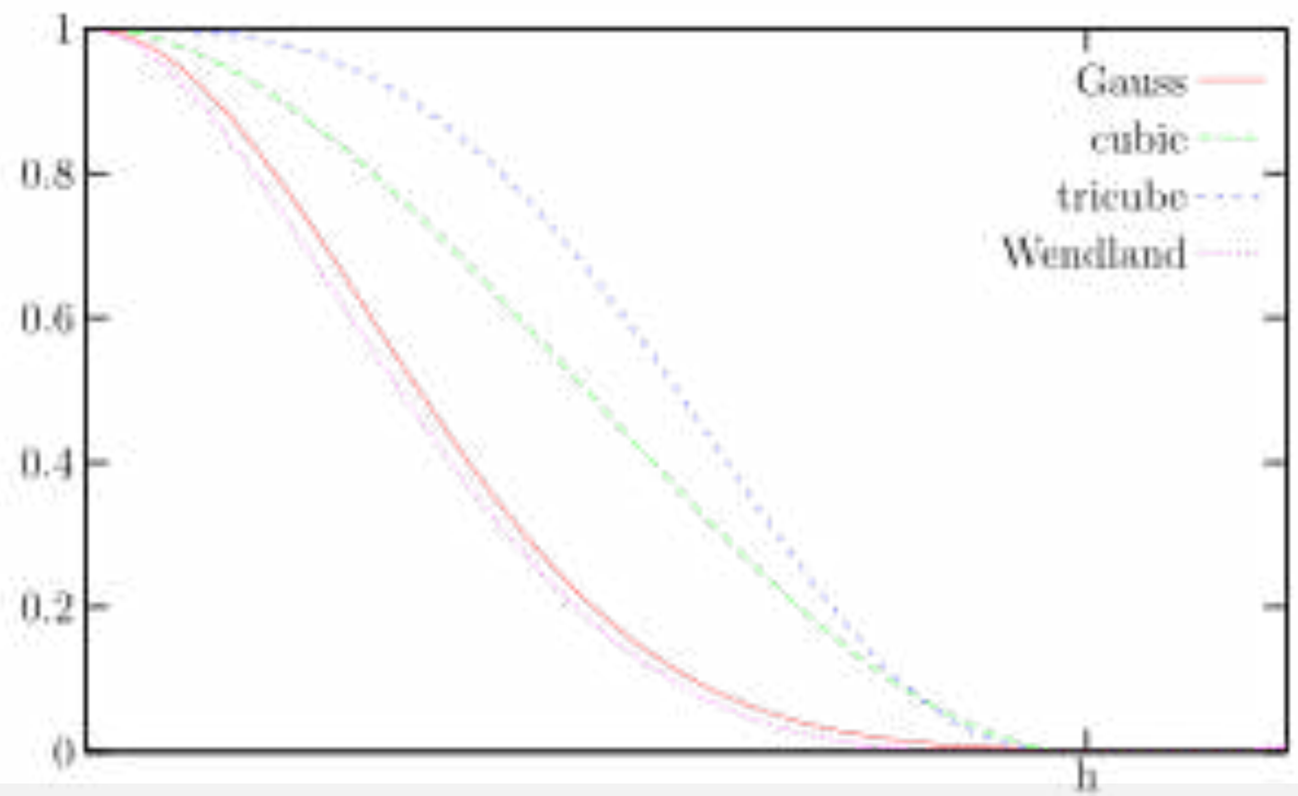


- From PCA we know: n happens to be the smallest eigenvector of the weighted covariance matrix B = (bij) ∈ ℝ3x3 with

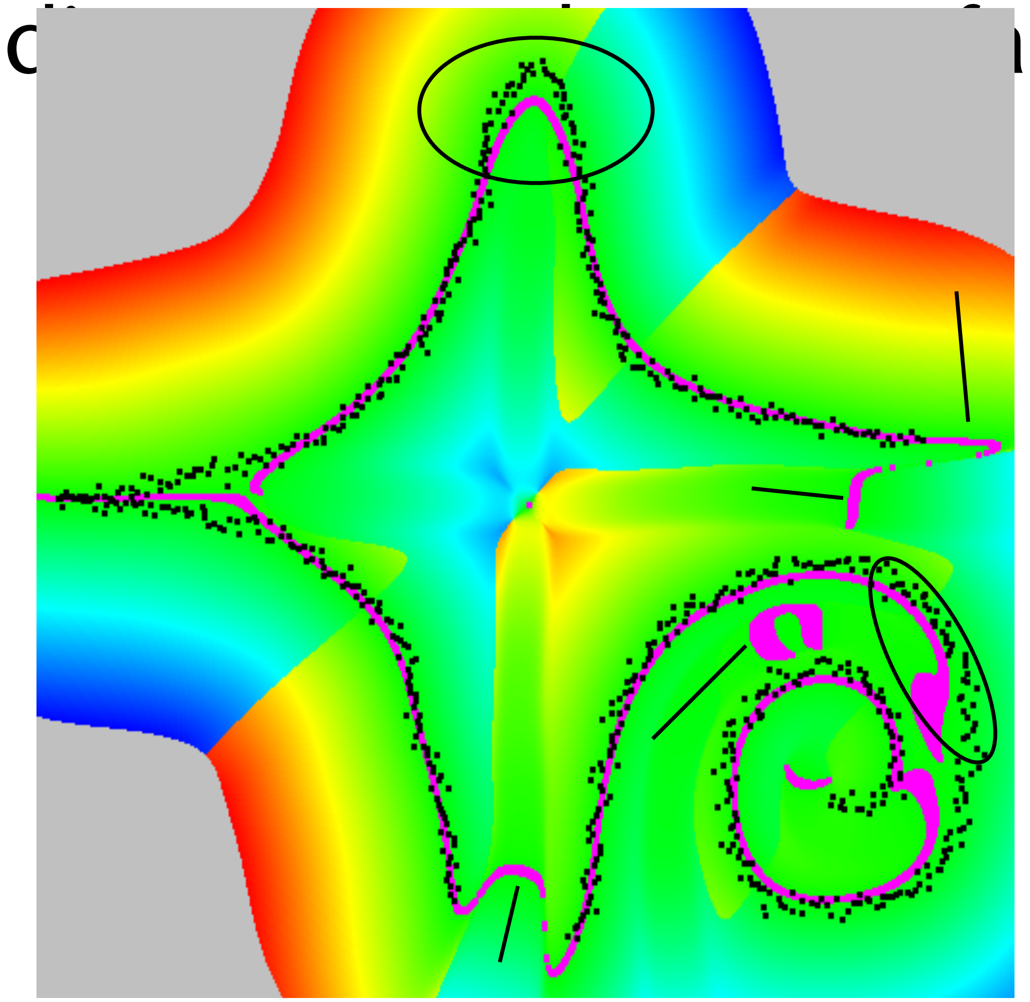
$$b_{ij} = \sum_{k=1}^{K} \theta(\|\mathbf{x} - \mathbf{p}_k\|)(p_{k,i} - a_i)(p_{k,j} - a_j)$$

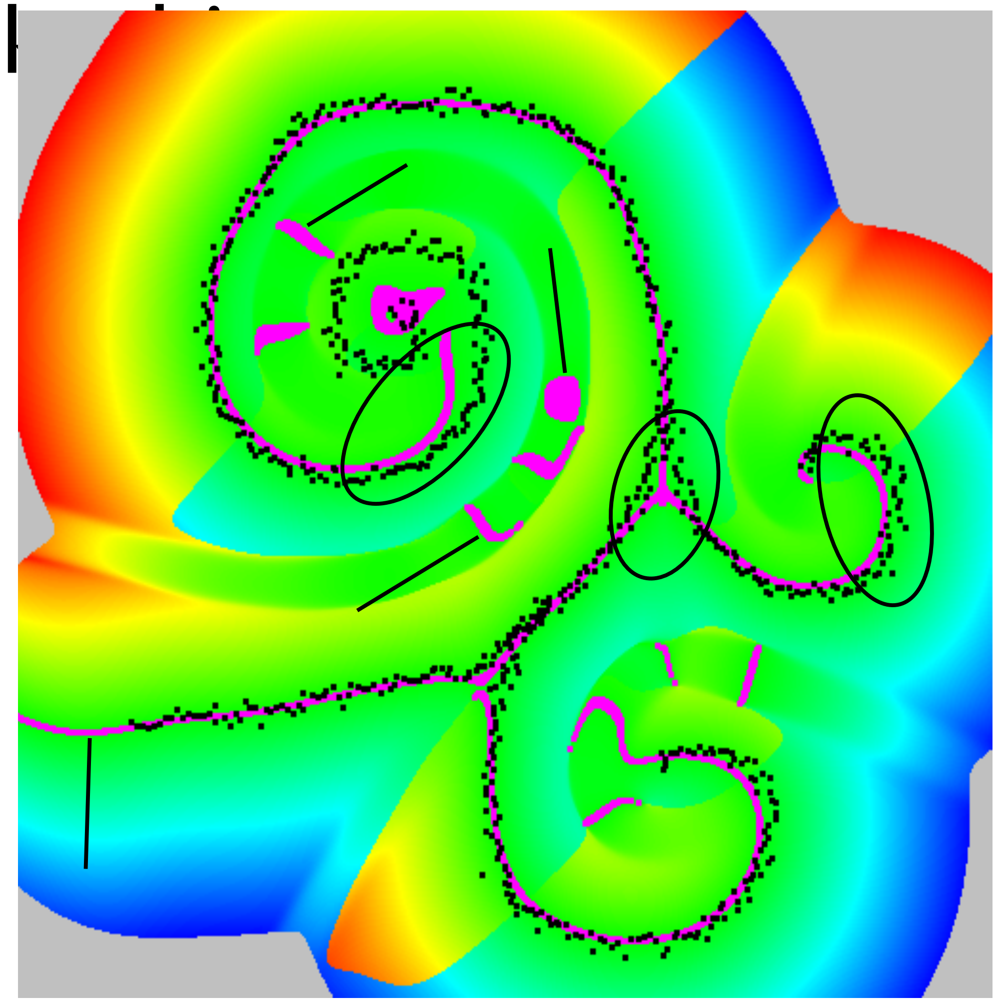
- For the weight function Θ, use (e.g.) a Gaussian kernel $\theta(d) = e^{-d^2/h^2}$, $d = \|\mathbf{x} - \mathbf{p}_i\|$

- 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 $\theta(d) = \left(1 - \frac{d}{h}\right)^4 \left(4\frac{d}{h} + 1\right)$

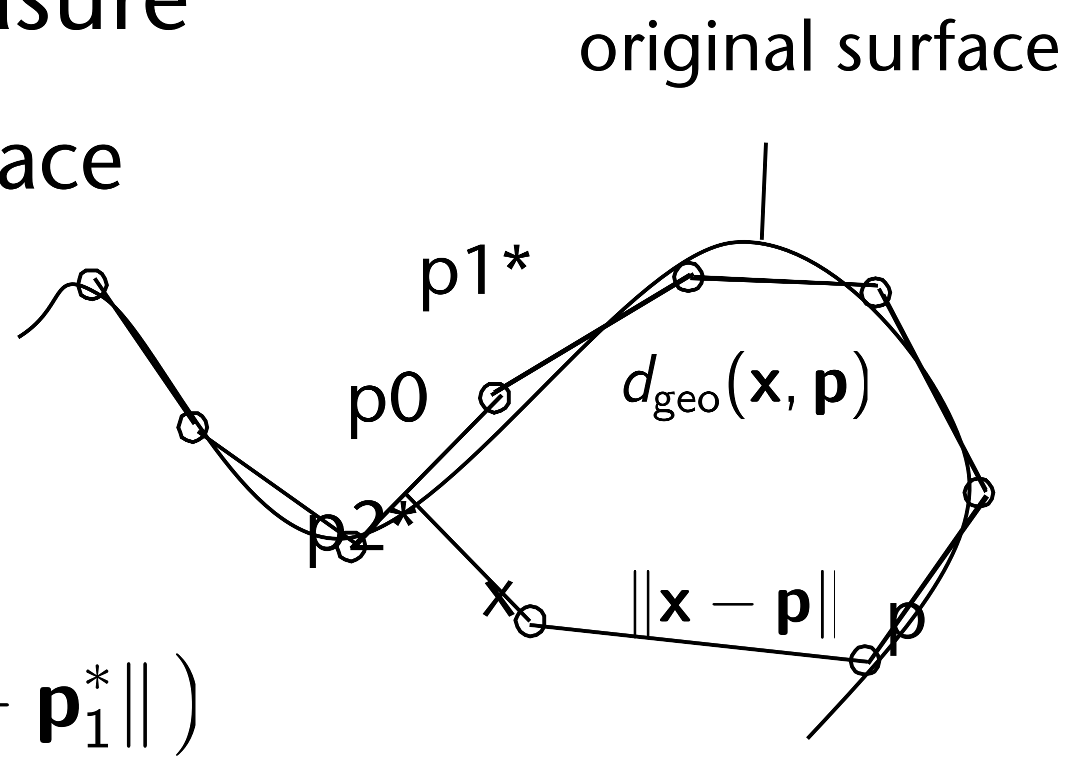


- Whatever kernel you use, it is fine to consider only "close neighbors" around x for the computation of a(x) and n(x)
  $\longrightarrow$ need lots of k-NN searches in P
- More important: what distance measure to use in $\theta(\|\mathbf{x} - \mathbf{p}_i\|)$?
- Euclidean d … acts lik…





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



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

$$a = \|\mathbf{p}^0 - \mathbf{p}_1^*\|$$

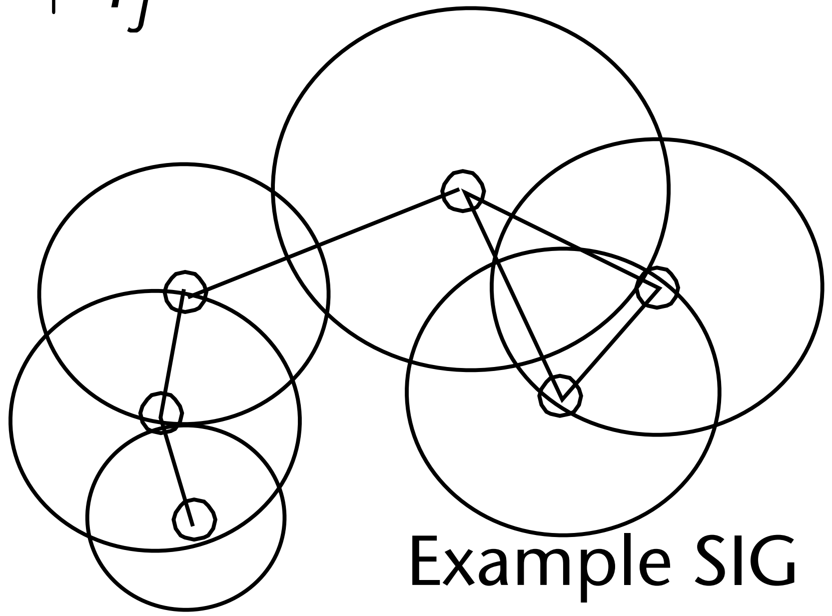
with $d(\mathbf{p}_i^*, \mathbf{p})$

and = length of shortest path through proximity graph

$$\|\mathbf{p}^0 - \mathbf{x}\|$$

Which Proximity Graph to Use

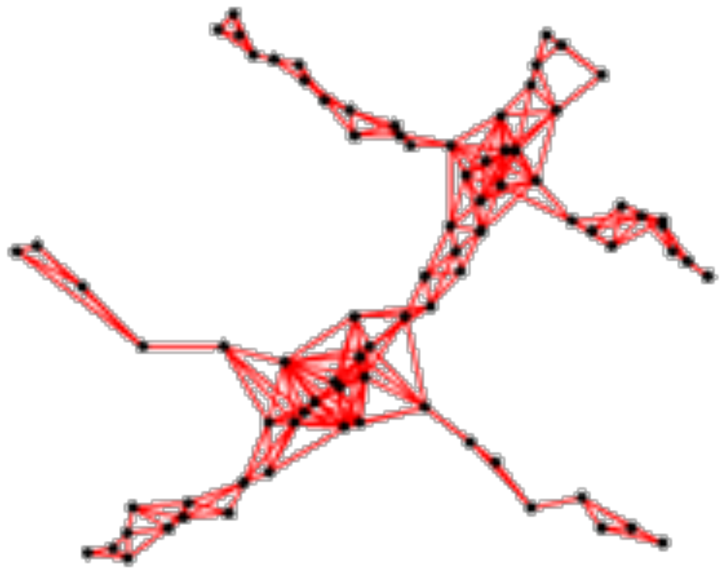
- Many kinds of proximity graphs
  - Delaunay graph (to be explained later)
    - Needs kind of a "pruning" because of "long" edges; still has problems
  - Most other proximity graphs are subgraphs of the Delaunay graph
  - Sphere-of-Influence graph (SIG): is not a subgraph of the DG
- Definition of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



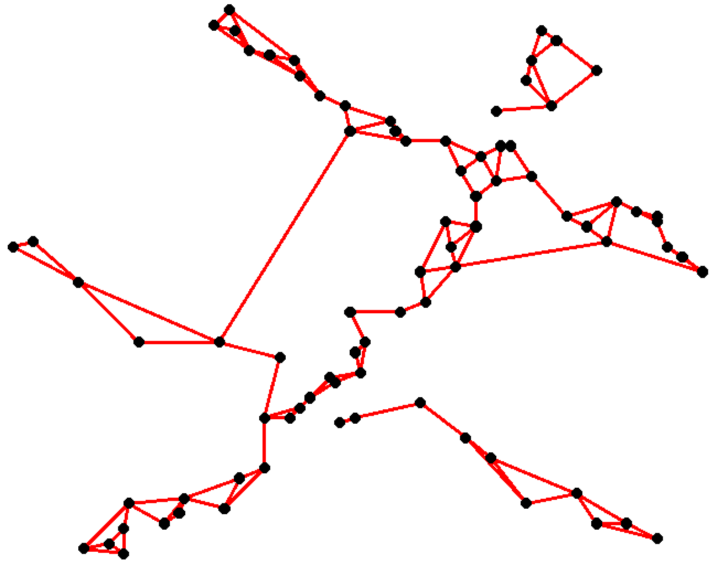
Example SIG

Results

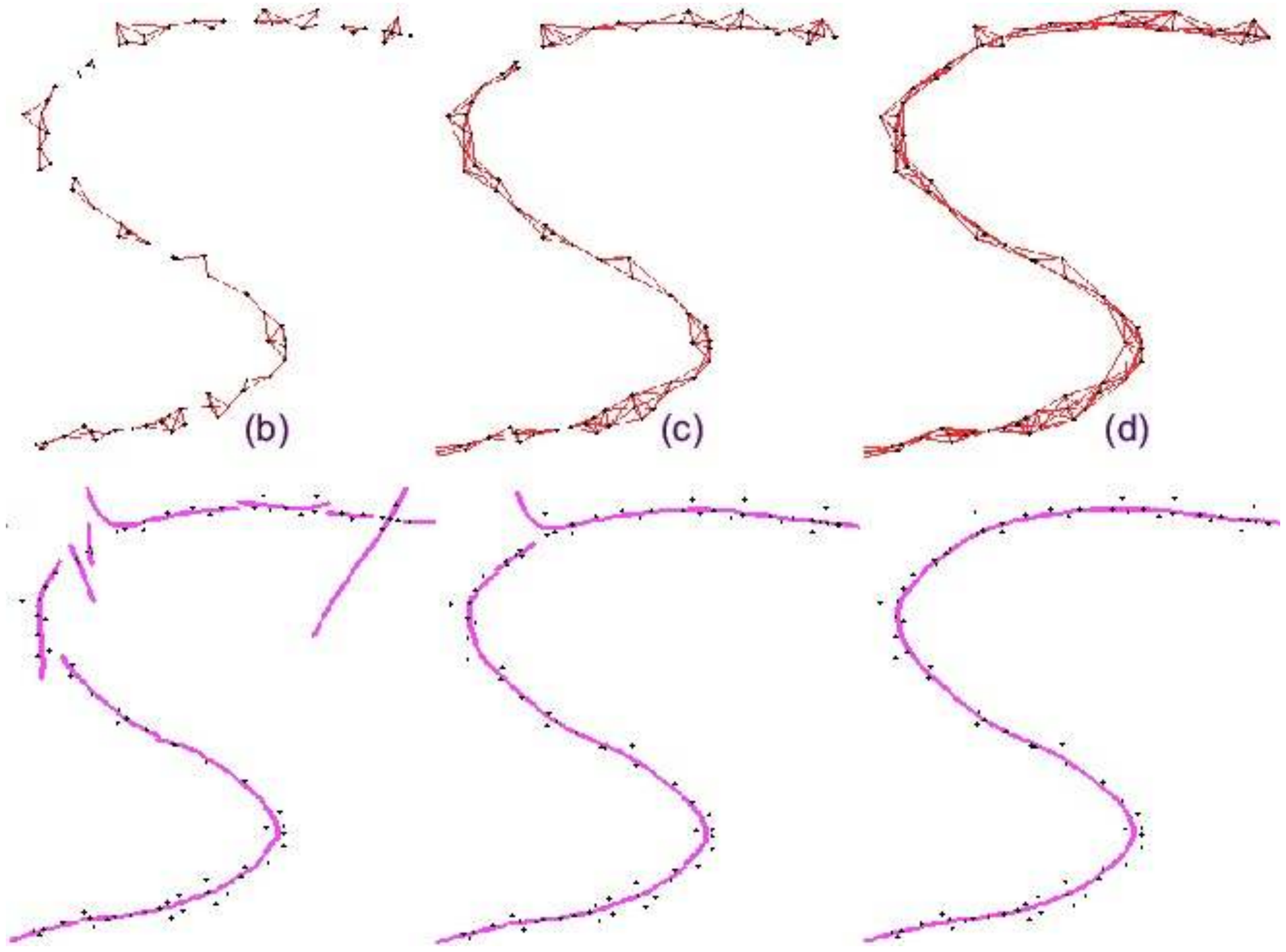
Example sphere-of-influence graph (k-SIG)



Delaunay graph with pruning



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

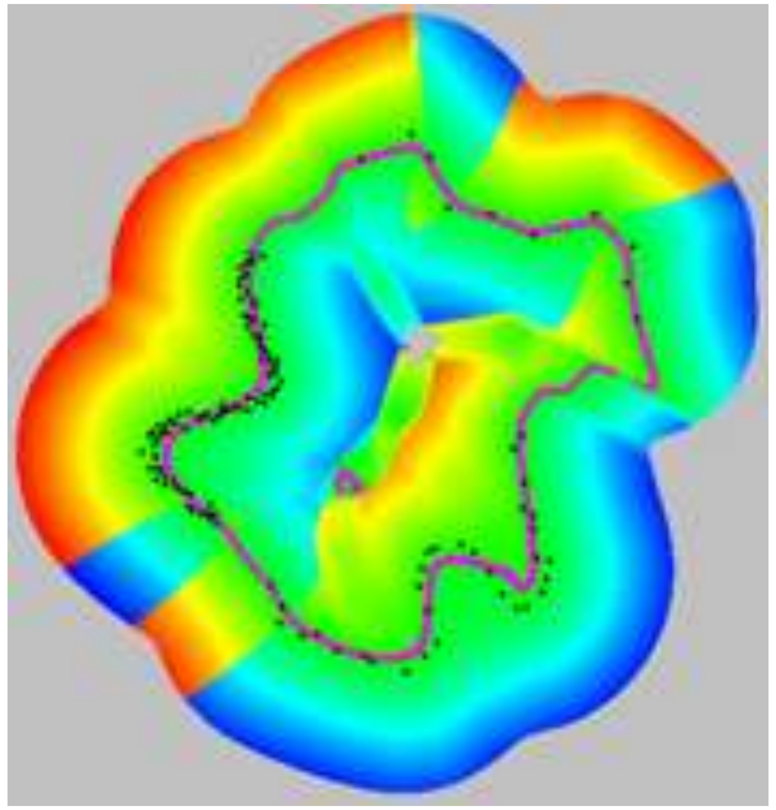


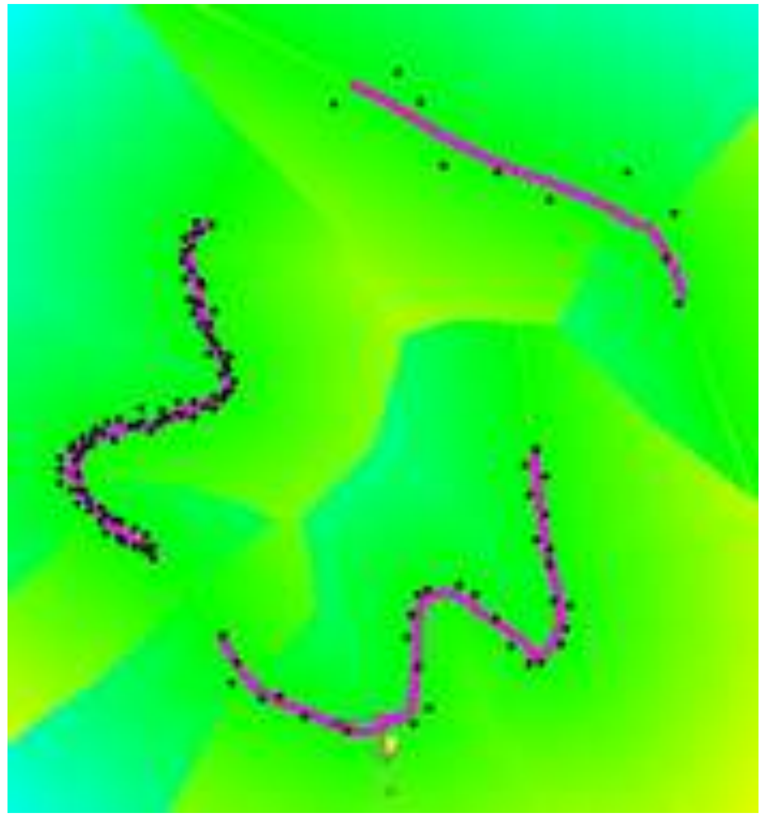
1-SIG 2-SIG 3-SIG

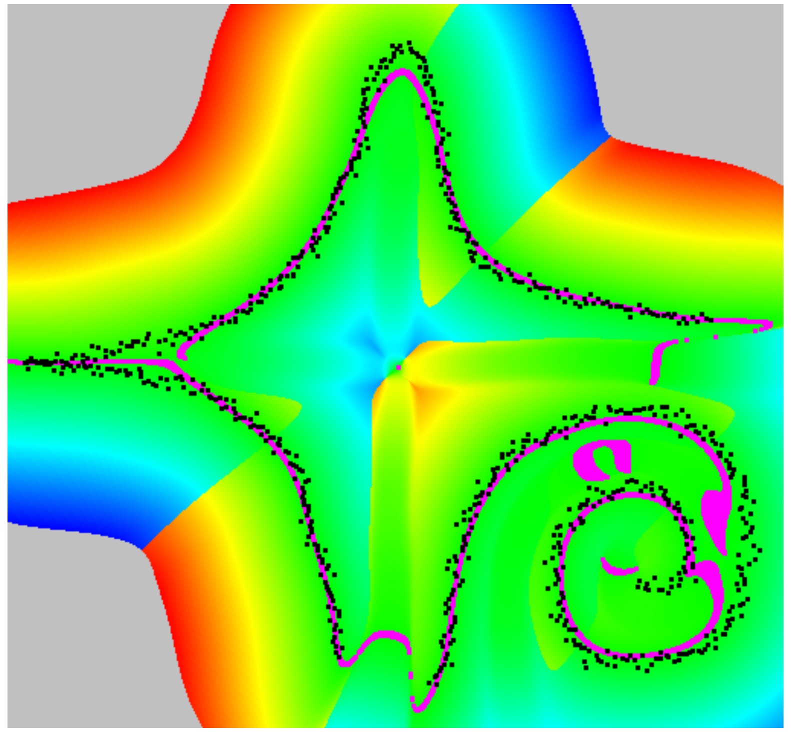
Weighted MLS surface
with Euclidean distance
and fixed bandwidth in kernel

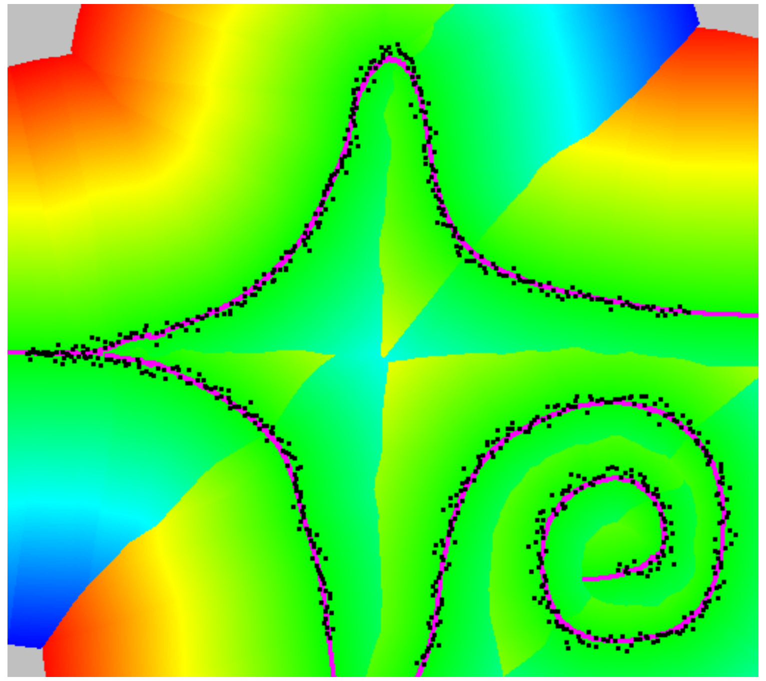
Weighted MLS surface
with proximity graph-based distance
and automatic bandwidth estimation in ke

→ Master's Thesis]



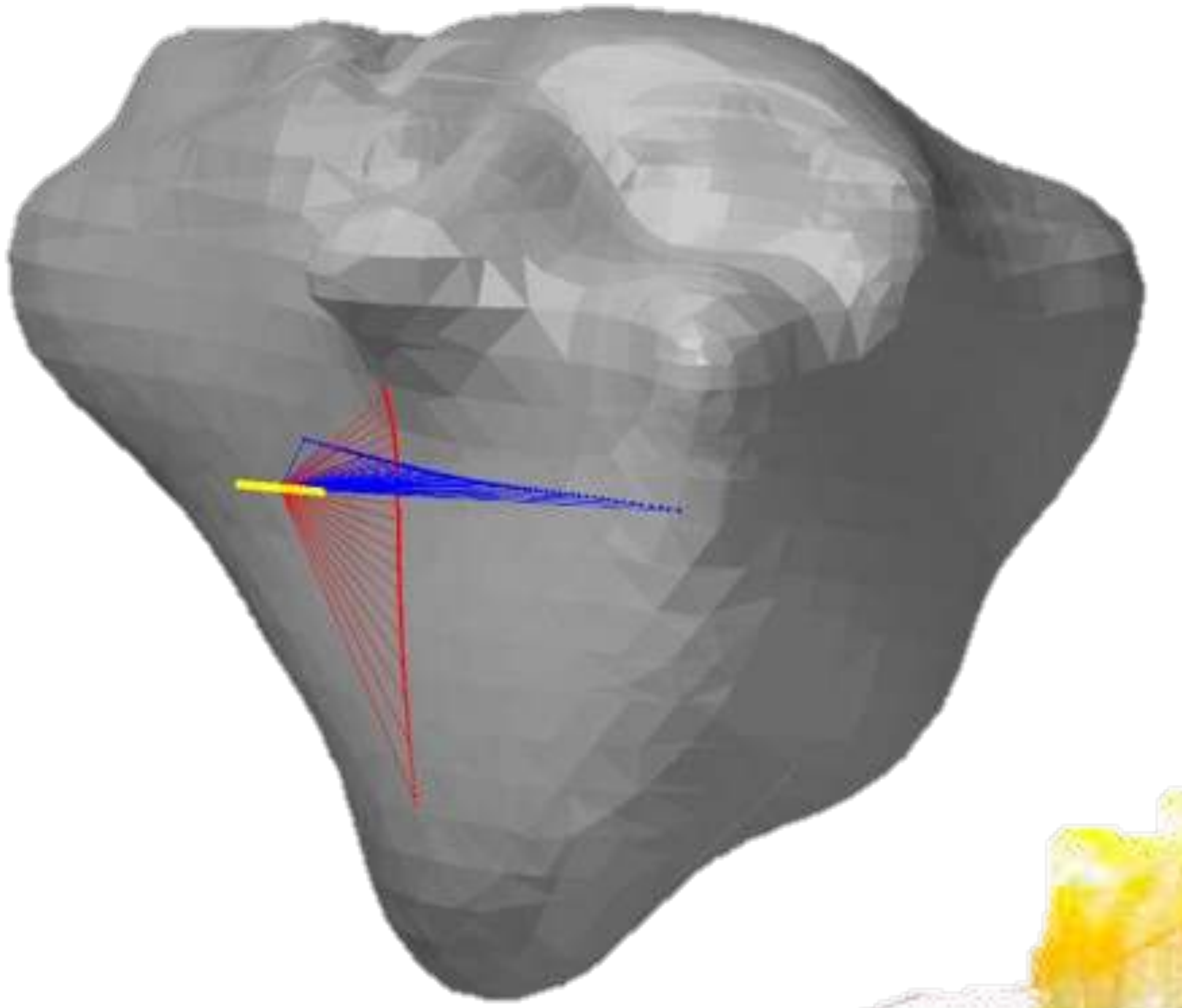






More info in [Klein & Zachmann, 2004] on cgvr.cs.uni-

Potential Application: Iceberg Visualization



Short digression about quaternions [Hamilton, 1843]

- Extension of complex numbers (does not work commutatively):

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

- Alternate notation:

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

- Axiome for the 3 imaginary units:

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

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

- From this immediately follow these laws of calculation:

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

Calculation rules for quaternions

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

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

- Conjugation: $q^* = w - a\mathbf{i} - b\mathbf{j} - c\mathbf{k}$
- Absolute (Norm): $|q|^2 = w^2 + a^2 + b^2 + c^2 = q \cdot q^*$
- Inverse of unit quaternions: $|q| = 1 \;\Rightarrow\; q^{-1} = q^*$

- Remark: sometimes it is convenient to represent the multiplication of two quaternions also by means of a matrix multiplication

$$\underset{\uparrow \atop \text{Quaternion-Mult. – not Scalarmult.!}}{q_1 \cdot q_2} = \underbrace{\begin{pmatrix} w_1 & -a_1 & -b_1 & -c_1 \\ a_1 & w_1 & -c_1 & b_1 \\ b_1 & c_1 & w_1 & -a_1 \\ c_1 & -b_1 & a_1 & w_1 \end{pmatrix}}_{\text{Q1}} \underset{\uparrow \atop \text{Written as column vector}}{q_2} = \underbrace{\begin{pmatrix} w_2 & -a_2 & -b_2 & -c_2 \\ a_2 & w_2 & c_2 & -b_2 \\ b_2 & -c_2 & w_2 & a_2 \\ c_2 & b_2 & -a_2 & w_2 \end{pmatrix}}_{\text{Q2}} \underset{\uparrow \atop \text{Column vector}}{q_1}$$

- In addition:

$$q_1 \cdot q_2^* = \underset{\uparrow \atop \text{Matrix to quaternion } q_2^*}{Q_2^*} q_1 = Q_2^\mathsf{T} q_1$$

Embedding the 3D vektor space in $\mathbb{H}$

- The vector space $\mathbb{R}^3$ can be embedded in $\mathbb{H}$ like this:

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

- Definition:
  quaternions of the form $(0, \mathbf{v})$ are called pure quaternions

Representation of rotations using quaternions

- Let be given Axis & Angle$(\varphi, \mathbf{r})$ with $\|\mathbf{r}\| = 1$
- Definie the corresponding quaternion as

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

- Observation: $|q| = 1$

- Theorem: Rotation by means of a quaternion
  Let $\mathbf{v} \in \mathbb{H}$ be a pure quaternion (= vector in 3D) and $q \in \mathbb{H}$ a unit quaternion.
  Then the figure

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

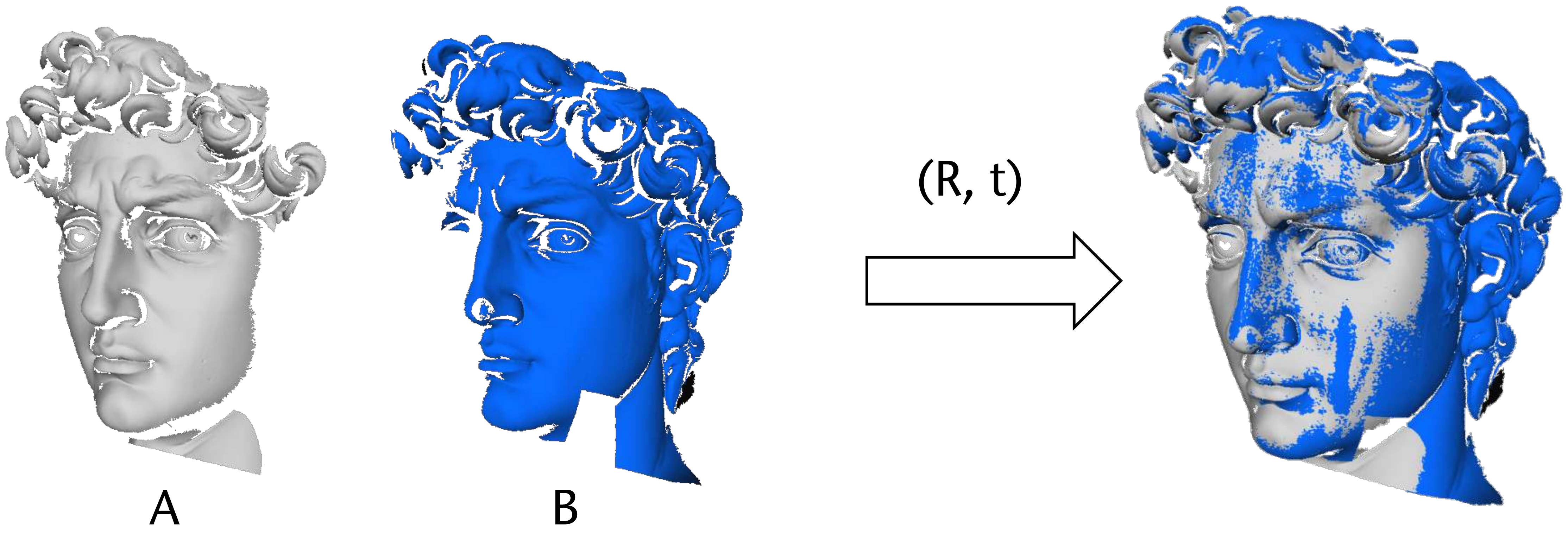
  describes a (right-handed) rotation of **v** around the angle φ and axis **r** are determined, where the pure quaternion **v**' arises.

Alignment / Registration of Shapes

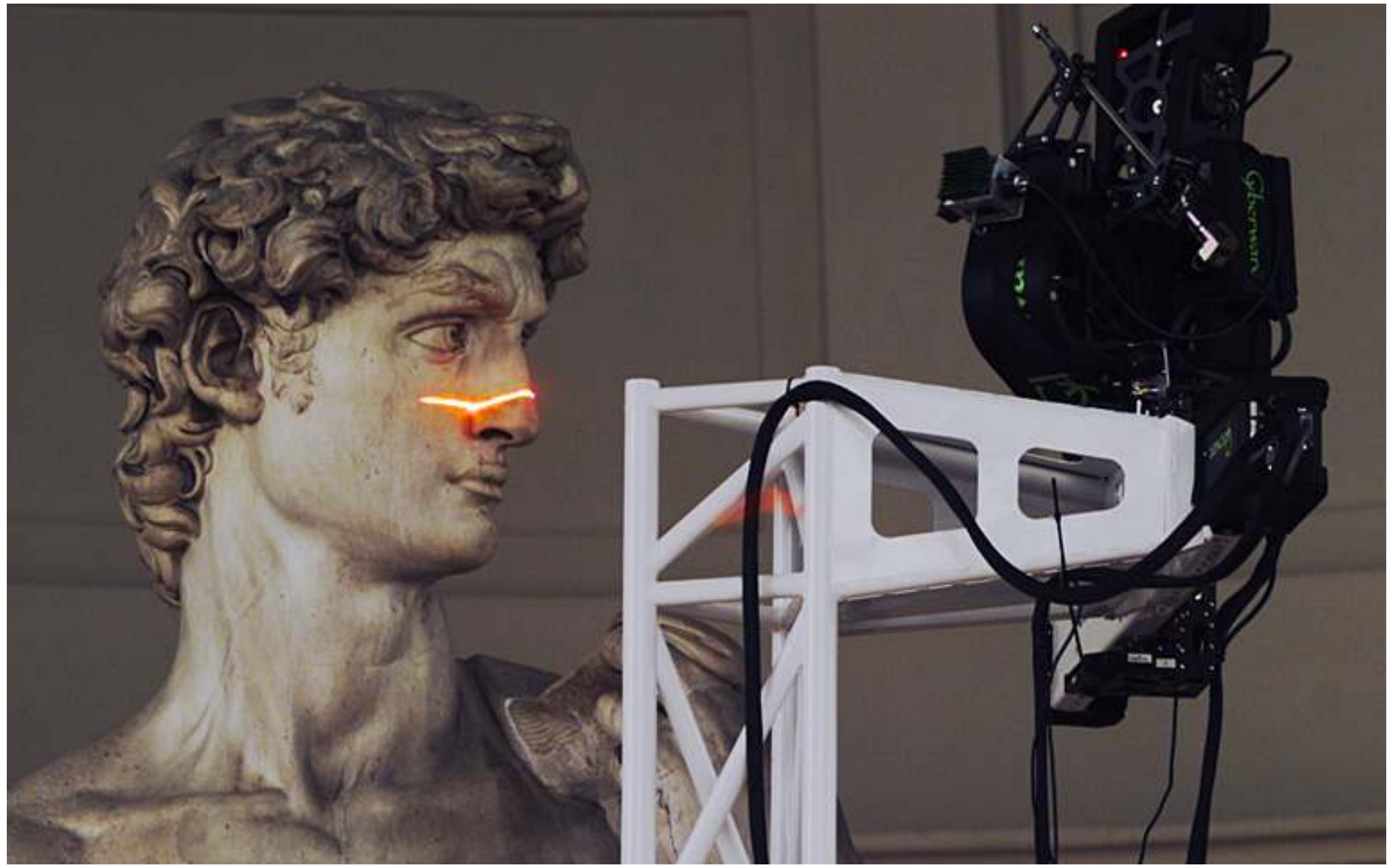
- See manuskript

Shape Registration

- 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

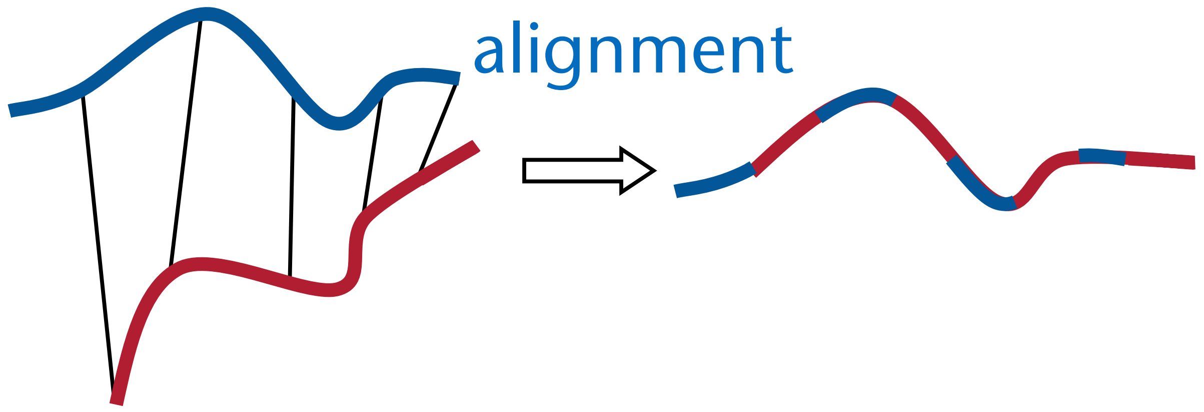


Motivation: Registration of Point Clouds

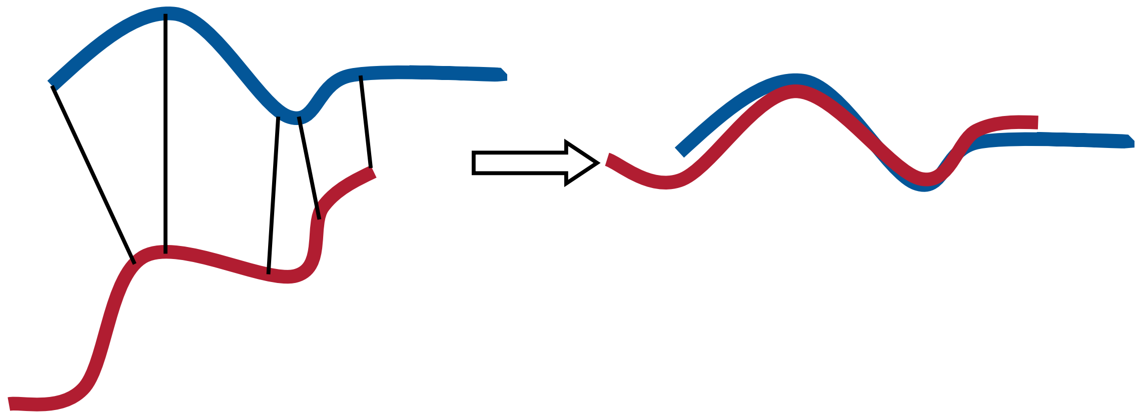


Approach

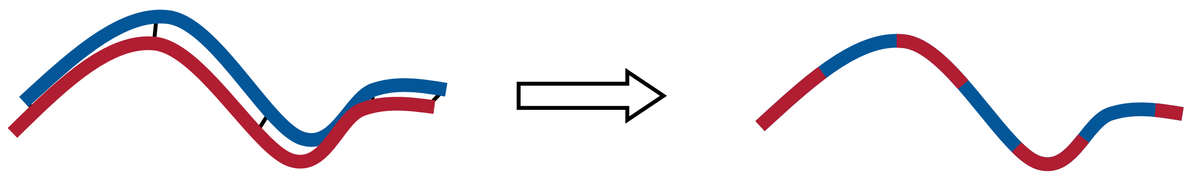
- We know: if correct correspondences are known, then we can find a correct relative rotation/translation (alignment)



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



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



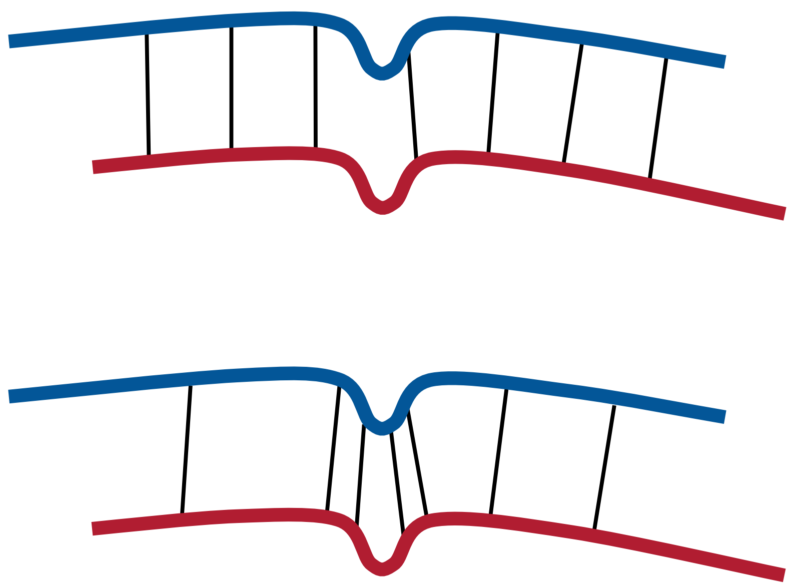
The Iterative Closest Point Algorithm (ICP)

```
repeat
  forall bᵢ in B: find NN in A  ⟶  Y ⊆ A
  compute optimal alignment transformation (R,t) from B to Y
  rotate/translate B
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 ("warm-starting")
  - Makes the initial ball for the "ball overlaps bounds" test (hopefully) relatively small
  - The traversal does not descend into subtrees far away from true NN

Variants / Optimizations

- Work only on a subsample of the points (of one or both shapes):
    - Poisson disk subsampling
    - 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 corresponding point pairs to "compatible" points (color, intensity, normals, curvature, ...) [Pulli 99]

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

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

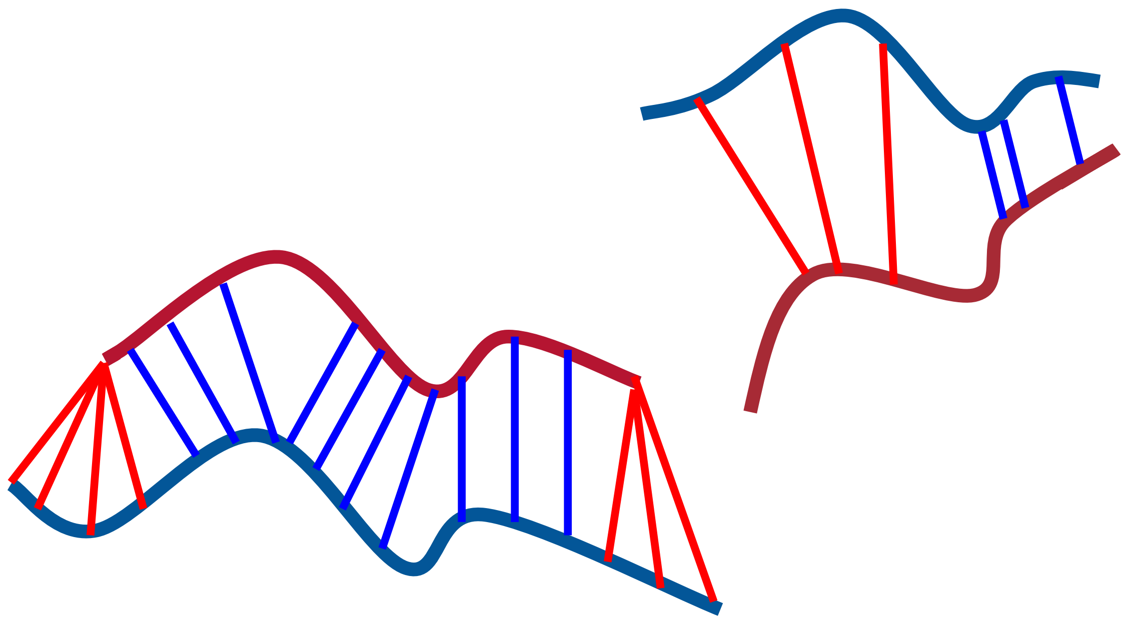
- As weight, you could consider:
  - Distance between corresponding points

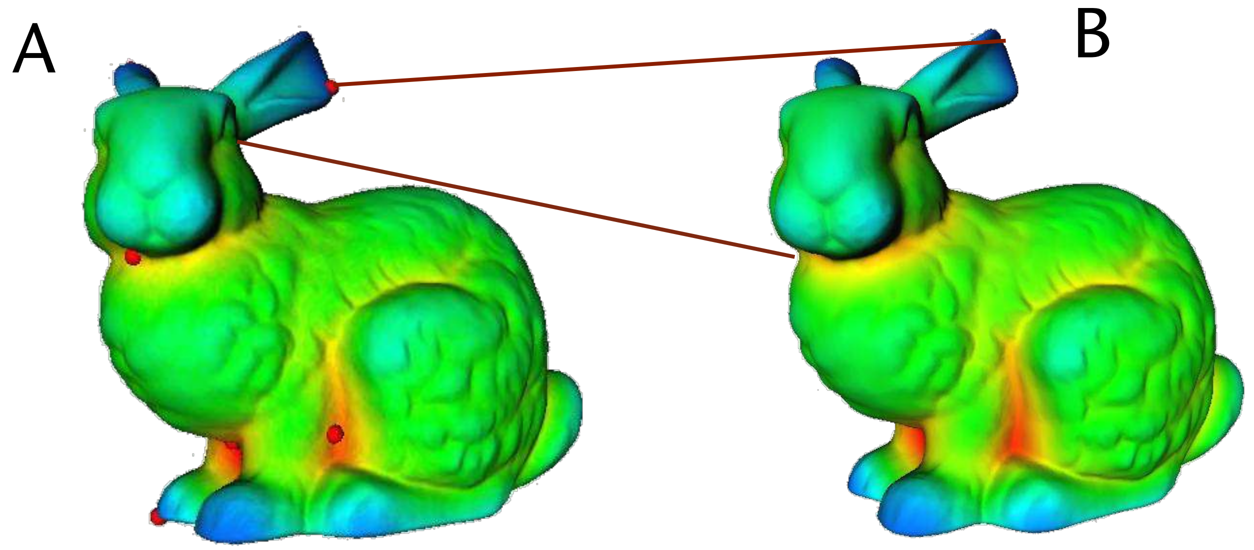
$$w_i = 1 - \frac{\| b_i - a_i \|}{\text{max dist}}$$

  - Scanner uncertainty

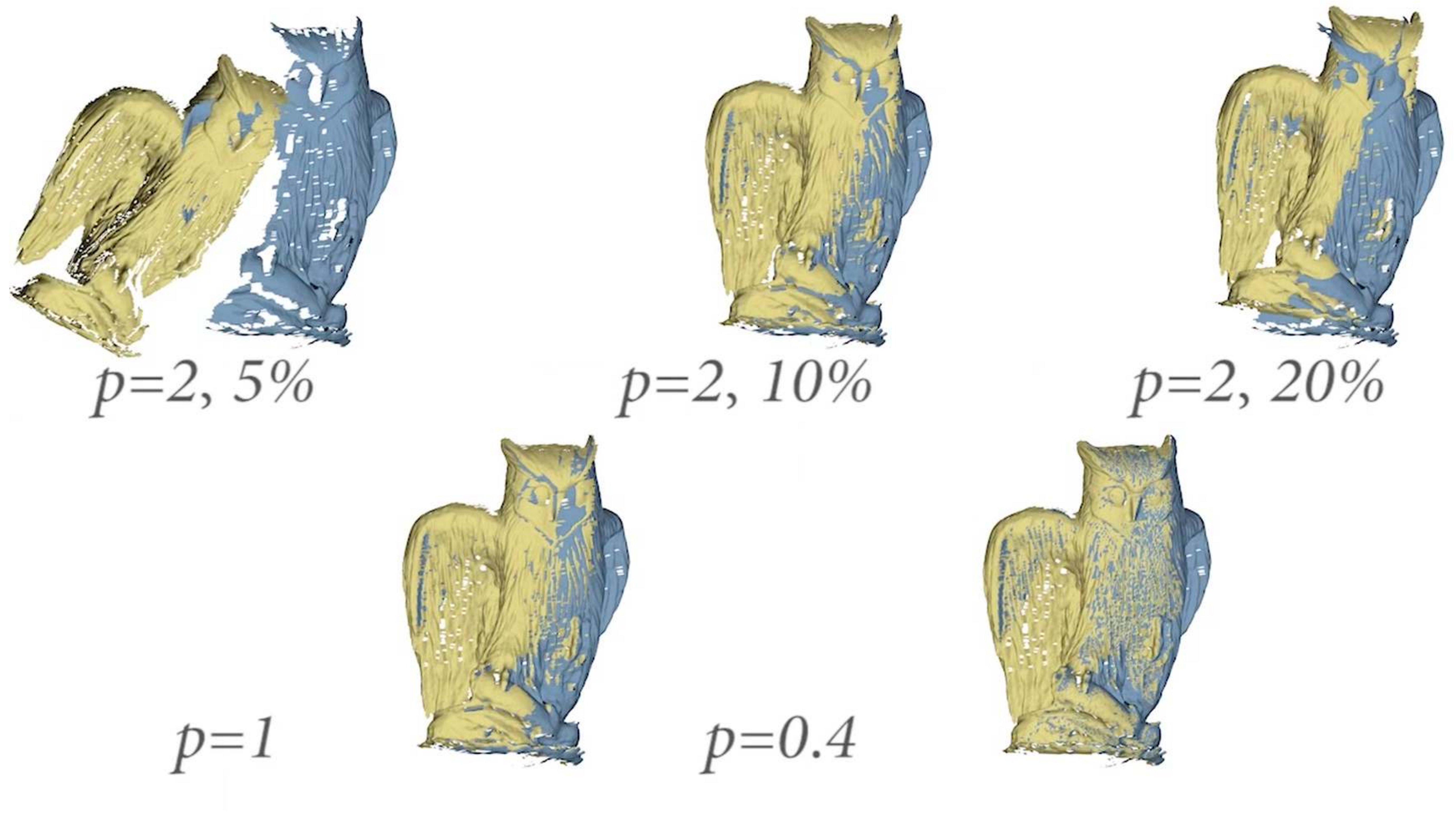
- Reject "bad" point pairs:
  - Reject pairs whose distance is in the top *x* % of all distances
  - Reject points at the "borders" of the shapes
  - Reject pairs that are *not consistent* with their neighboring pairs [Dorai 98]:
  - Two pairs (a_1,b_1) and (a_2,b_2) are not consistent if

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



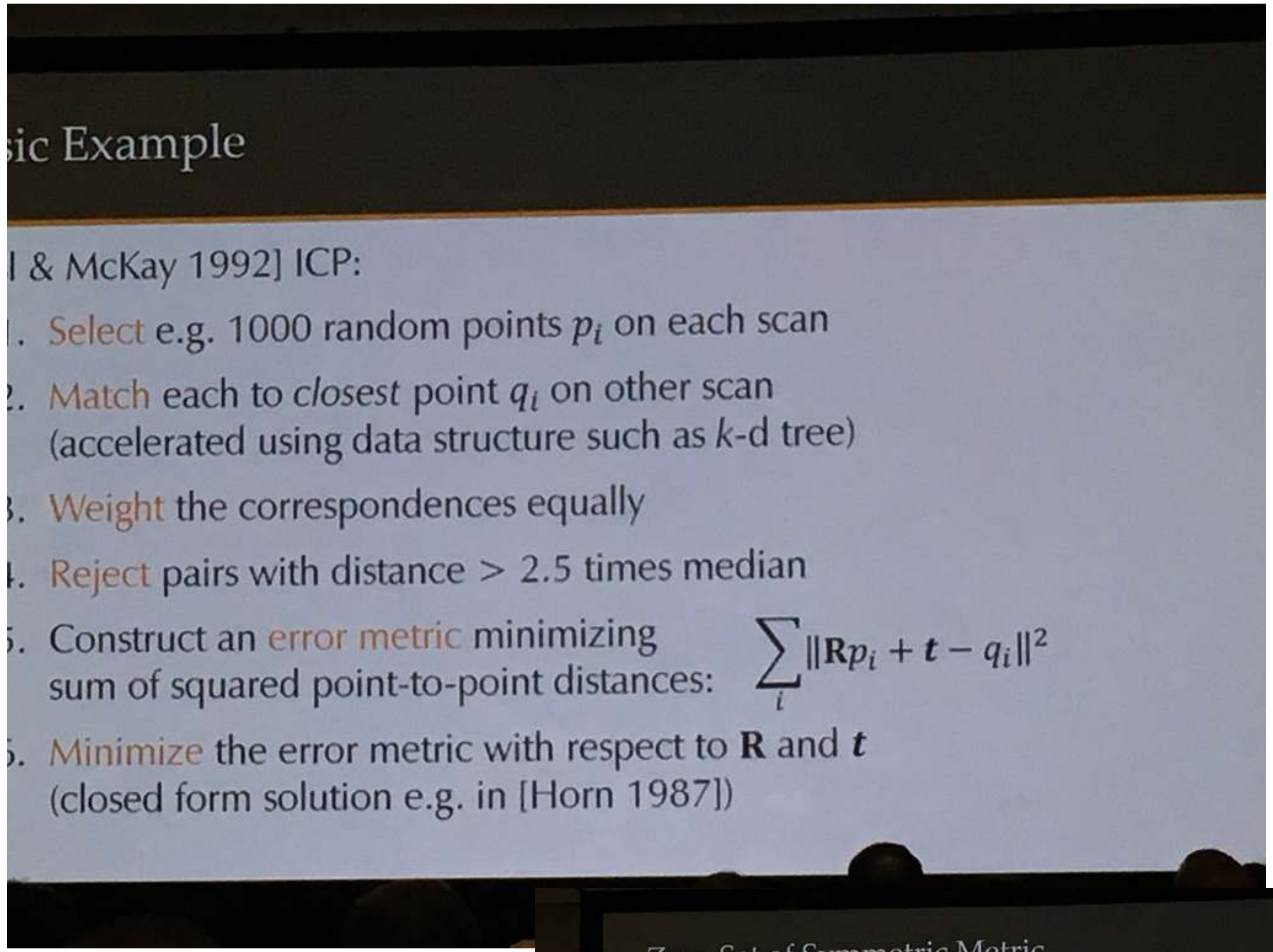


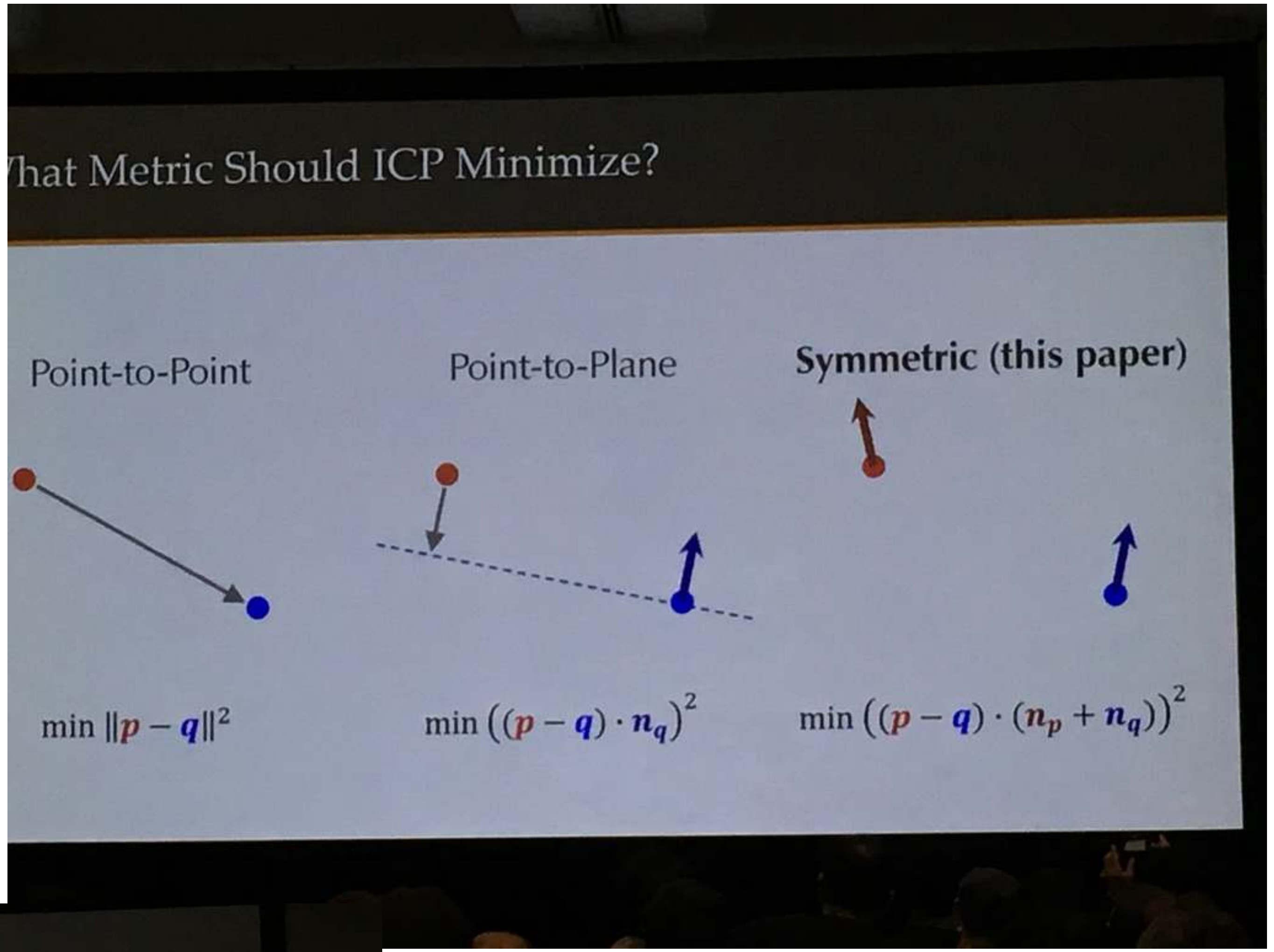
Experiments with Various Rejection Rates, and Different p-Norms

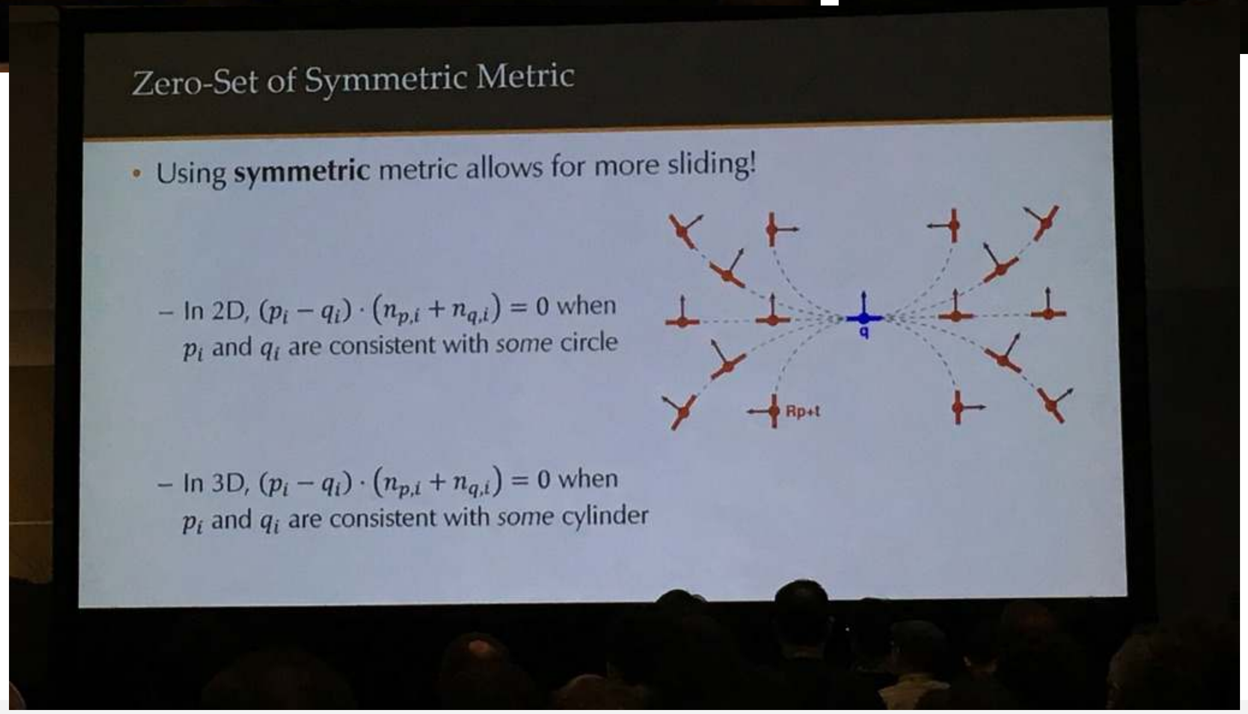


Sofien Bouaziz, Andrea Tagliasacchi, Mark Pauly: "Sparse Iterative Closest Point". Symposium on Geometry Processing 2013

From the Siggraph 2019







Stackless kd-tree traversal for ray-tracing

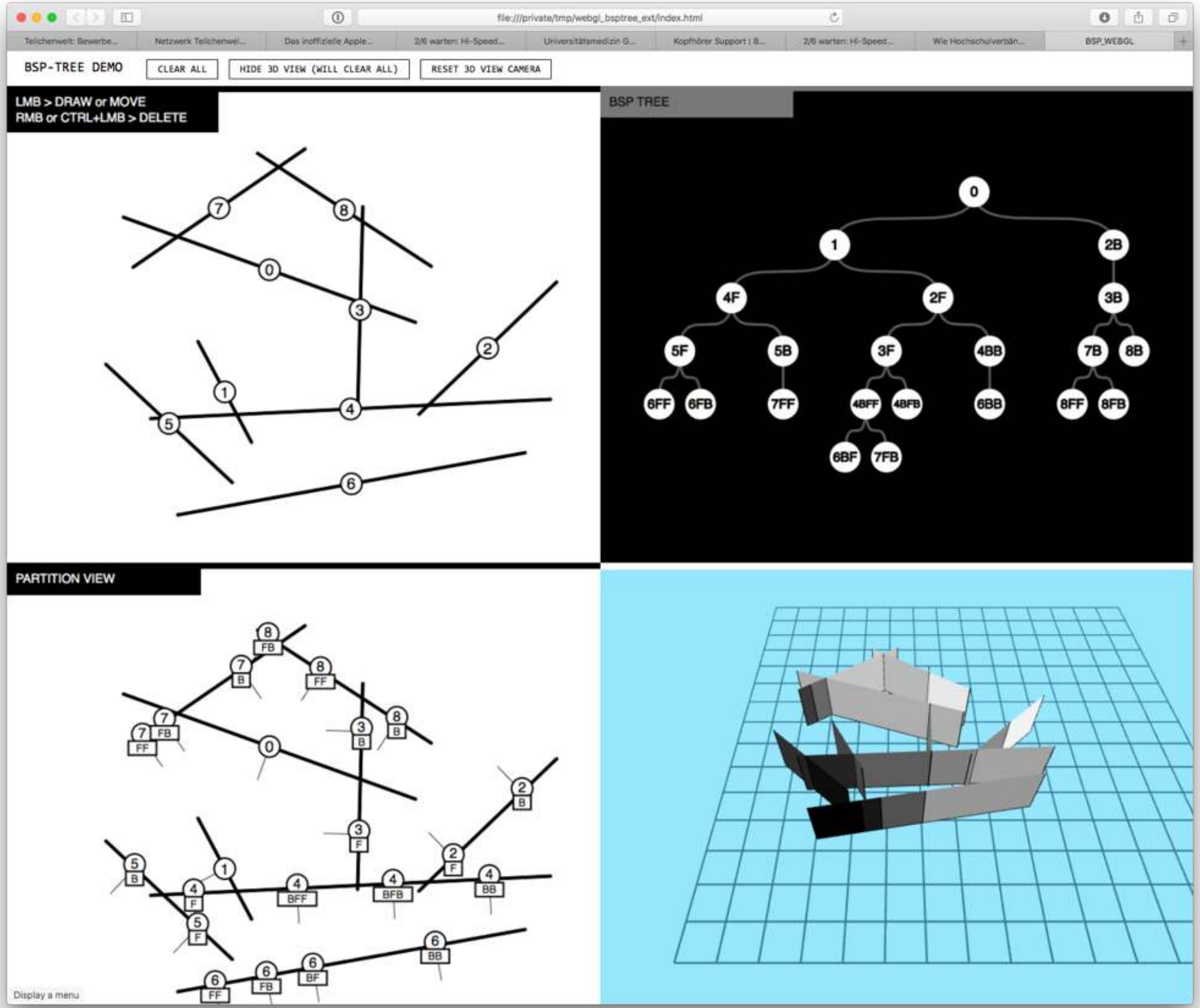
Stefan Popov, Johannes Günther, Hans-Peter Seidel, and Philipp Slusallek.
Nvidia GeForce 8800GTX, CUDA, 2007.

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

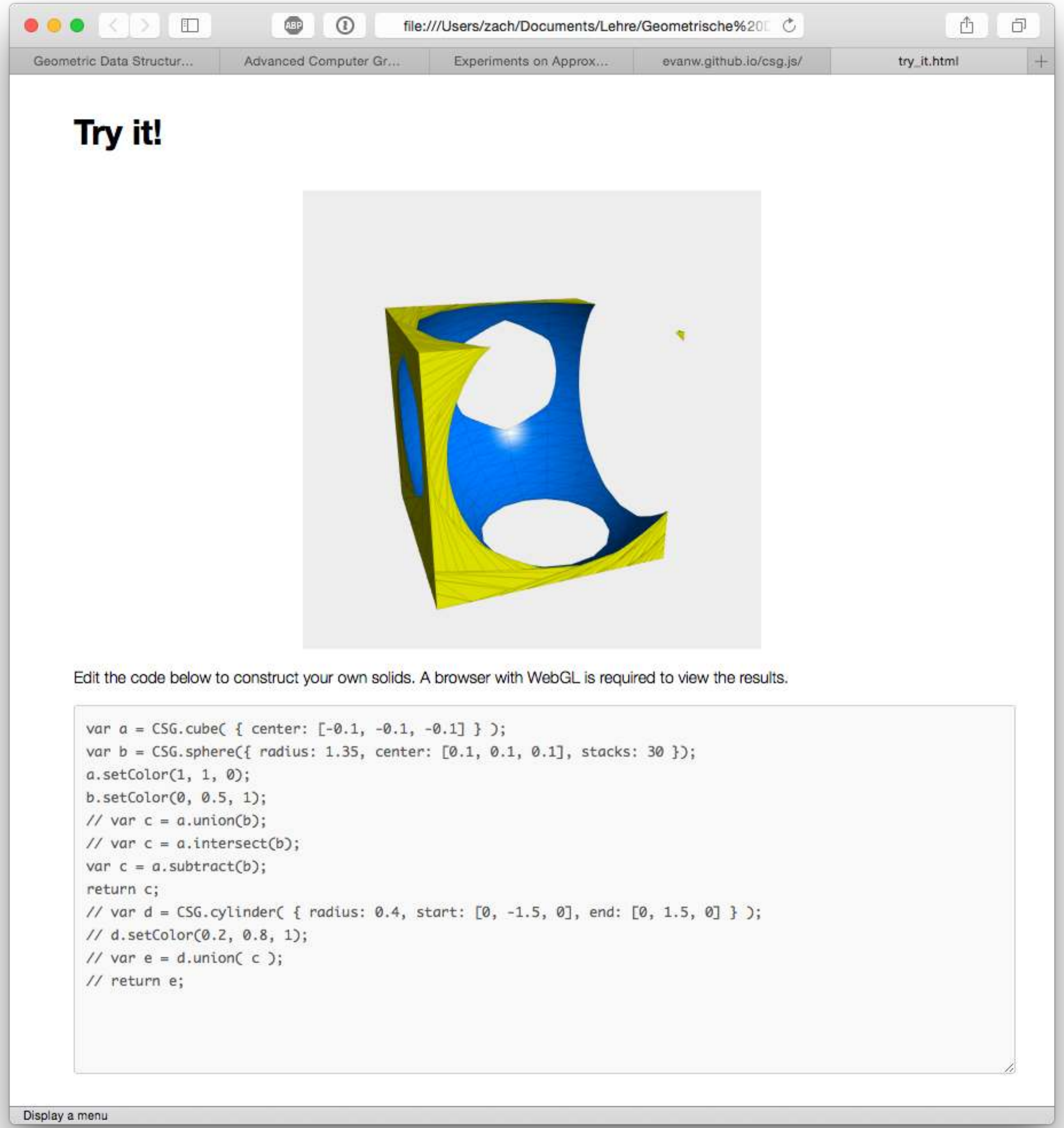
Real-Time KD-Tree Construction on Graphics Hardware

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

BSP Demo


BSP-TREE DEMO
CLEAR ALL
HIDE 3D VIEW (WILL CLEAR ALL)
RESET 3D VIEW CAMERA
LMB > DRAW or MOVE
RMB or CTRL+LMB > DELETE
BSP TREE
PARTITION VIEW
Display a menu

Constructive Solid Geometry (CSG) using BSP's



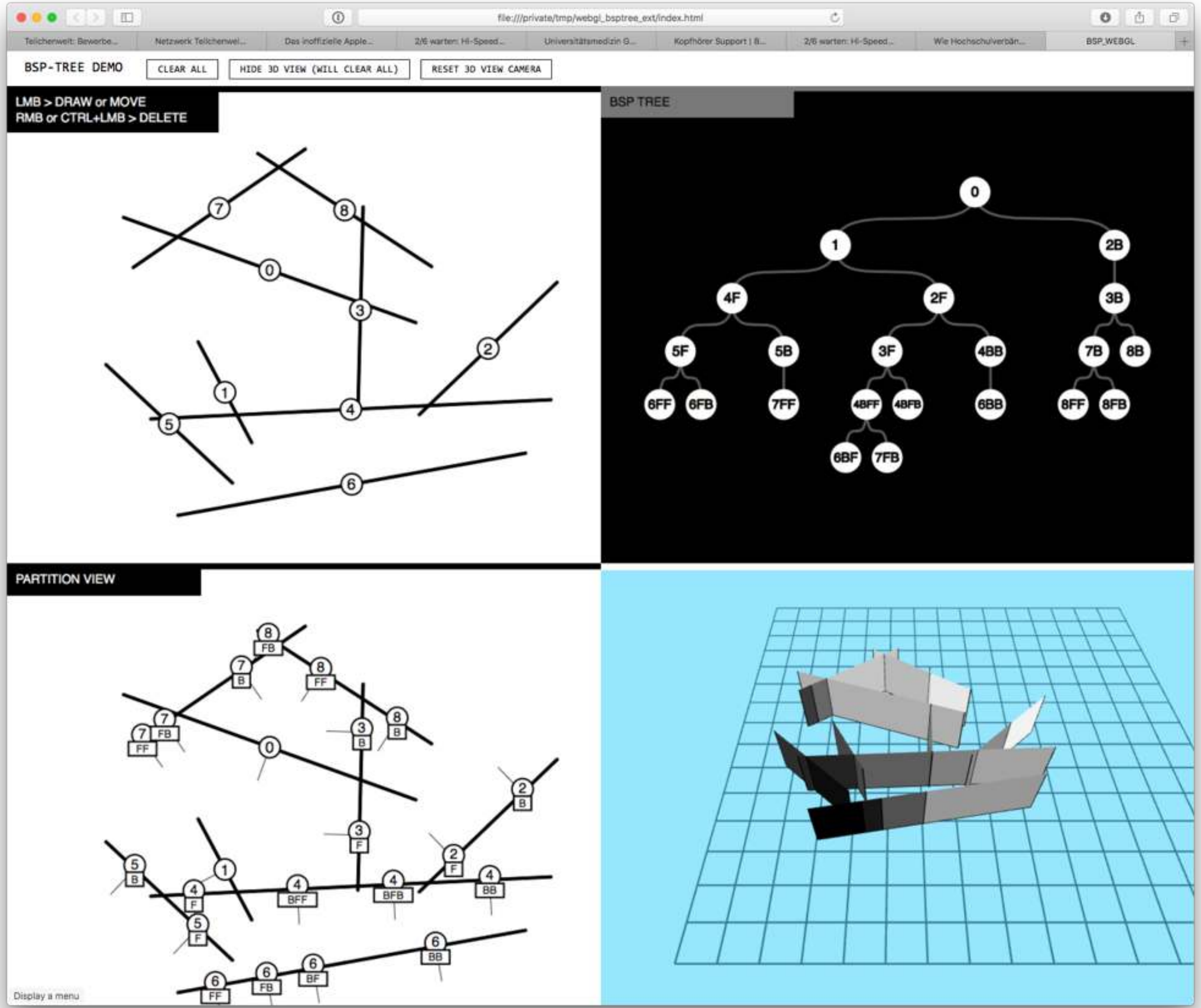
http://evanw.github.io/csg.js/

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

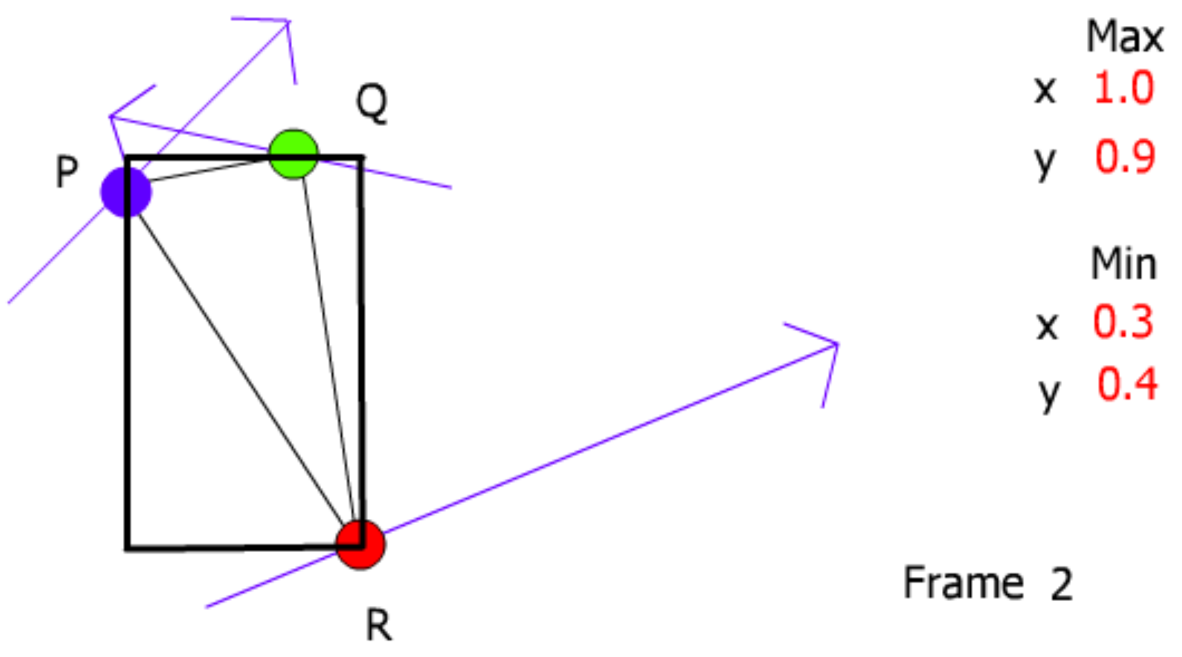
http://bastian.rieck.ru/uni/bsp/

Kinetic Data Structures Motivation: BSP Tree with Moving Planes

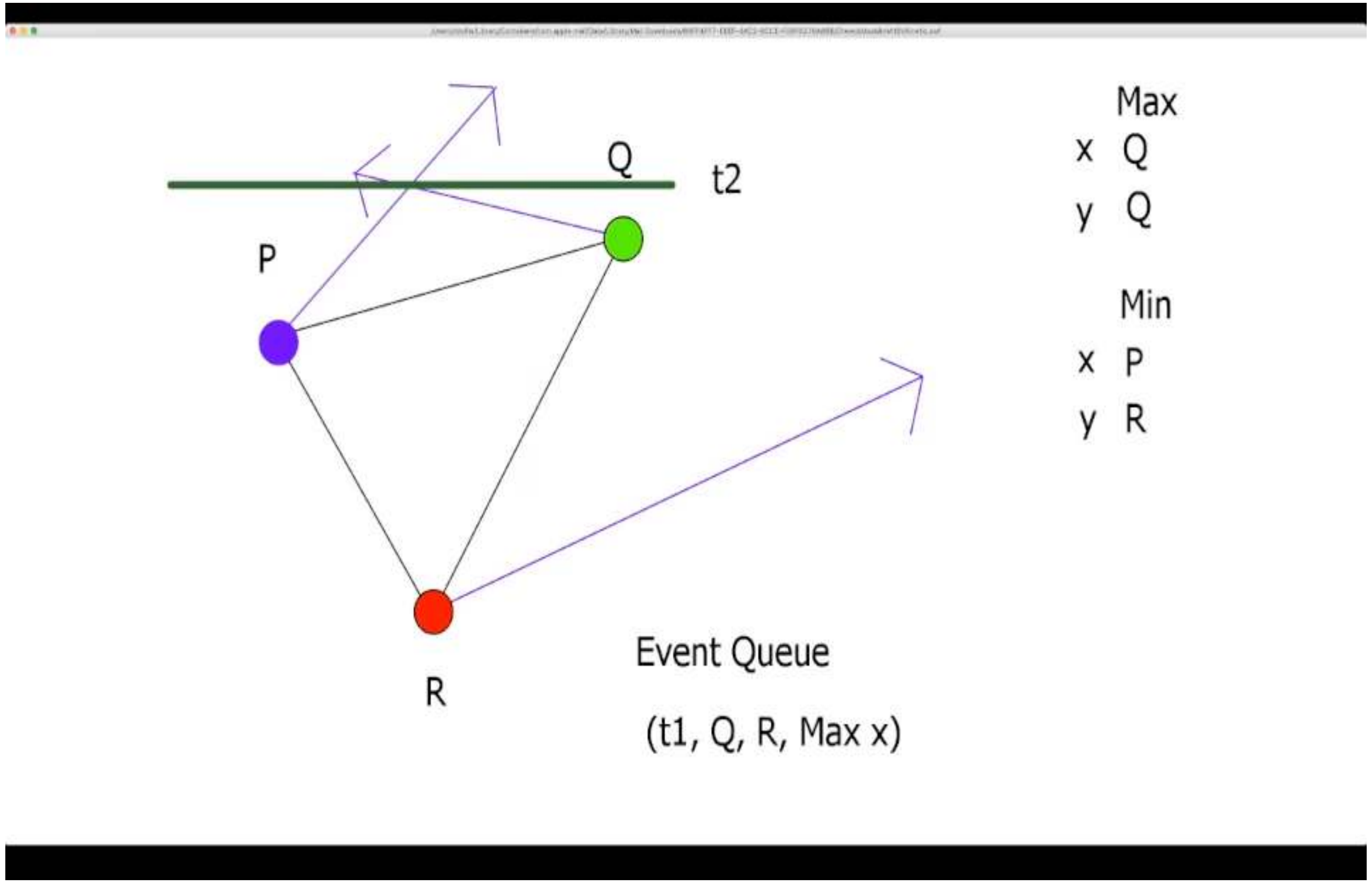


Kinetic Data Structures – Motivation

Brute force update of bbox

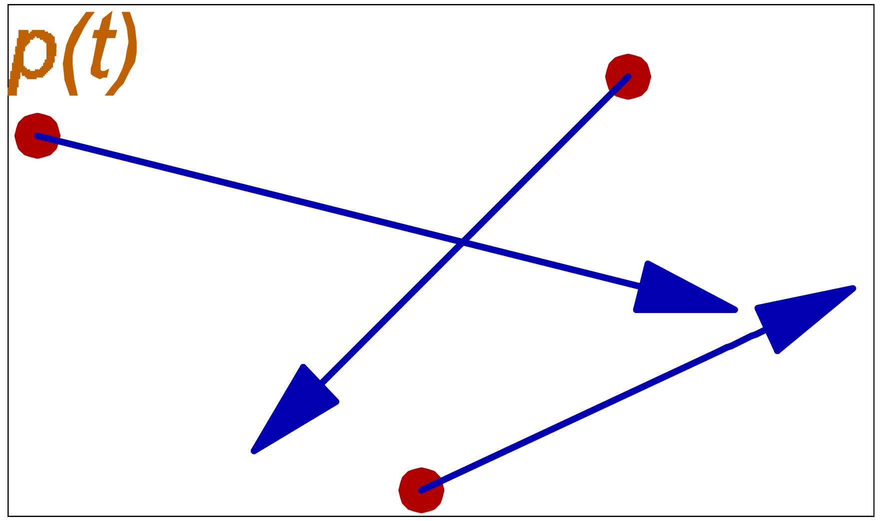


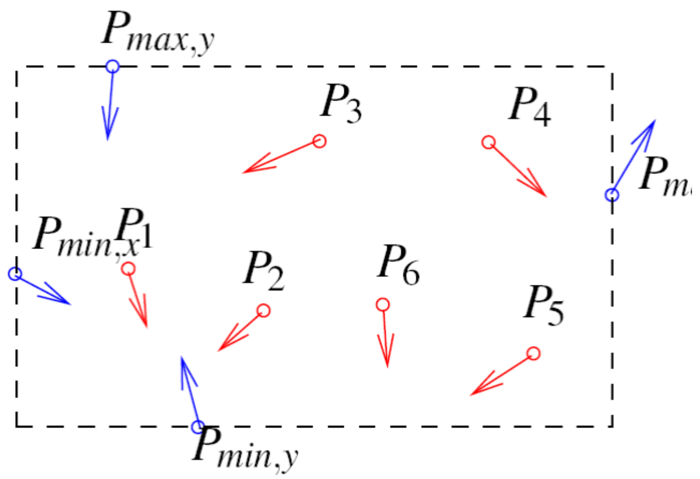
Kinetic update of bbox

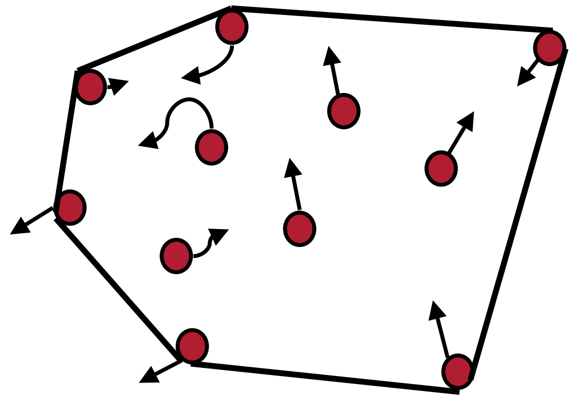


General Concept of Kinetic Data Structures (KDS)

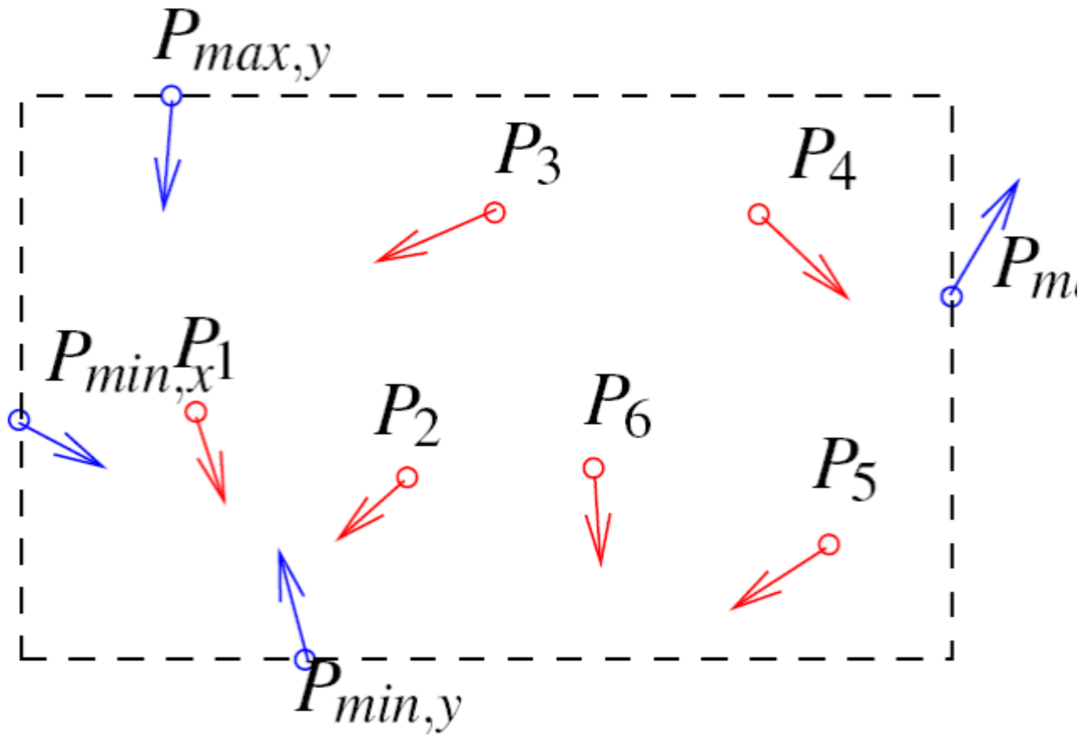
- Given:
  - A number of objects (points, lines, polygons, boxes, ...)
  - A flight path for each of these objects, given by an algebraic function
    - In practice, we assume linear motion
- Attribute = the task / purpose of a KDS
  - Examples: bbox of a set of points, kd-tree over a set of points, convex hull, ...

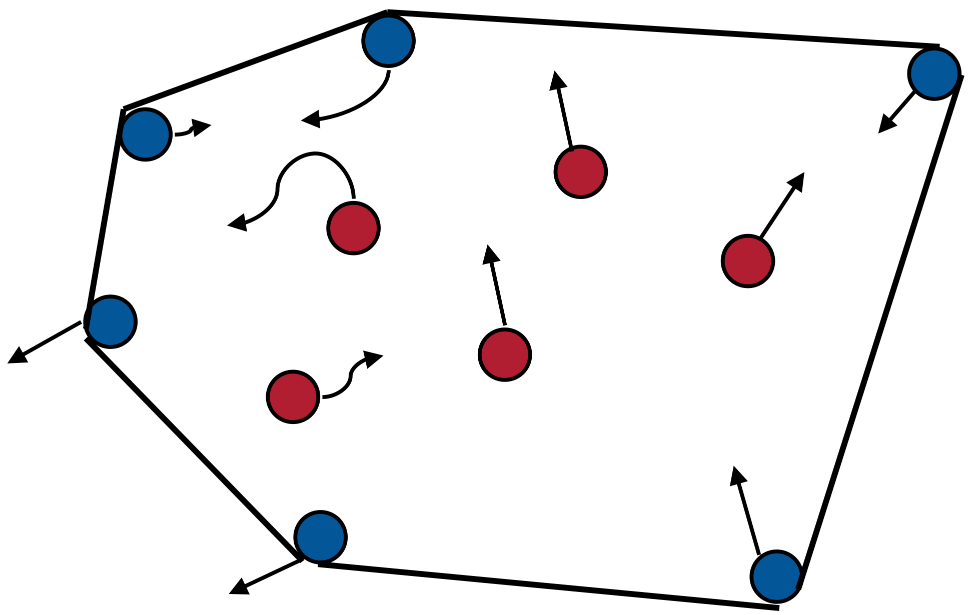




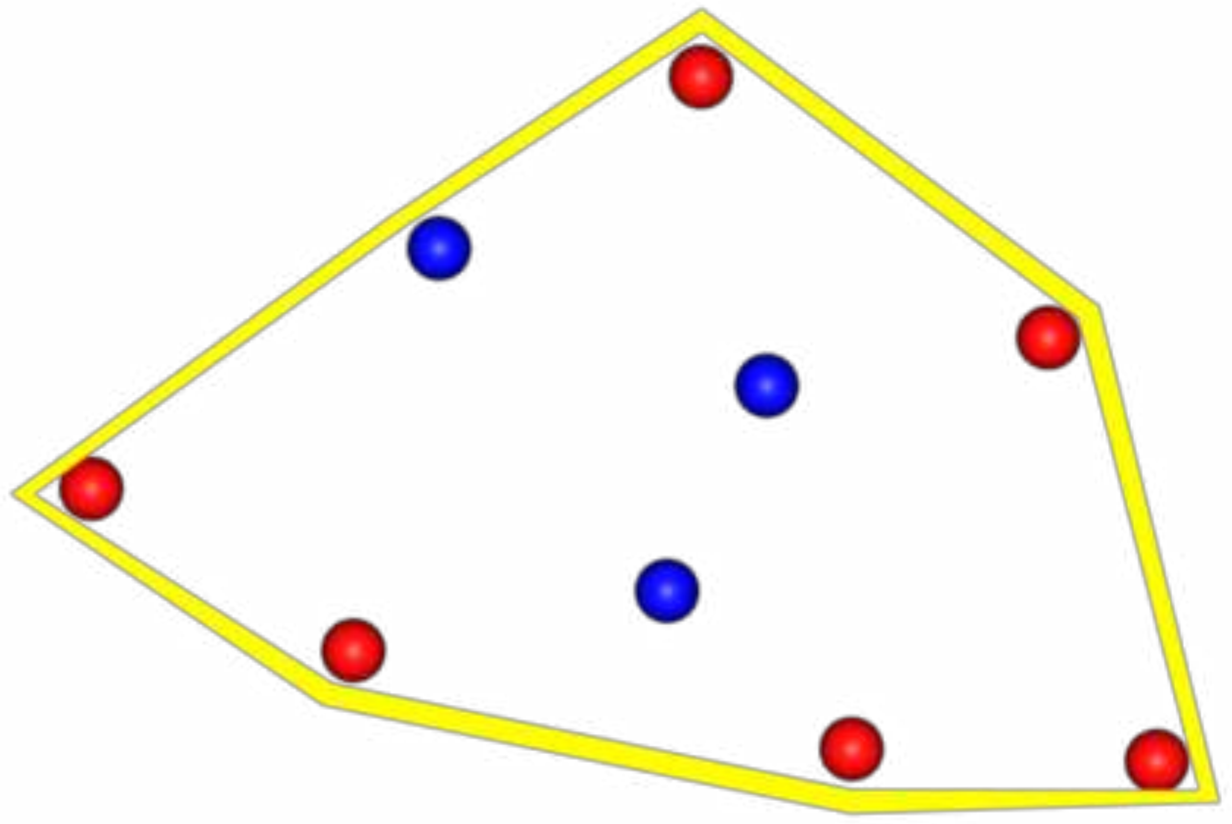


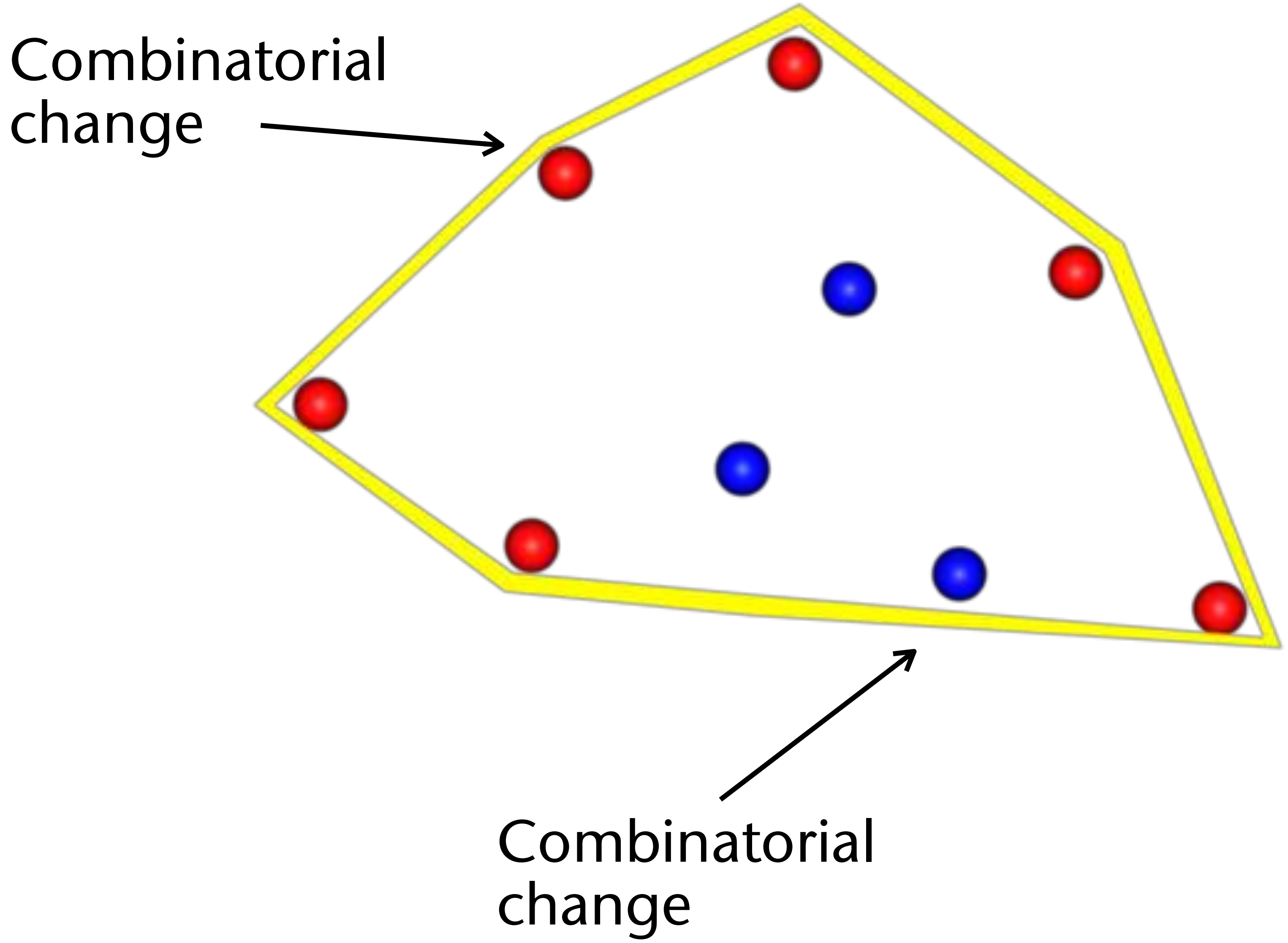
- Combinatorial structure = "everything that describes the attribute *except* concrete coordinates"
  - Examples:
    - Convex hull: those points that form the vertices (corners) of the convex hull
    - Bbox: those points that realize the min/max on at least one of the coord axes
    - Kd-tree: all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

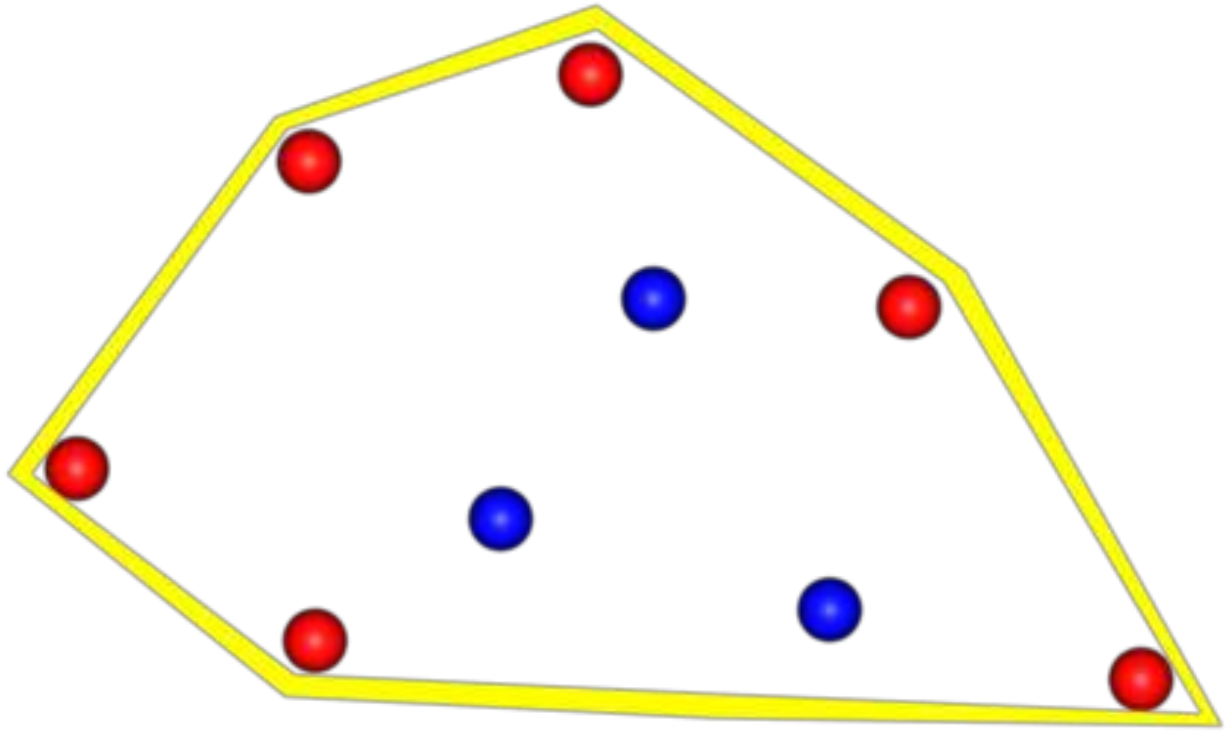


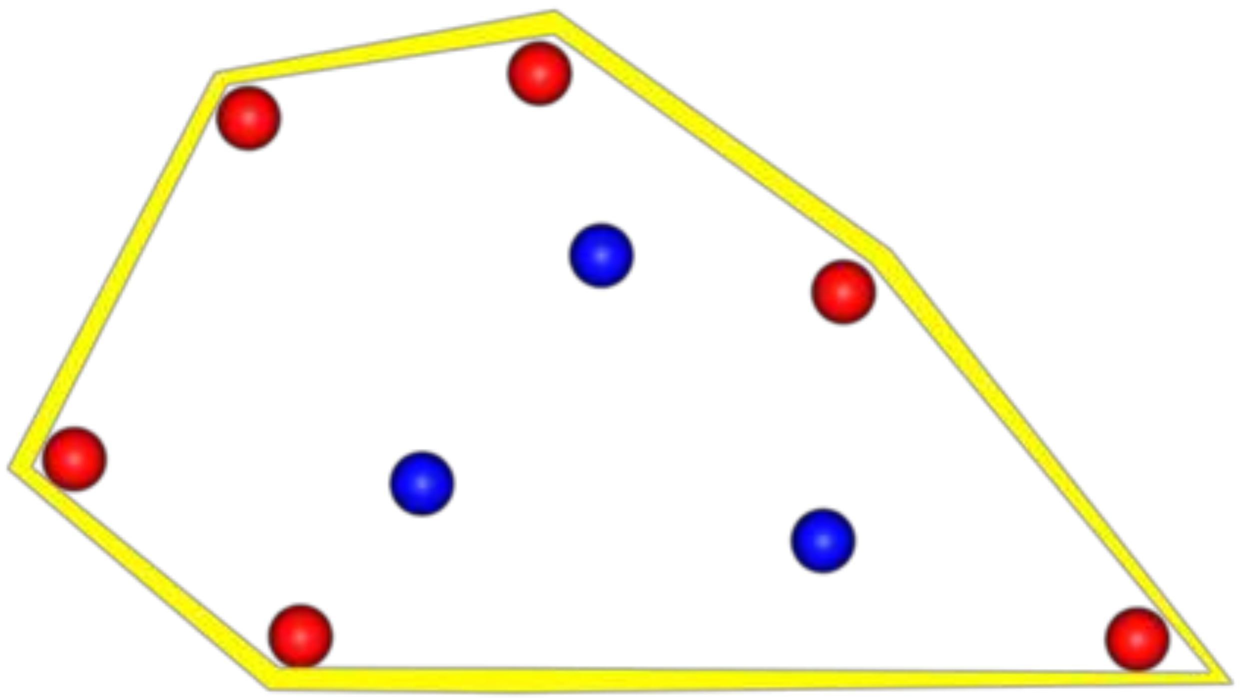


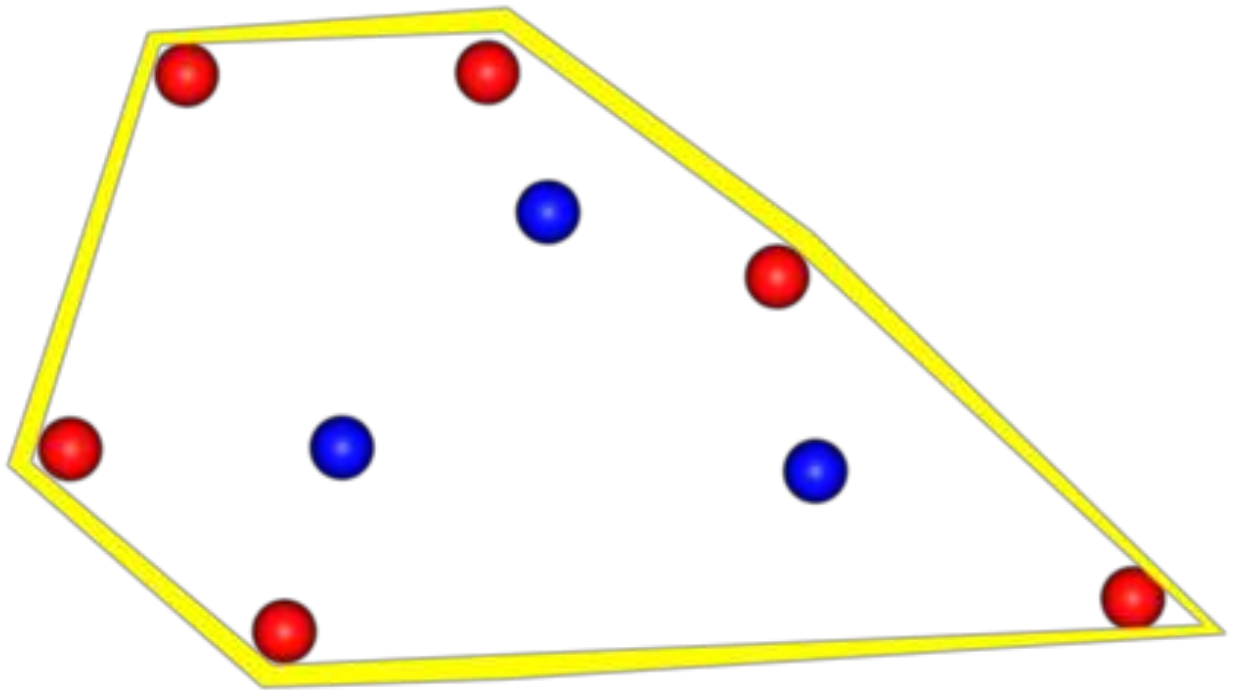
Example of Change of Combinatorial Changes

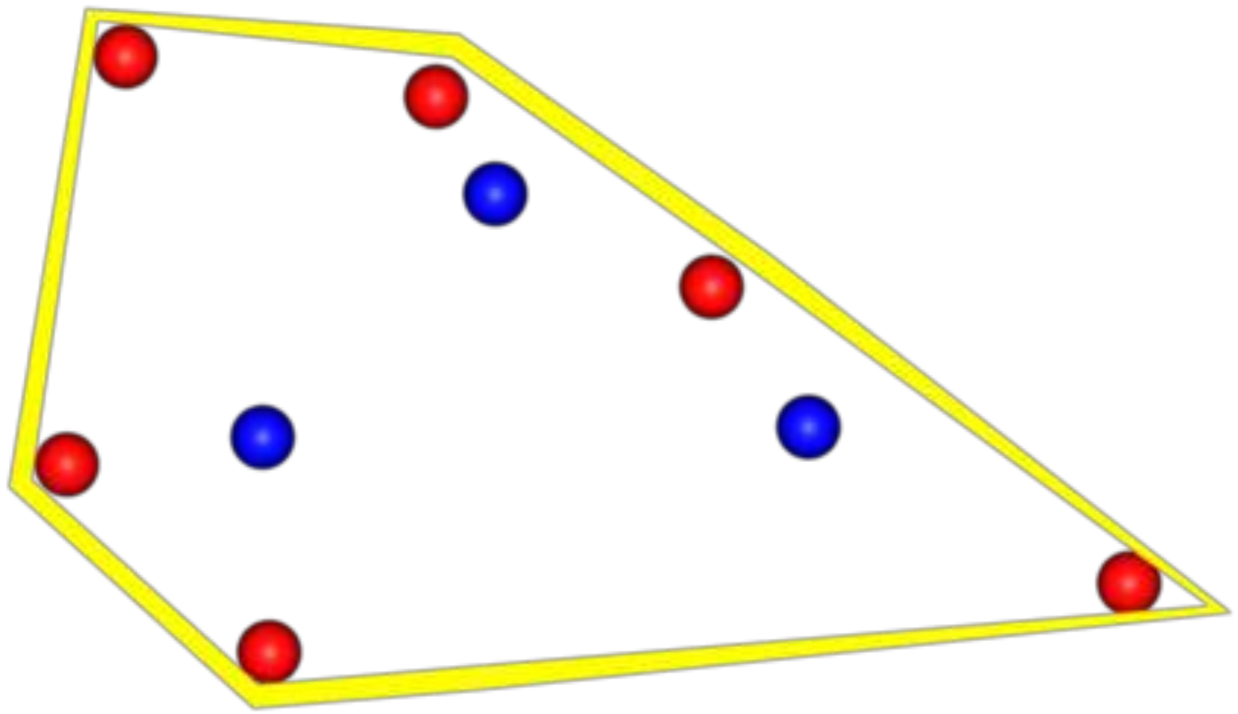


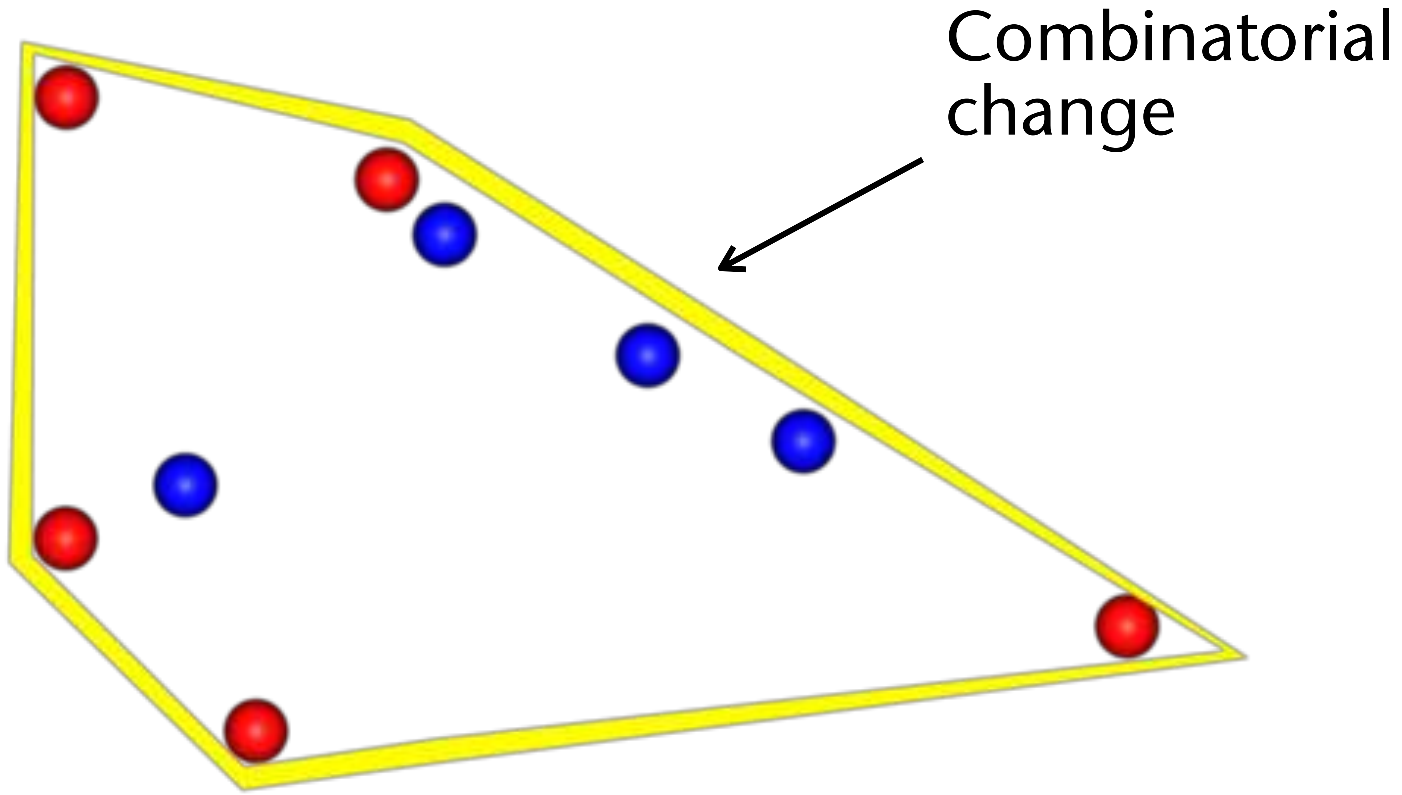

Combinatorial
change
Combinatorial
change

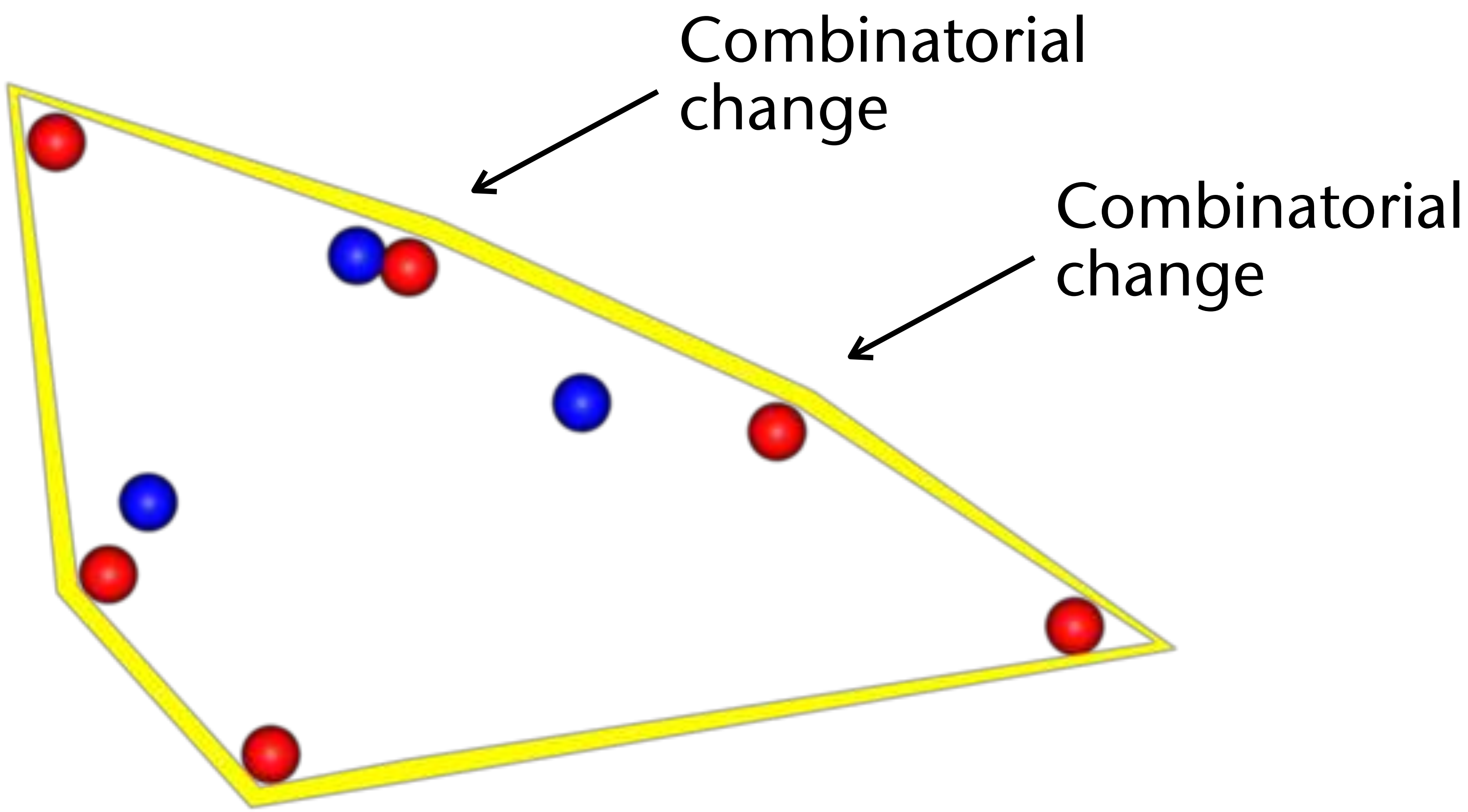


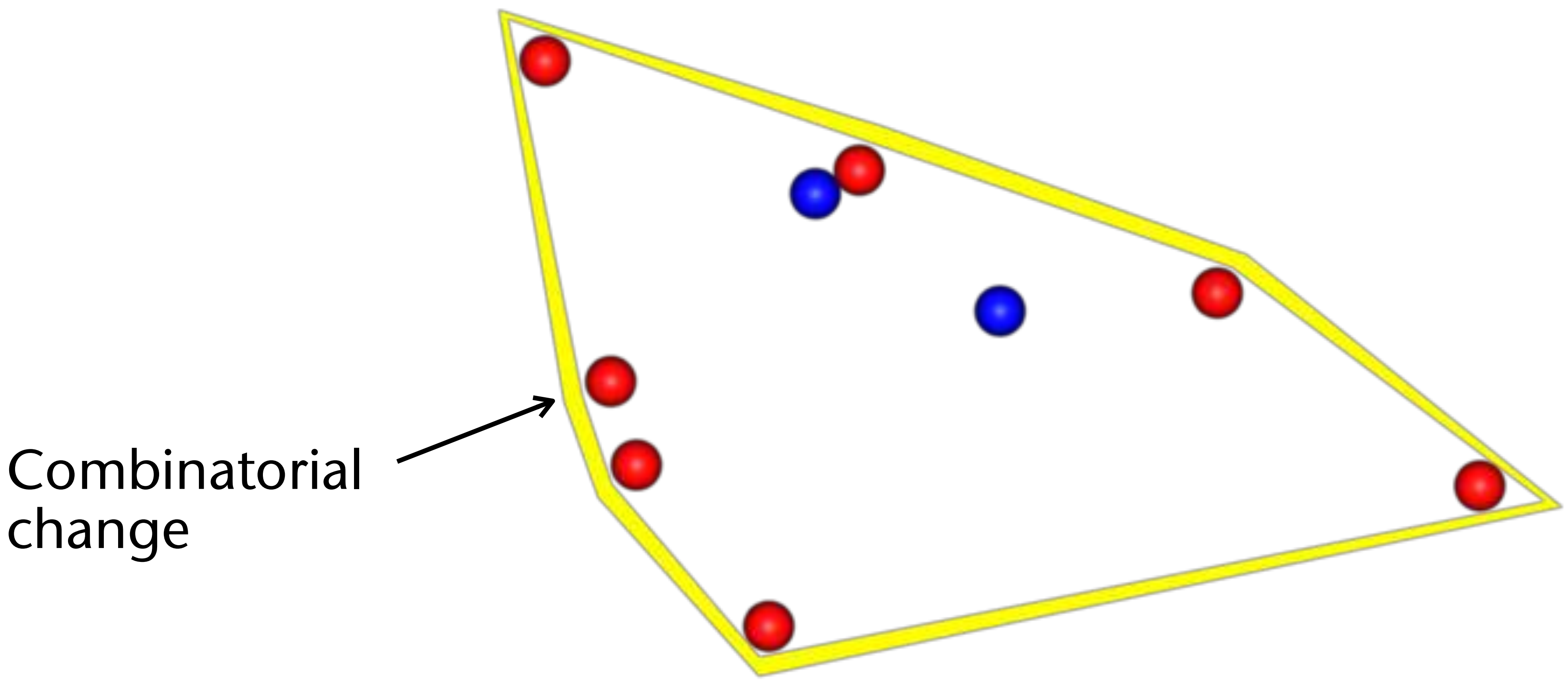


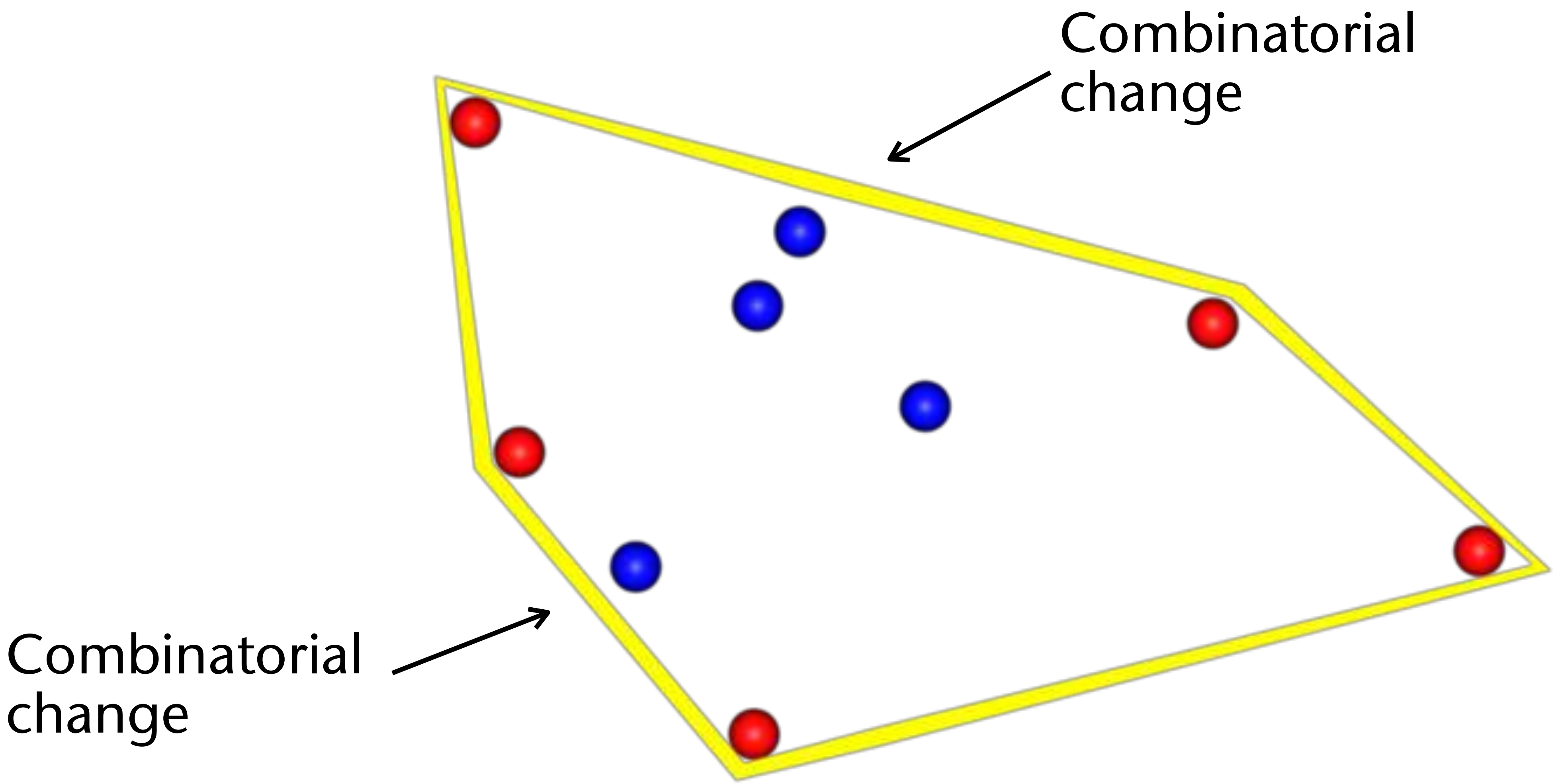





Combinatorial
change


Combinatorial change
Combinatorial change


Combinatorial
change


Combinatorial
change
Combinatorial
change

More Definitions for KDS

- Certificate = simple geometric relation (a.k.a. geometric predicate) involving a few of the objects
  - Example: $\mathbf{p} \cdot \mathbf{n} < 0$, where $\mathbf{p}$ is an input point and $\mathbf{n}$ is a normal (stored somewhere in the KDS)
    - E.g.: the plane equations of the faces of the convex hull of a set of points
- 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
    - In case of convex hull: one of the points leaves the current convex hull, i.e., "crosses" over a plane
  - Internal event = event where the combinatorial structure remains the same, but the set of certificates changes
    - Convex hull: are there any internal events?
- Kinetic data structure (KDS) for a geometric attribute =
  1. A set of certificates that is true whenever the combinatorial structure of the attribute is valid; as well as
  2. A set of rules (algorithm) for repairing the attribute and the set of certificates in case of an event

Are There Internal Events in the KDS for the Convex Hull in 2D?

https://www.menti.com/bejuo8zvux

Generic Main Loop to Maintain a KDS

```
initialize the attribute for the input objects
initialize the set of certificates
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:
  do computations using the KDS ...
  update time t_new := t_old + Δt
  while timestamp(front event in queue) <= t_new:
    pop 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 a graphical system, the main loop might look like this …

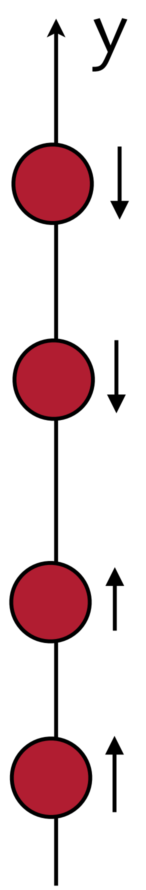
```
initialization ...
while simulation runs:
  determine time t of next rendering
  get foremost event from the event queue
  while timestamp(event) < t:
    update KDS
    get next event from the event queue
  use the attribute of the KDS (e.g., bbox, kd-tree, BVH, …)
  render scene
```

Performance Measures for KDS

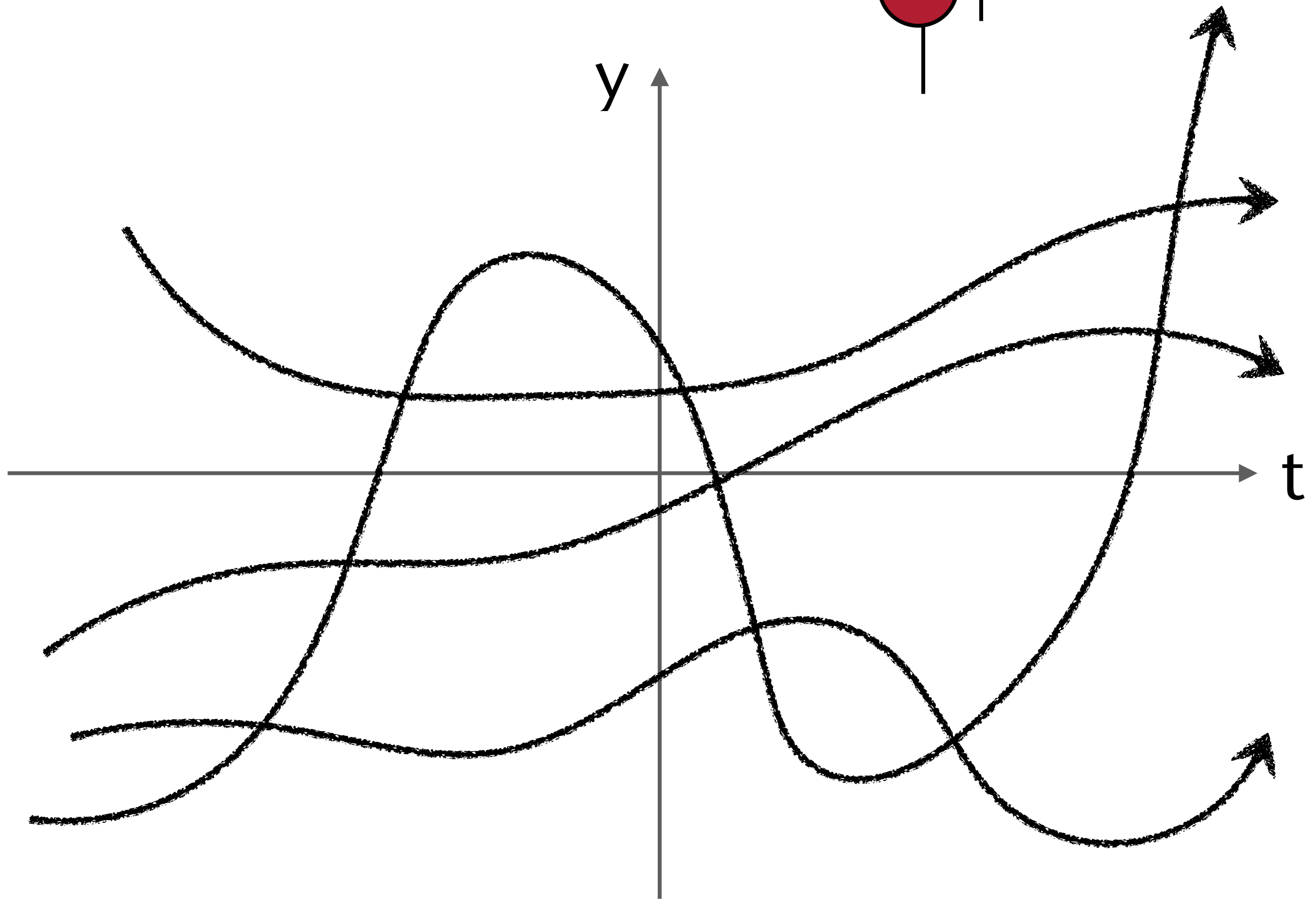
1. Responsiveness:
   A KDS is responsive, if the cost to update the set of certificates and the attribute in case of an event is "small"
   - Usually, "small" = $O(\log^S n)$ or $O(n^\varepsilon)$
2. Efficiency:
   A KDS is efficient, if the ratio of #(total events) / #(external events) is small
   - 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

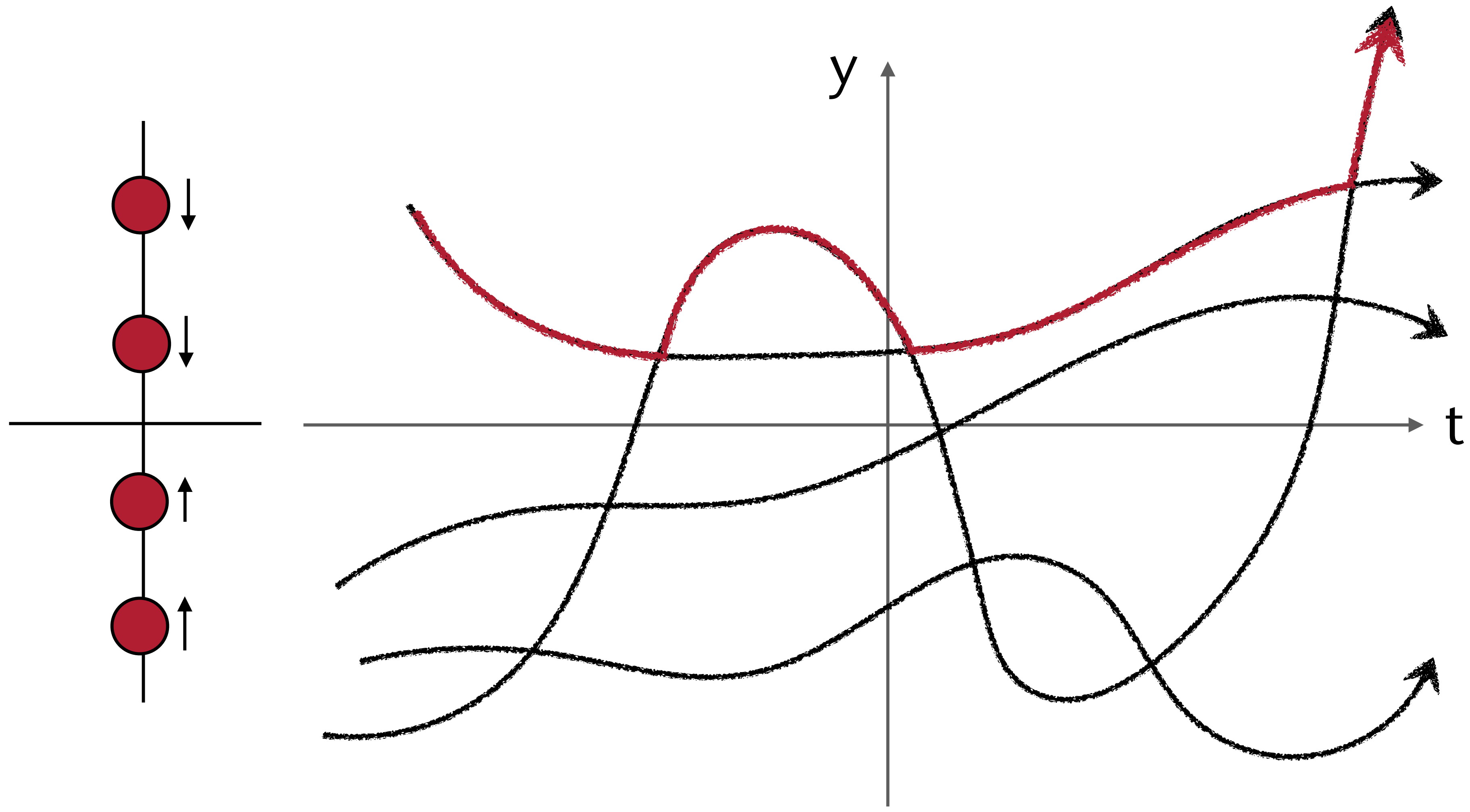
- Maintain the topmost among points moving along the y-axis
    - Is a building block for the kinetic bbox



- Look at the *ty*-plane (flight paths)

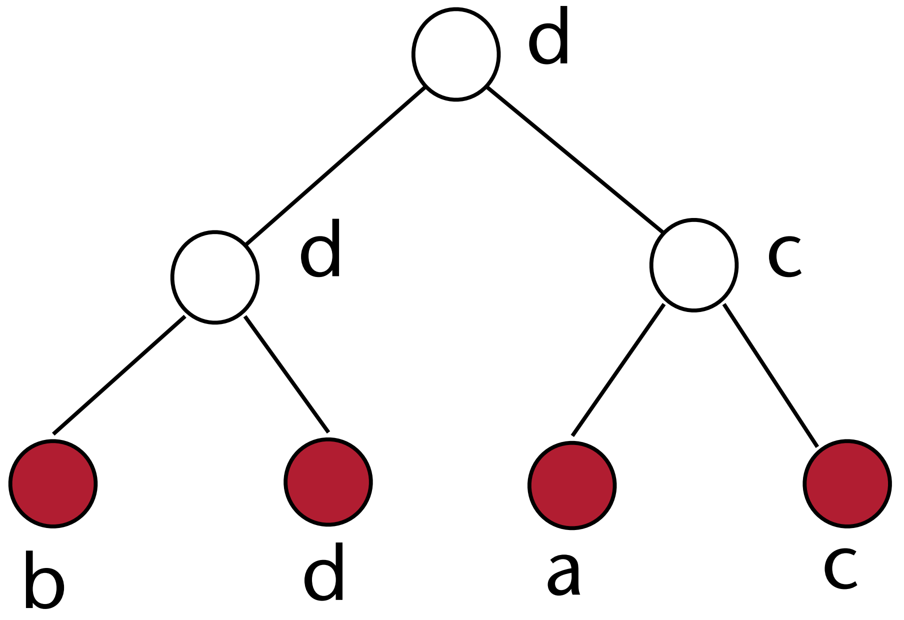


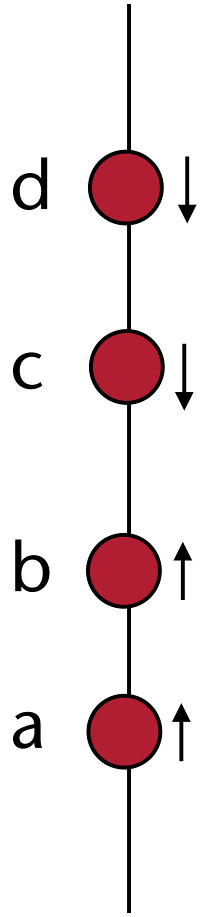
- We are interested in the *upper envelope:*



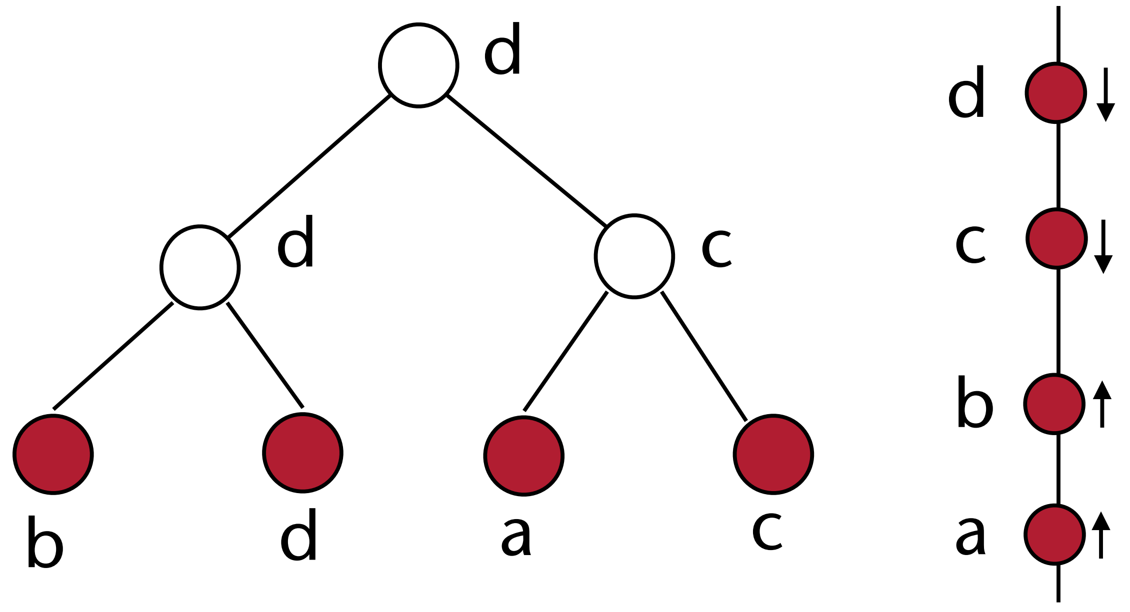
- Theorem (w/o proof) [Sharir, Hart, Agarwal and others]:
  Given n flight paths. If any pair of flight paths intersects at most s times, then the complexity of computing the upper envelope is in $O(n \log n)$

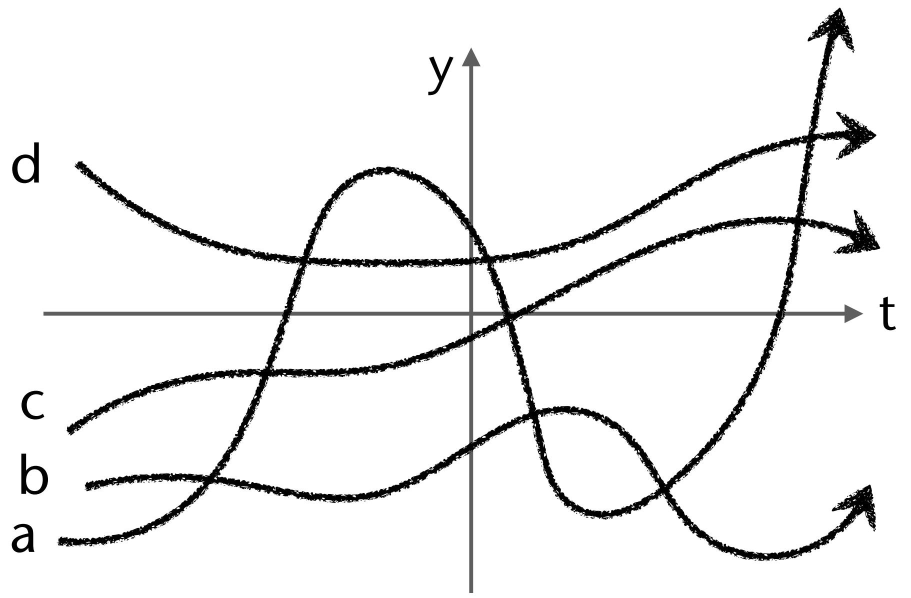
- Problem: change of flight path $\longrightarrow$ recomputation of the envelope
  - Takes O(n log n) time
  - Can we update the envelope / topmost point faster?
- Solution: the tournament tree = kinetic heap
  - Leaves = points
  - Inner nodes = topmost of its two children
  - Event queue = p-queue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





- For all inner nodes, maintain certificate: "left child point is above right child point"
- Event = left/right points swap order along y axis
- Processing an event:
  - Replace pt stored in node with the "winner" and delete/add two events in the event queue
    - Which ones?
  - Potentially propagate new point up through tree
  - Takes $O(\log^2 n)$ time $\longrightarrow$ responsive
- # certificates = # inner nodes = $O(n)$ $\longrightarrow$ compact
- Each point participates in $O(\log n)$ events $\longrightarrow$ local





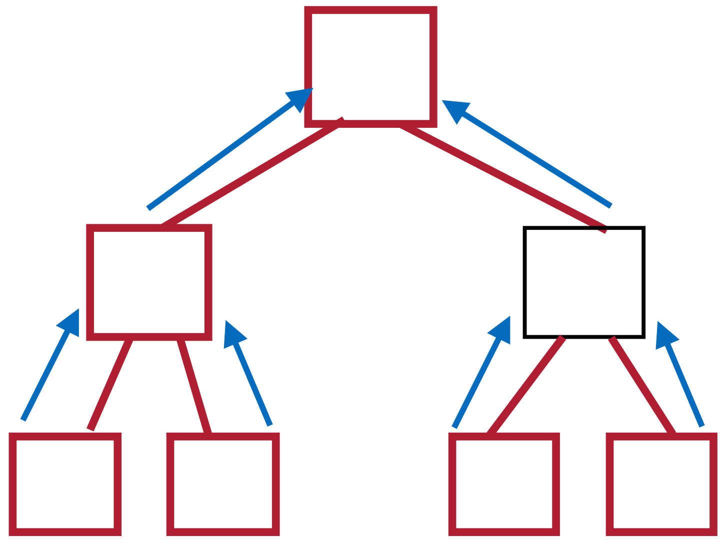
- A problem with deformable objects: BVH becomes invalid

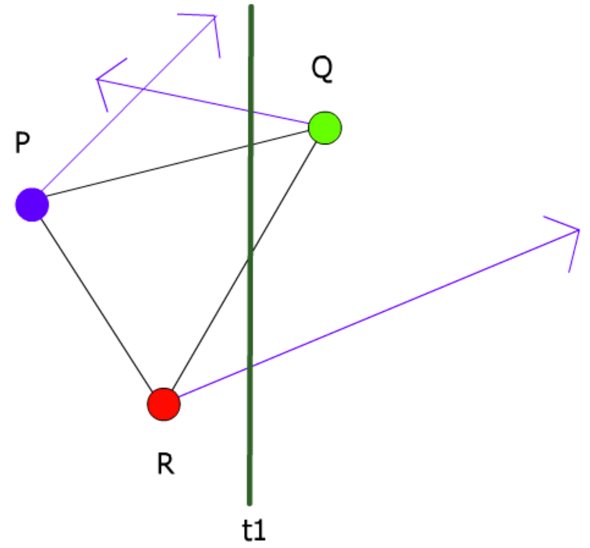
Classic BVH update:

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

Kinetic BVH update:

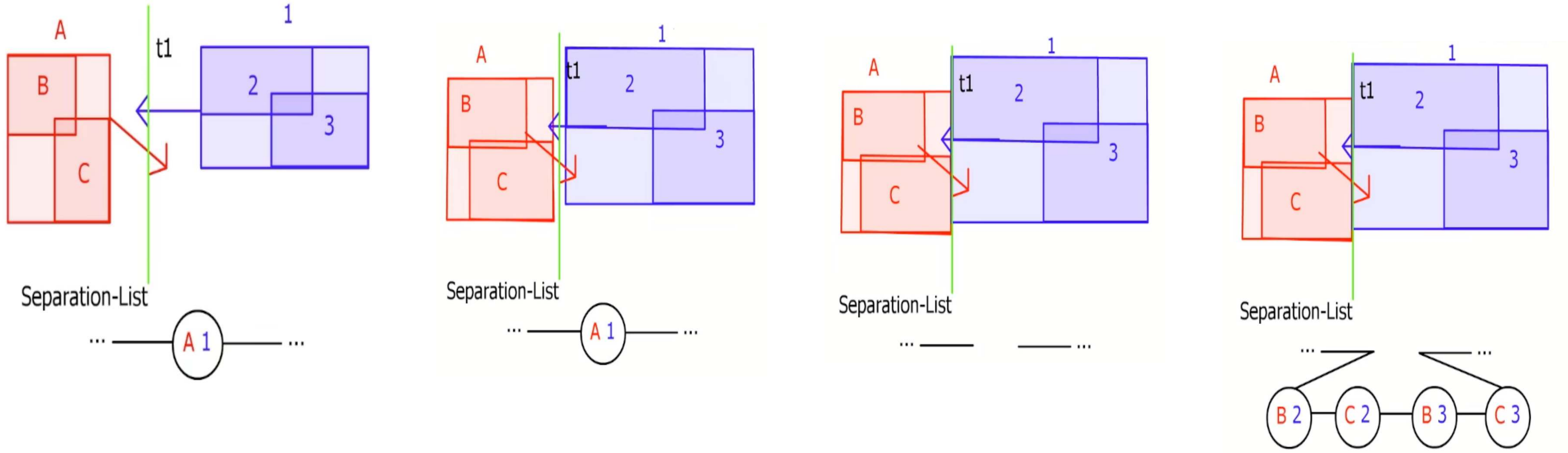
- Event-based (do work only, if something essential changed)
- $O(n \log n)$ → independent of query/ sim. frequency!



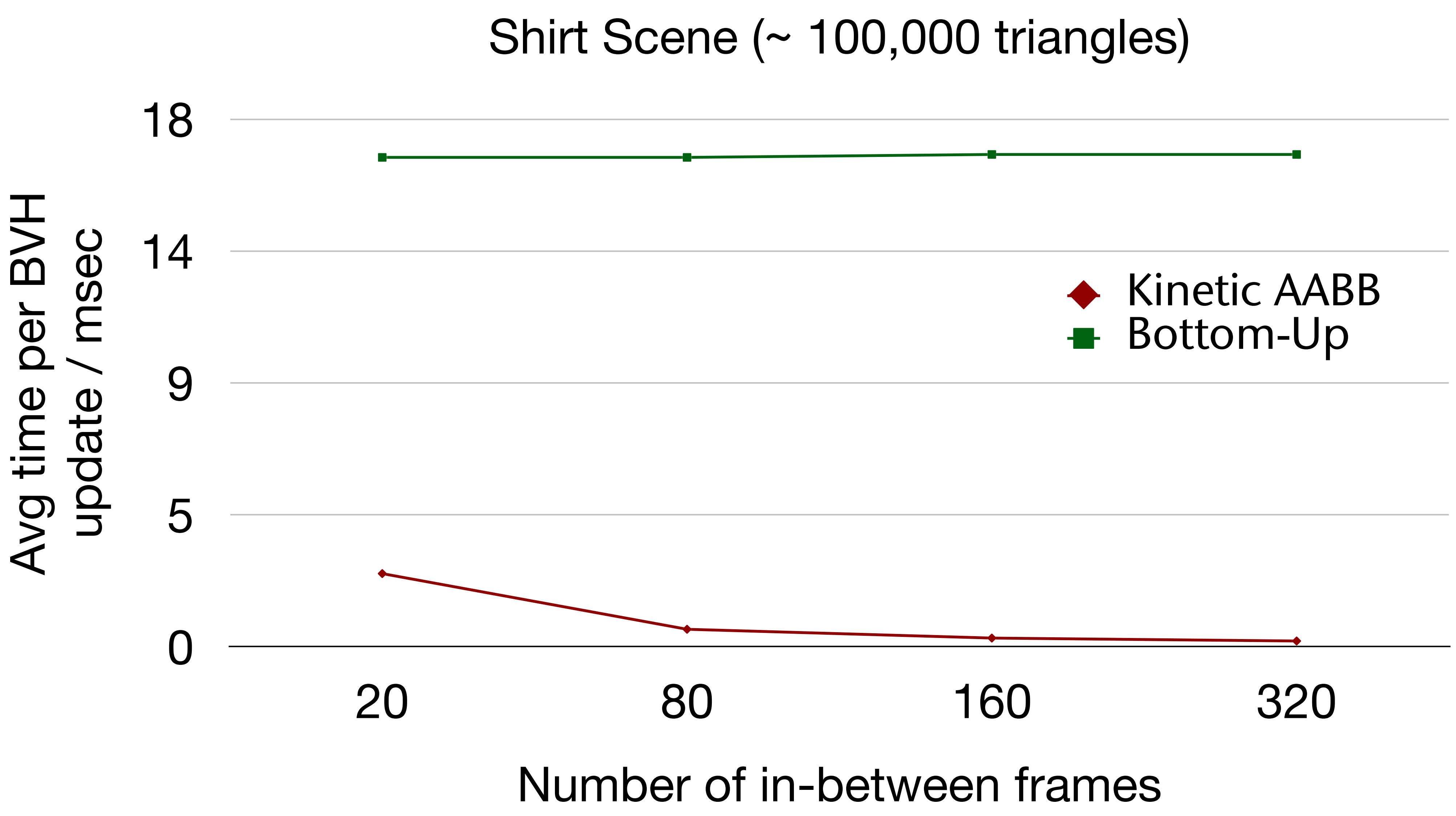


Extension: the Kinetic Separation List

- Definition: A separation list stores *pairs of BVs* in two BVHs, resp., which are non-overlapping and which have parents that do overlap (i.e., those pairs of BVs where the simultaneous traversal of the BVHs during collision detection stopped)



Results



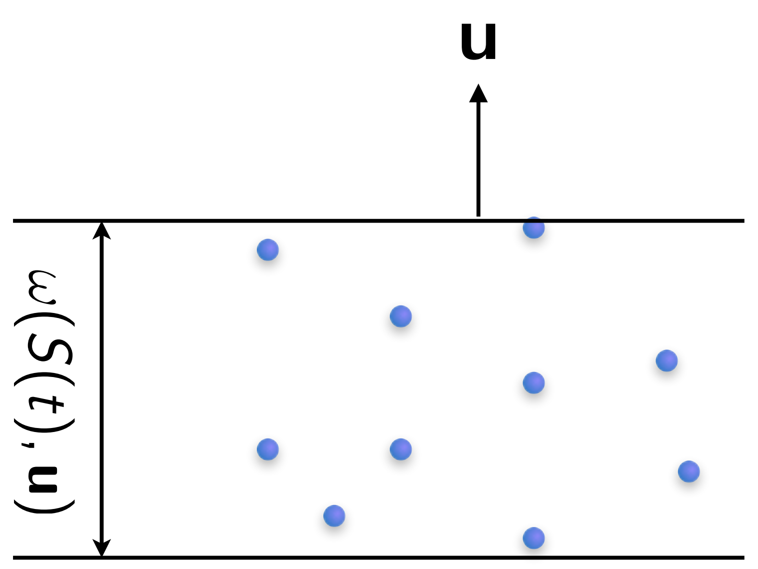
Shirt Scene (~ 100,000 triangles)
Avg time per BVH update / msec
18
14
9
5
0
20
80
160
320
Number of in-between frames
Kinetic AABB
Bottom-Up

Problems of KDS

- Number of events can kill performance
- Computing event times is expensive
- KDS as a whole can become very complex, housekeeping becomes too expensive and bug-prone (e.g., kinetic BSP in 3D)
- KDS needs to be updated throughout time, even if we don't need it for a long duration in-between queries

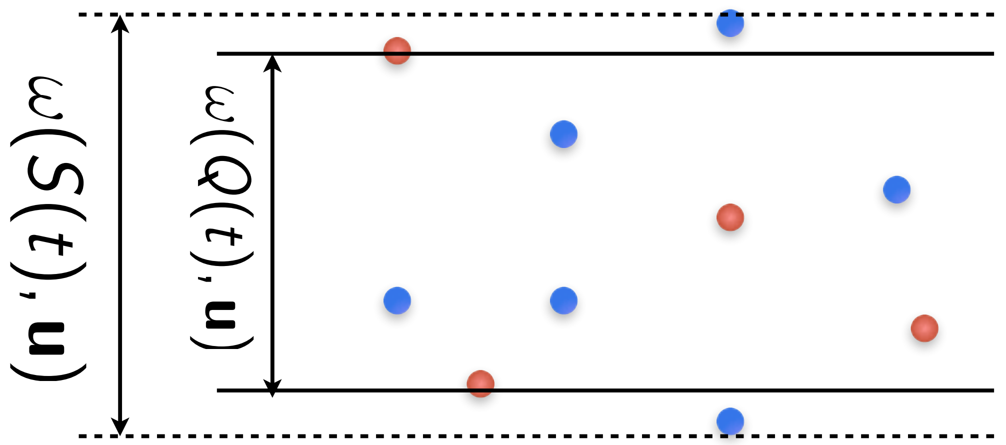
Sketch of a Possible Approach by Way of an Example

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



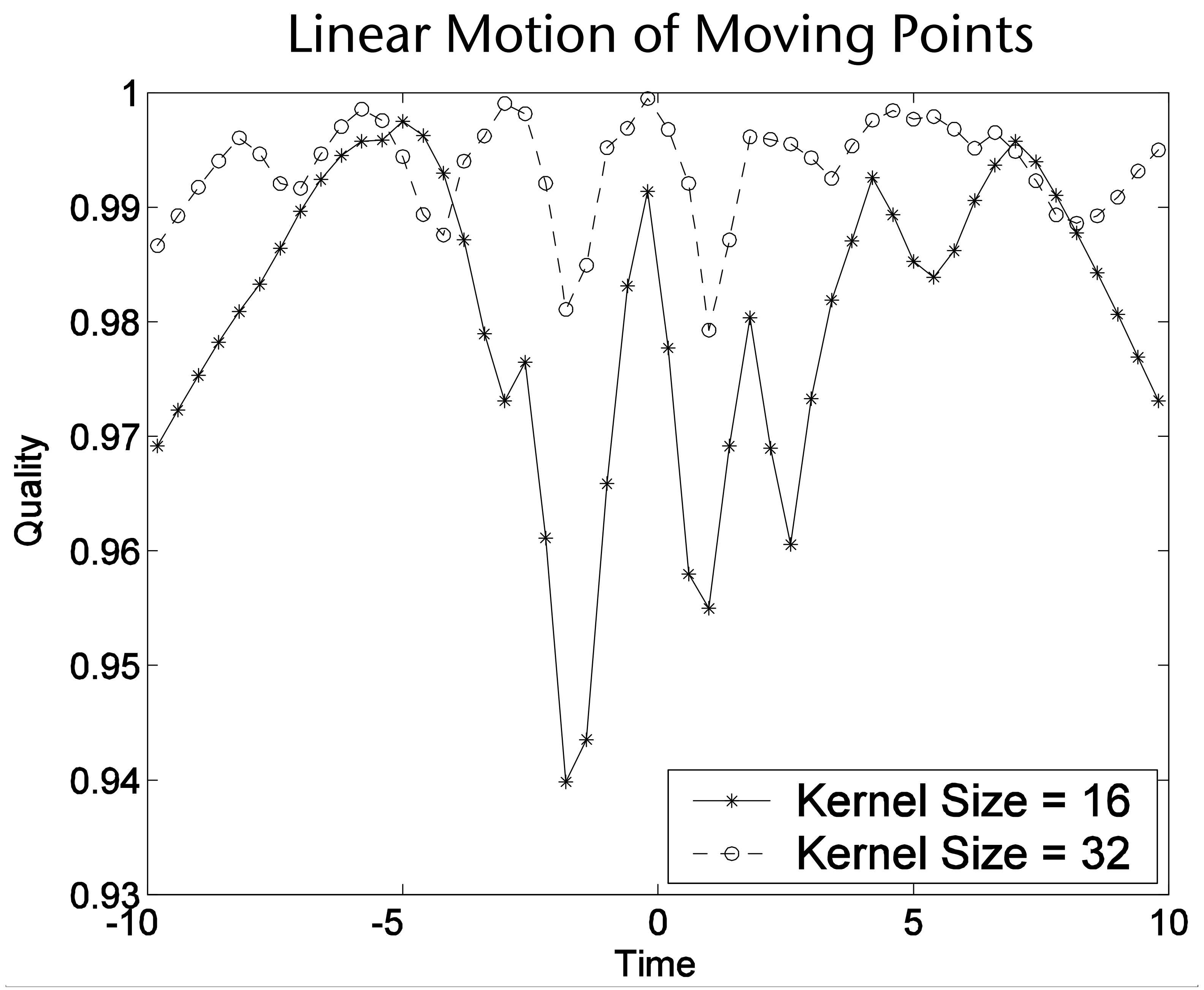
- Definition: ε-kernel
  Let $Q \subseteq S$. Q is called an ε-kernel of S iff

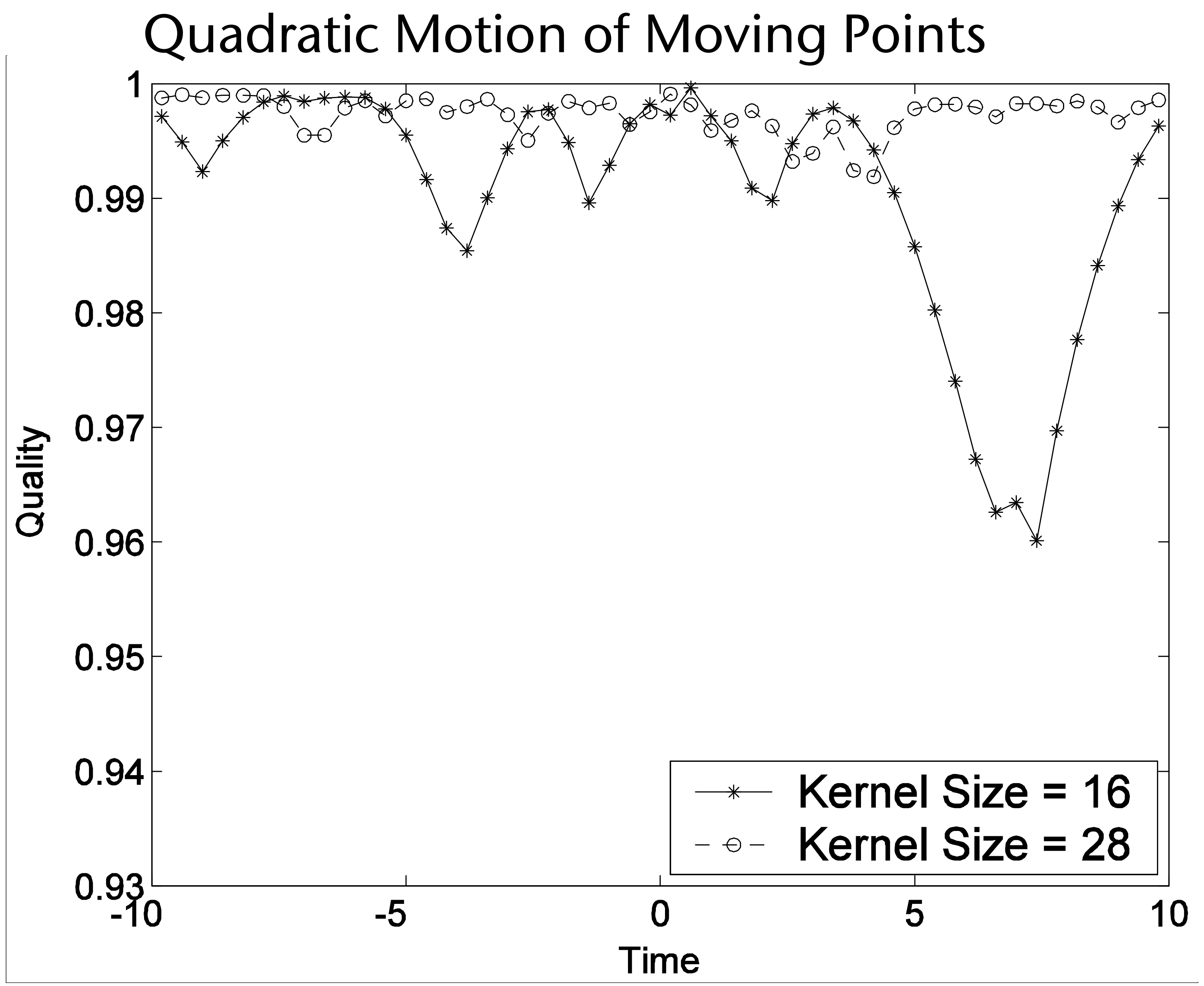
$$\forall t : \omega(S(t), \mathbf{u}) \leq (1+\varepsilon)\omega(Q(t), \mathbf{u})$$



- Theorem (w/o proof) [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 $O\left(n + \frac{1}{\varepsilon^3}\right)$.

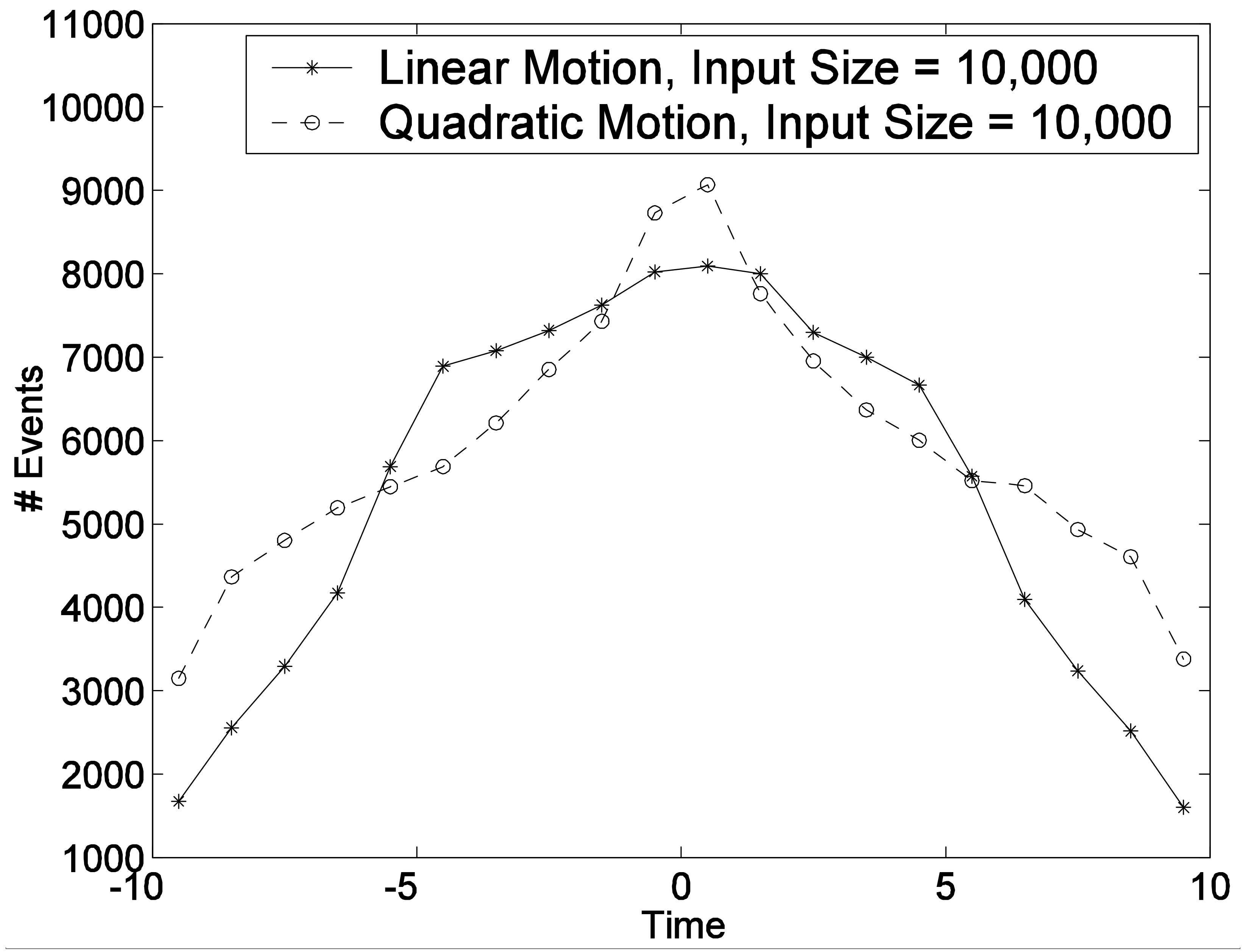
Results for BBox Maintained by Eps-Approximate KDS



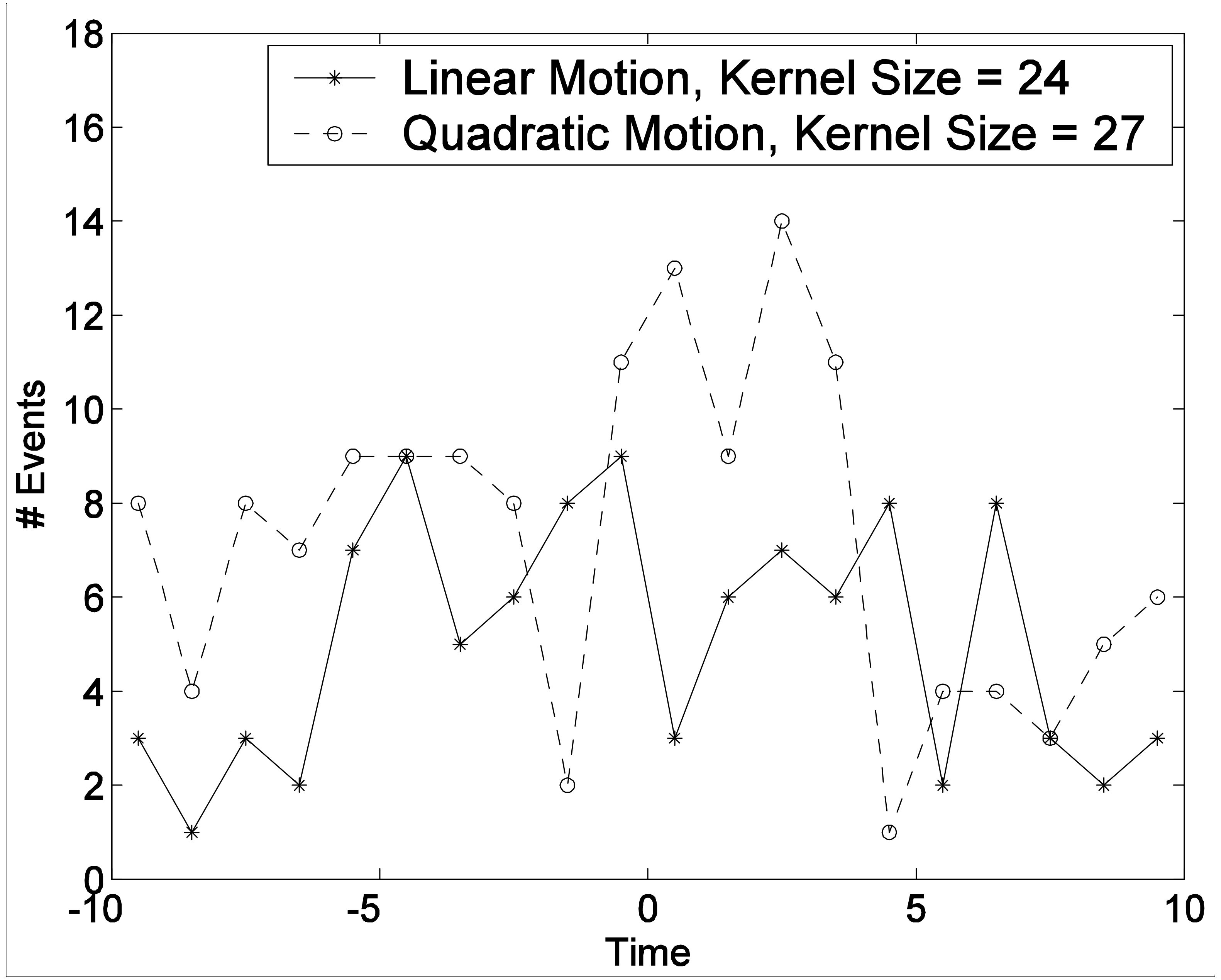


10,000 moving points
Error < 0.02 for kernel of size 32

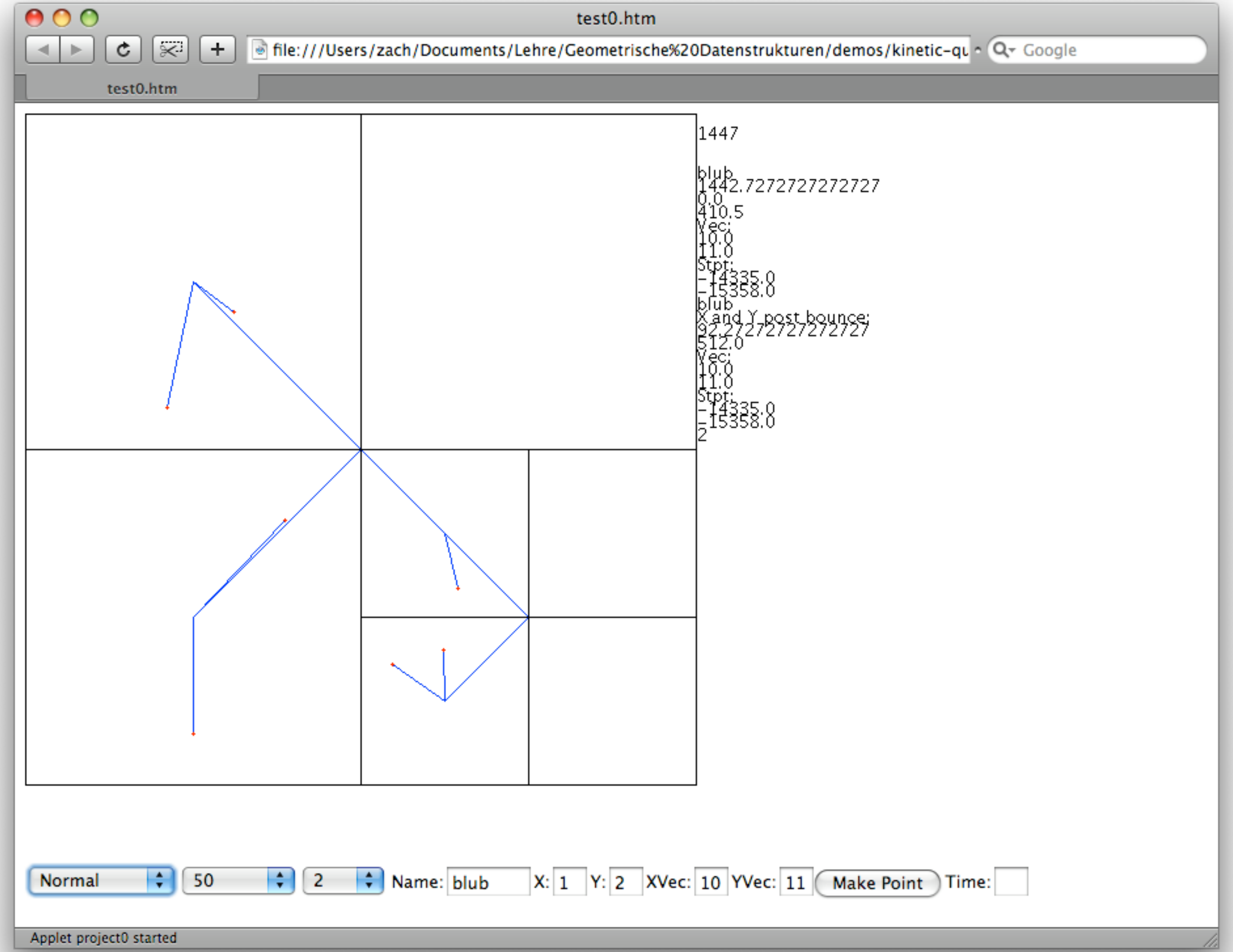
Exact Algorithm



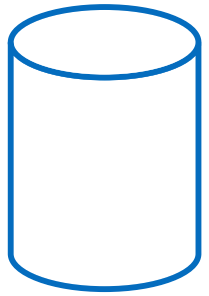
Approximation Algorithm



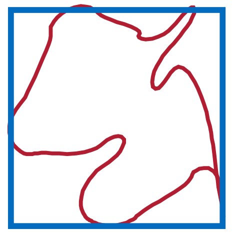
Kinetic Quadtree Demo



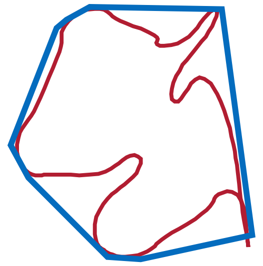
Different Types of Bounding Volumes



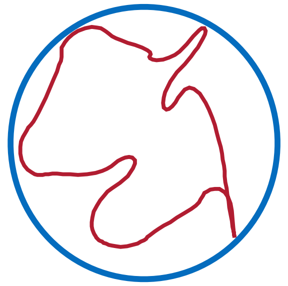
Cylinder
[Weghorst et al., 1985]



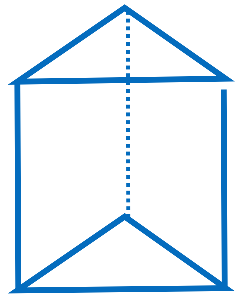
Box, AABB (R*-trees)
[Beckmann, Kriegel, et al., 1990]



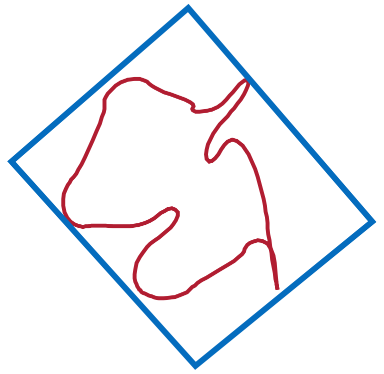
Convex hull
[Lin et. al., 2001]



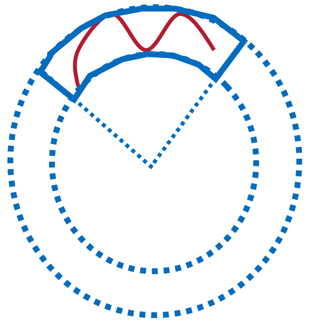
Sphere
[Hubbard, 1996]



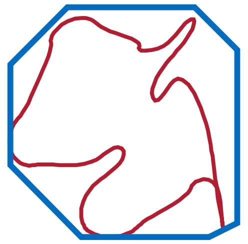
Prism
[Barequet, et al., 1996]



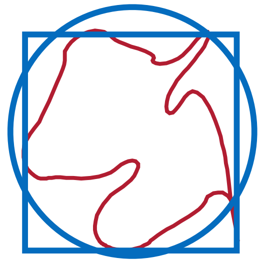
OBB (oriented bounding box)
[Gottschalk, et al., 1996]



Spherical shell
[…]

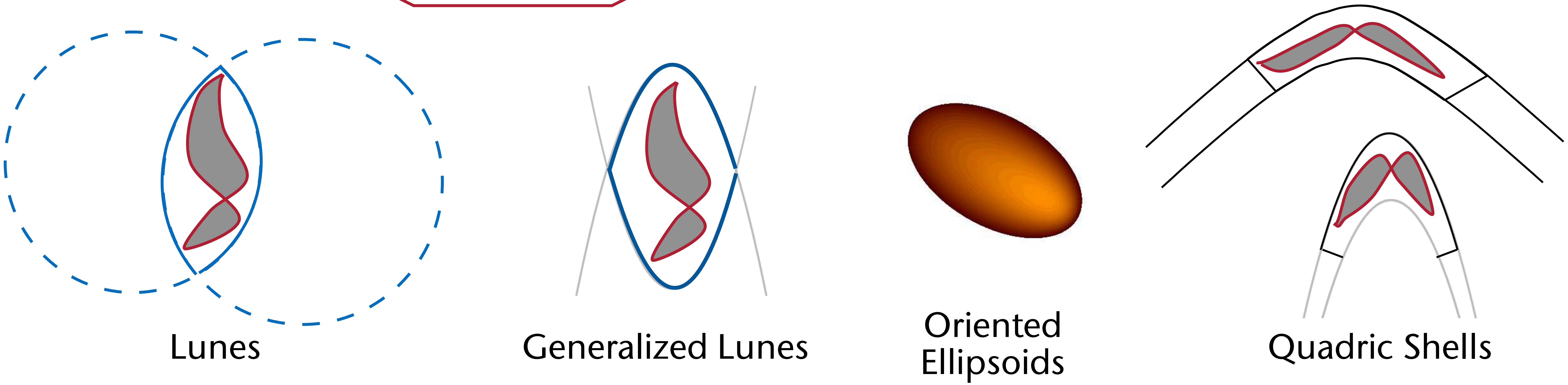


k-DOPs / Slabs
[Zachmann, 1998]



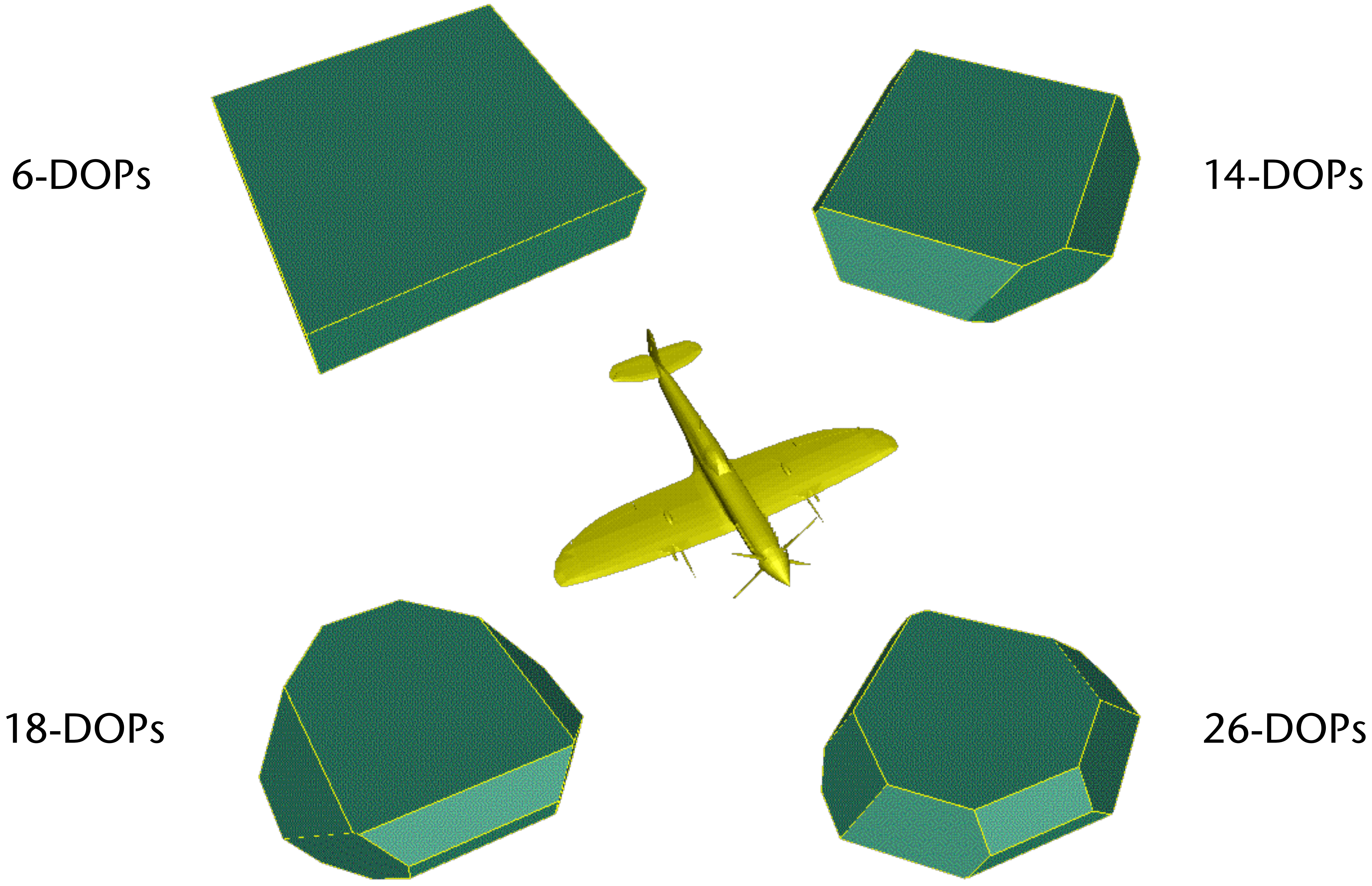
Intersection of
several, other BVs

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

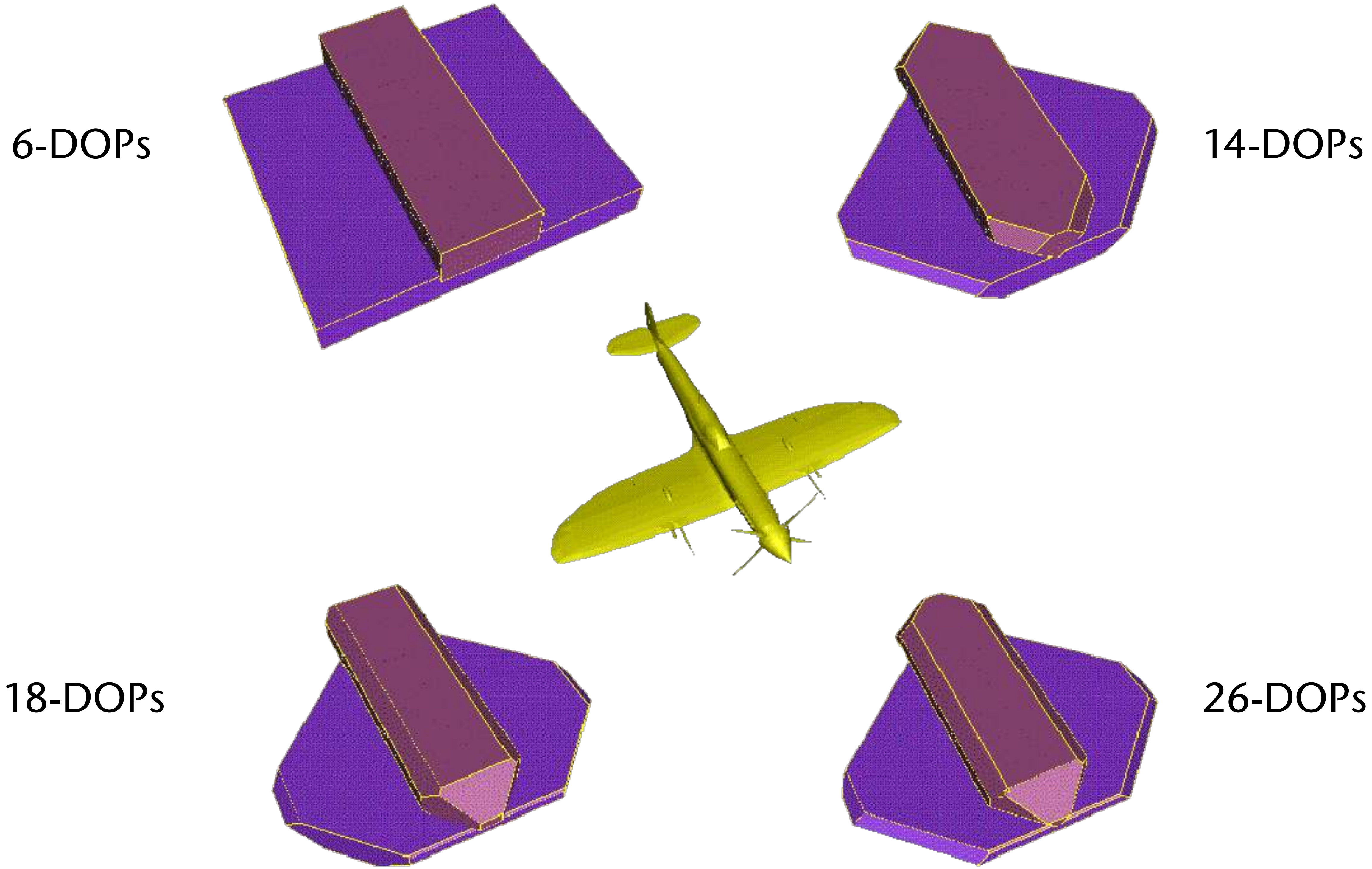


- Research questions:
  - Fast intersection of two BVs for collision detection?
    - Compute is cheap, memory transfer is expensive $\longrightarrow$ BV compression?
    - Exact / approximate (biased) intersection tests?
  - Fast intersection test for rays against such BVs?
  - Efficient BVH construction?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

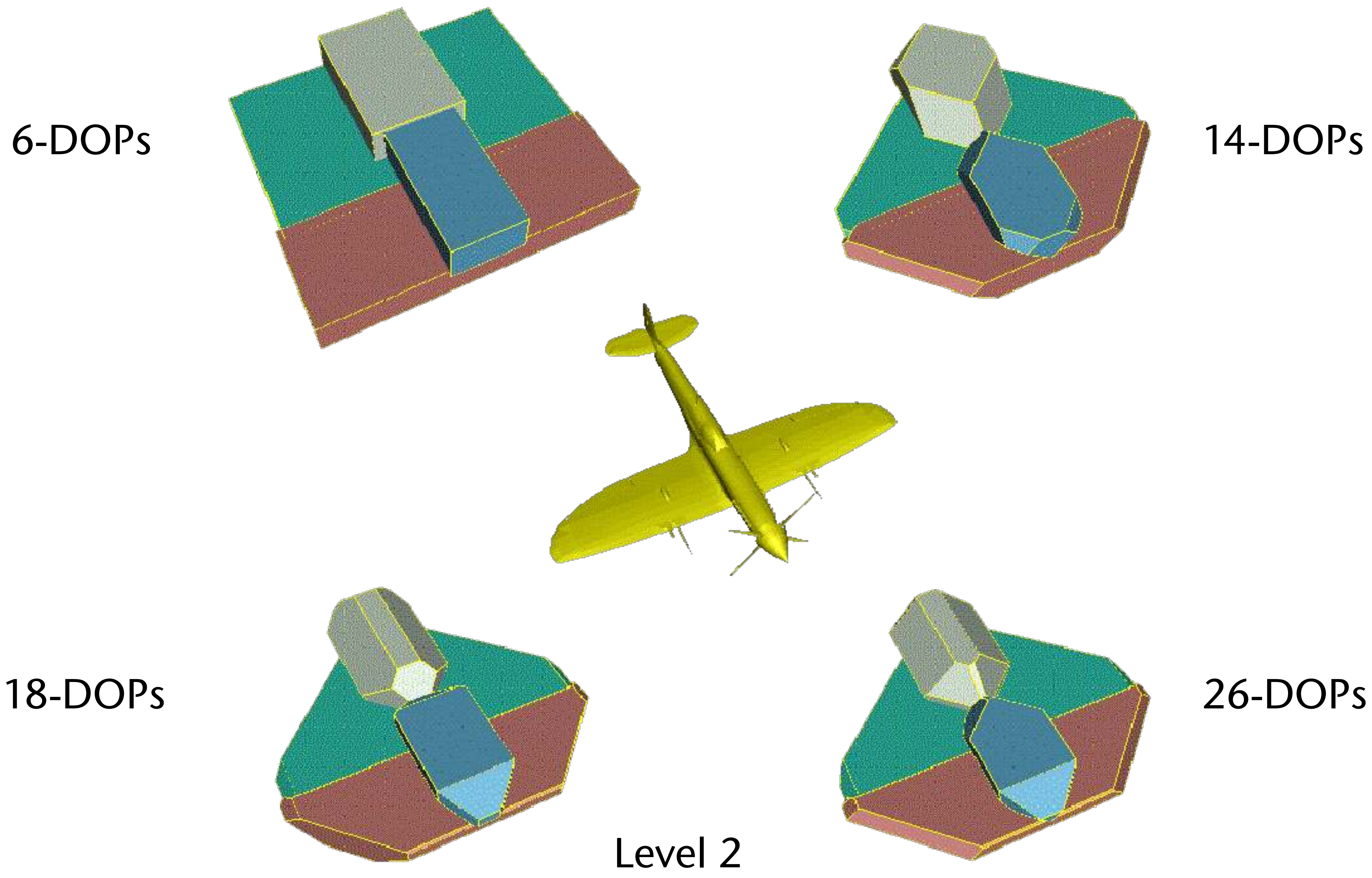
BVH with k-DOPs

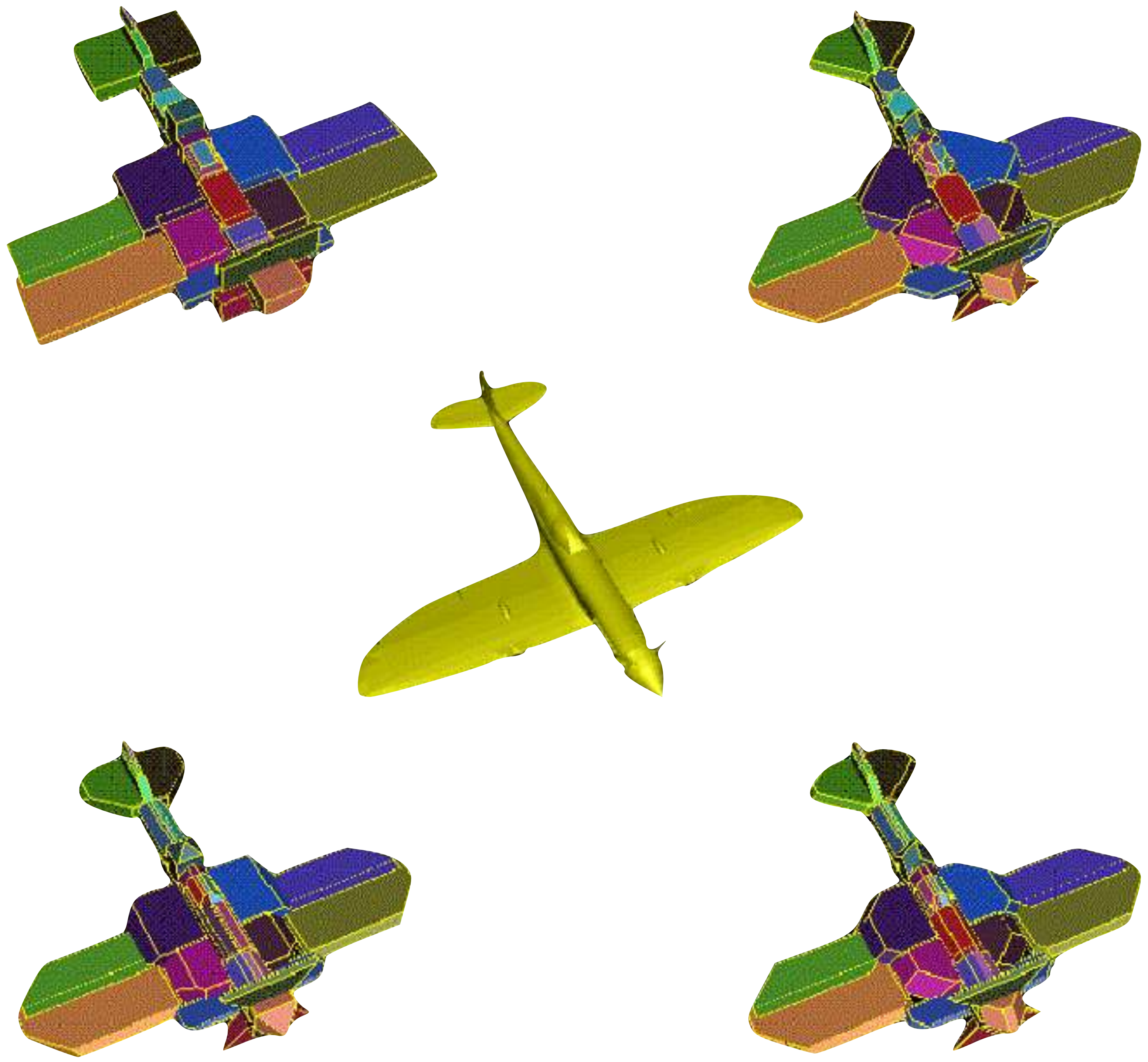


Level 0



Level 1


6-DOPs
14-DOPs
18-DOPs
26-DOPs
Level 2



6-DOPs

14-DOPs

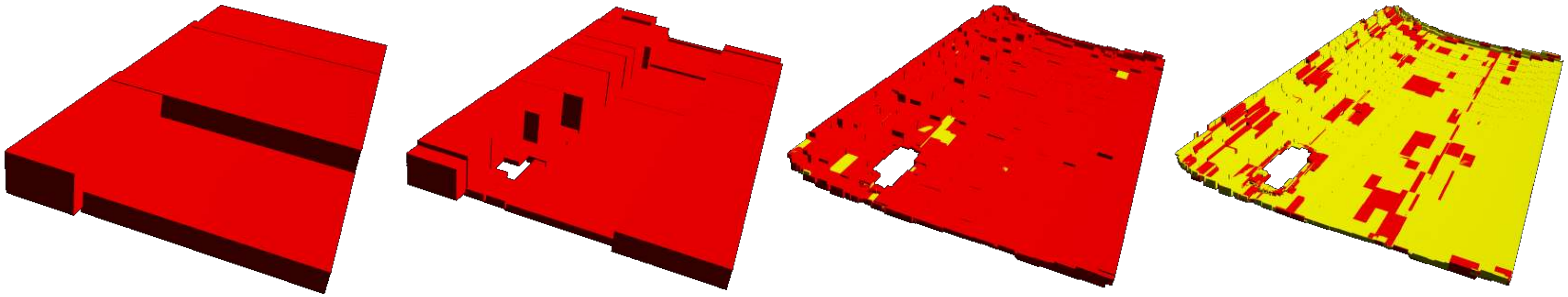
18-DOPs

26-DOPs

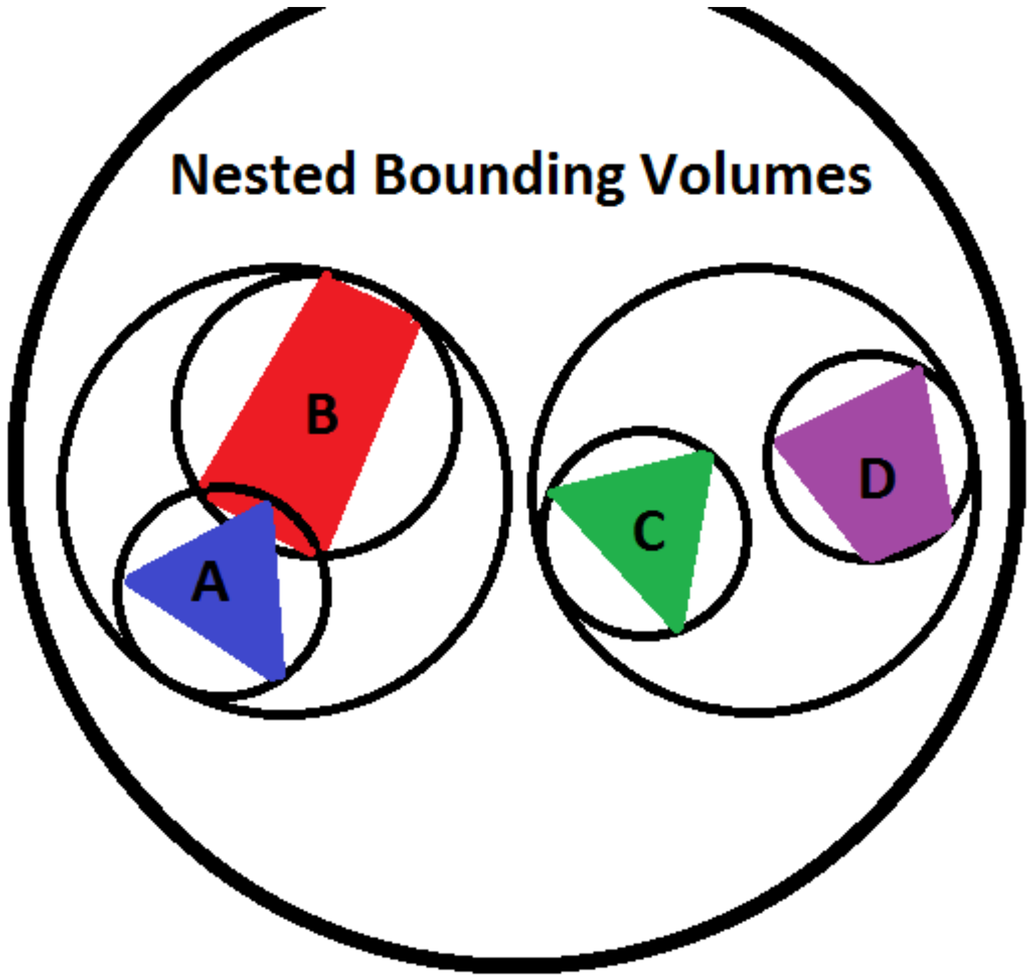
Level 5

Level 8

BVH with AABBs



Wrapped vs Layered BVH



Layered BVH:
a BV must bound its child BVs

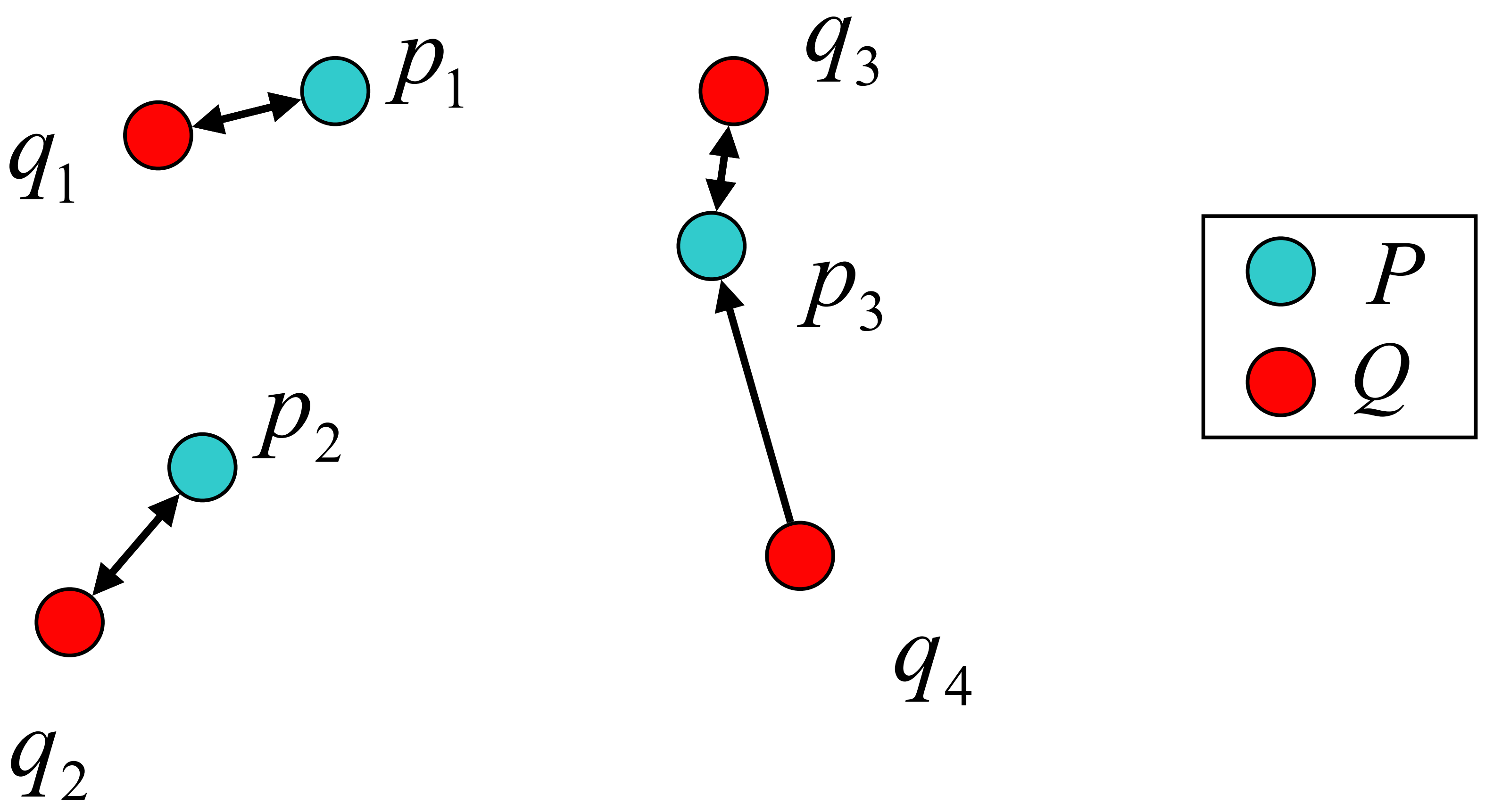
Wrapped BVH:
a BV bounds its associated primitives,
but not necessarily its child BVs

Directed Hausdorff distance

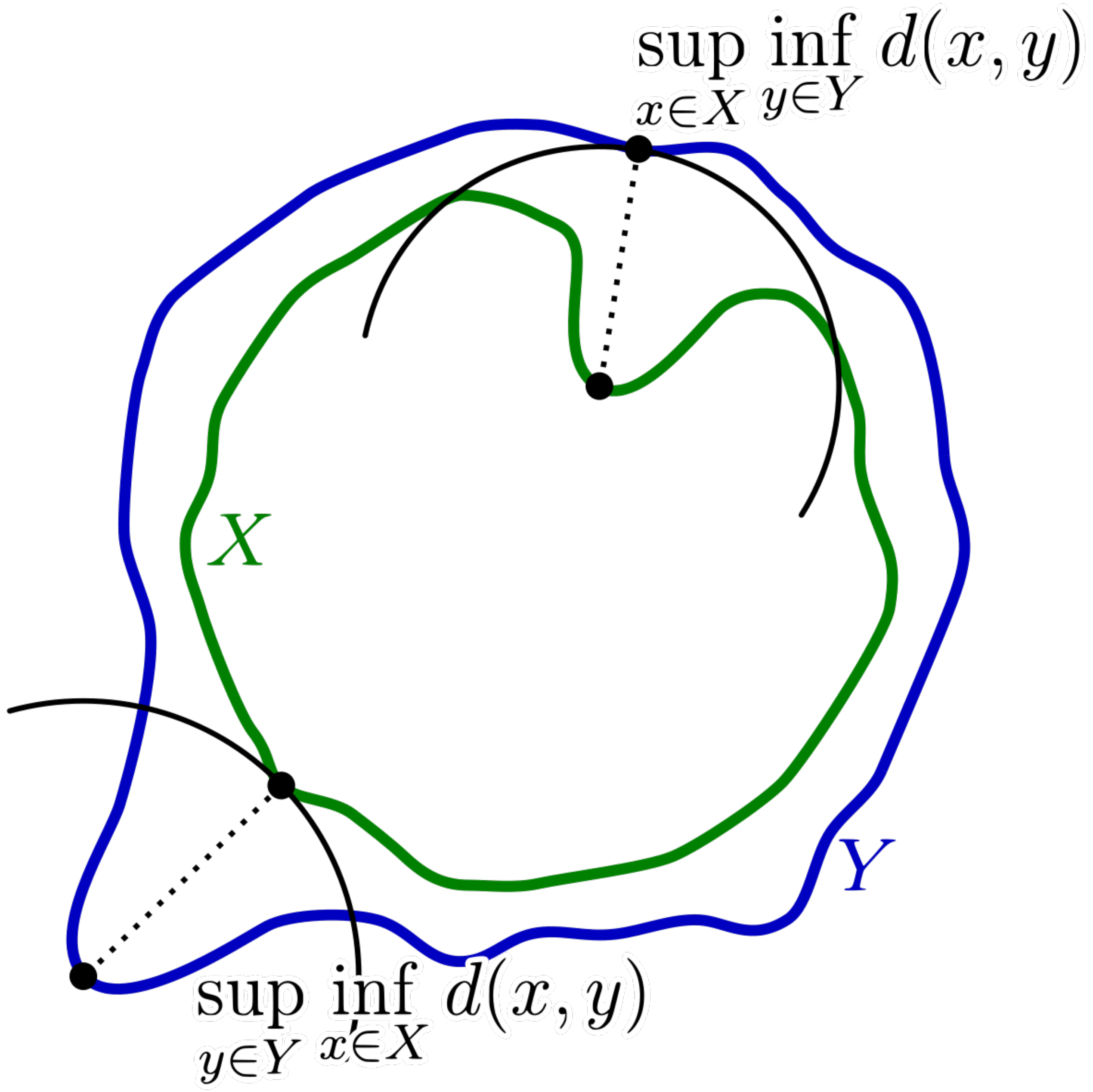
- Def:
  maximum distance of a set, P, to the nearest point in the other set, Q.

$$h(P,Q) = \max_{p \in P} \min_{q \in Q} d(p,q)$$

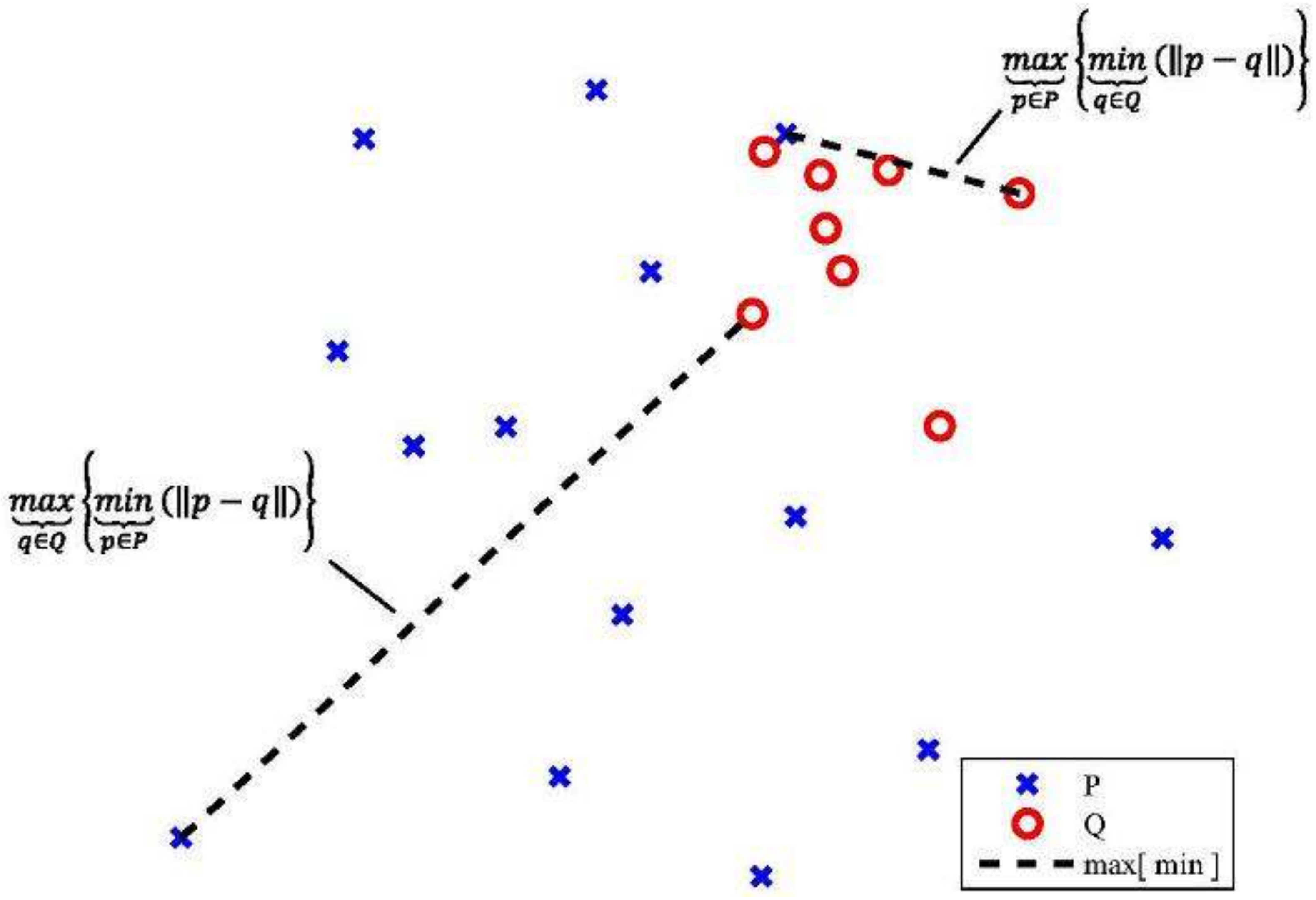
- Example:
  - h(P, Q) = $d(p_2, q_2)$
  - h(Q, P) = $d(p_3, q_4)$
- Property: Not symmetric



Examples



$\sup_{x \in X} \inf_{y \in Y} d(x, y)$
X
Y
$\sup_{y \in Y} \inf_{x \in X} d(x, y)$

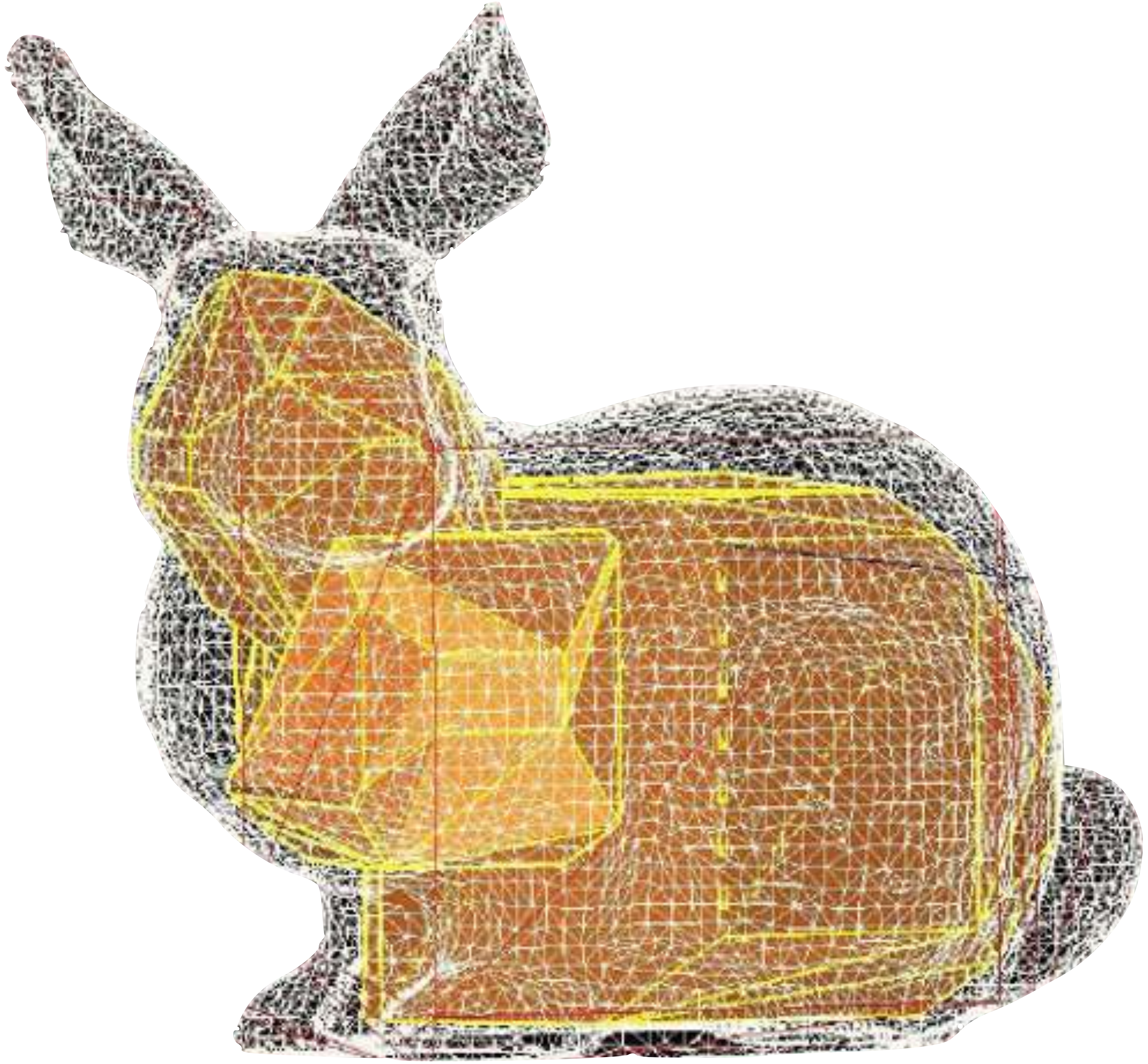


$\underbrace{max}_{p \in P} \left\{ \underbrace{min}_{q \in Q} (\|p - q\|) \right\}$
$\underbrace{max}_{q \in Q} \left\{ \underbrace{min}_{p \in P} (\|p - q\|) \right\}$
P
Q
max[min]

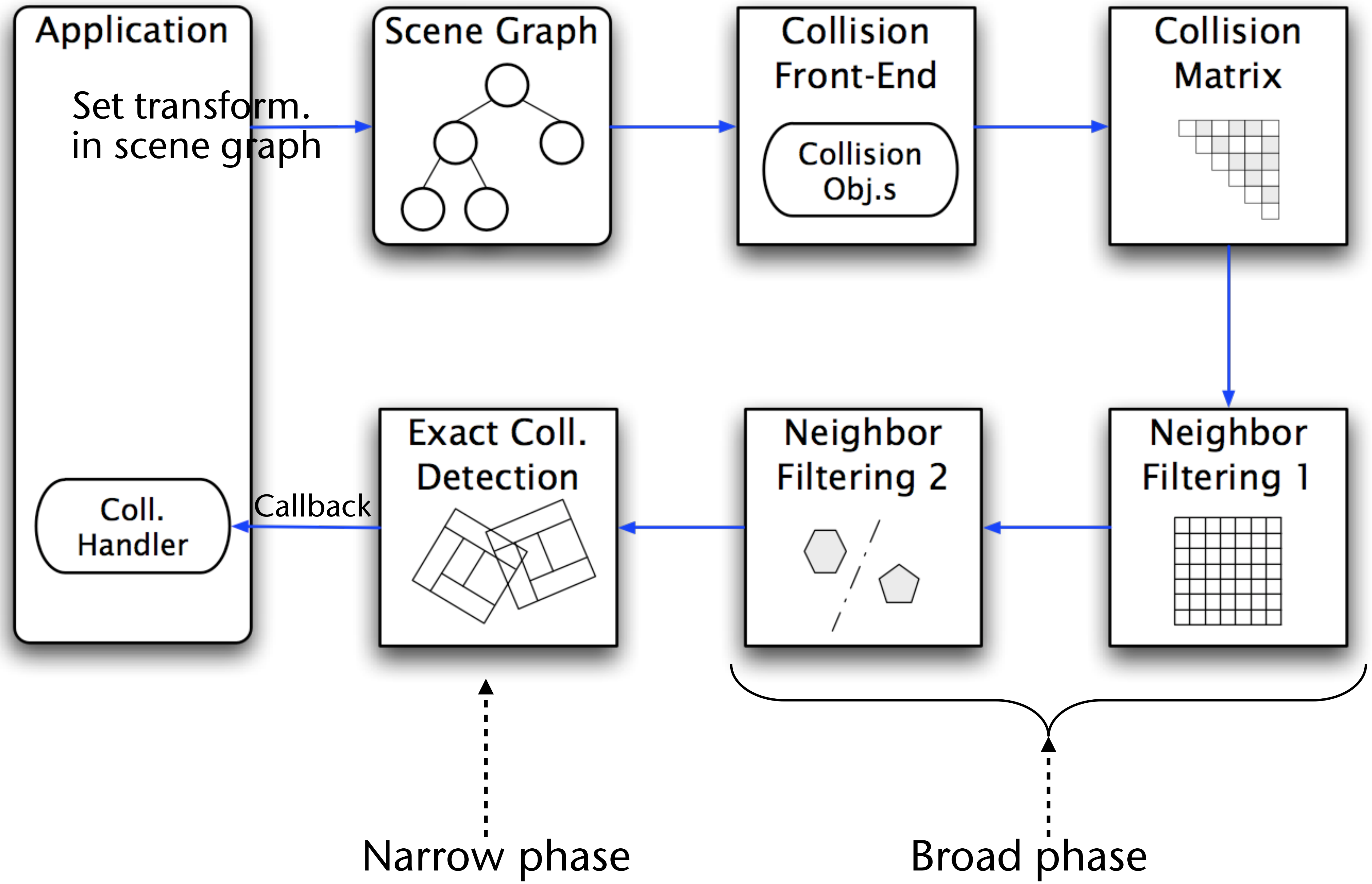
Bidirectional Hausdorff Distance

$$H(P,Q) = \max\{\, h(P,Q)\,,\; h(Q,P)\,\}$$

Digression: Bounding Volumes can Also be Used as *Inner* BVs



The Collision Detection Pipeline



Hierarchical Collision Detection using BVHs

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

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

  return

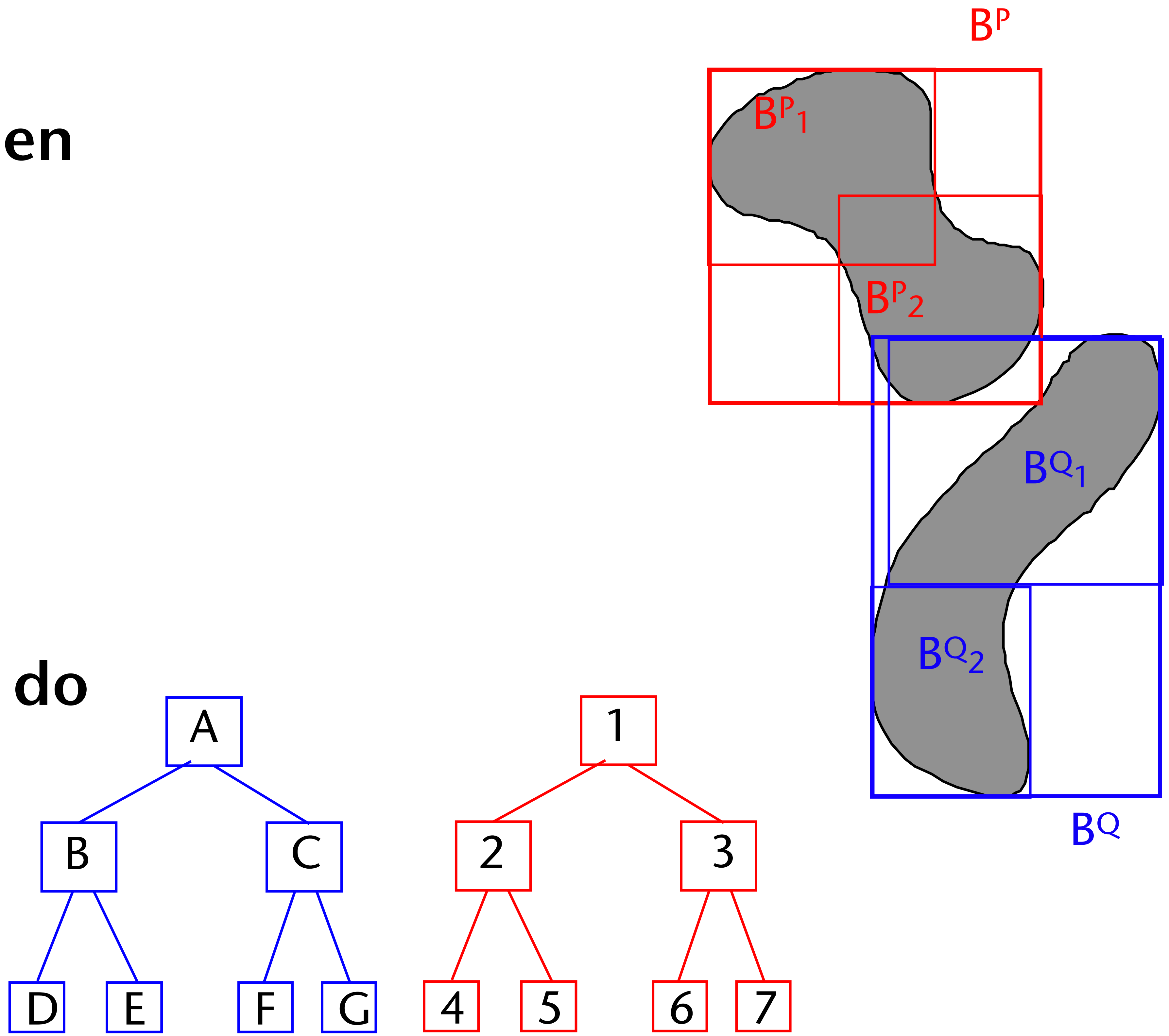
if X,Y are leaves **then**

  check polygons

else

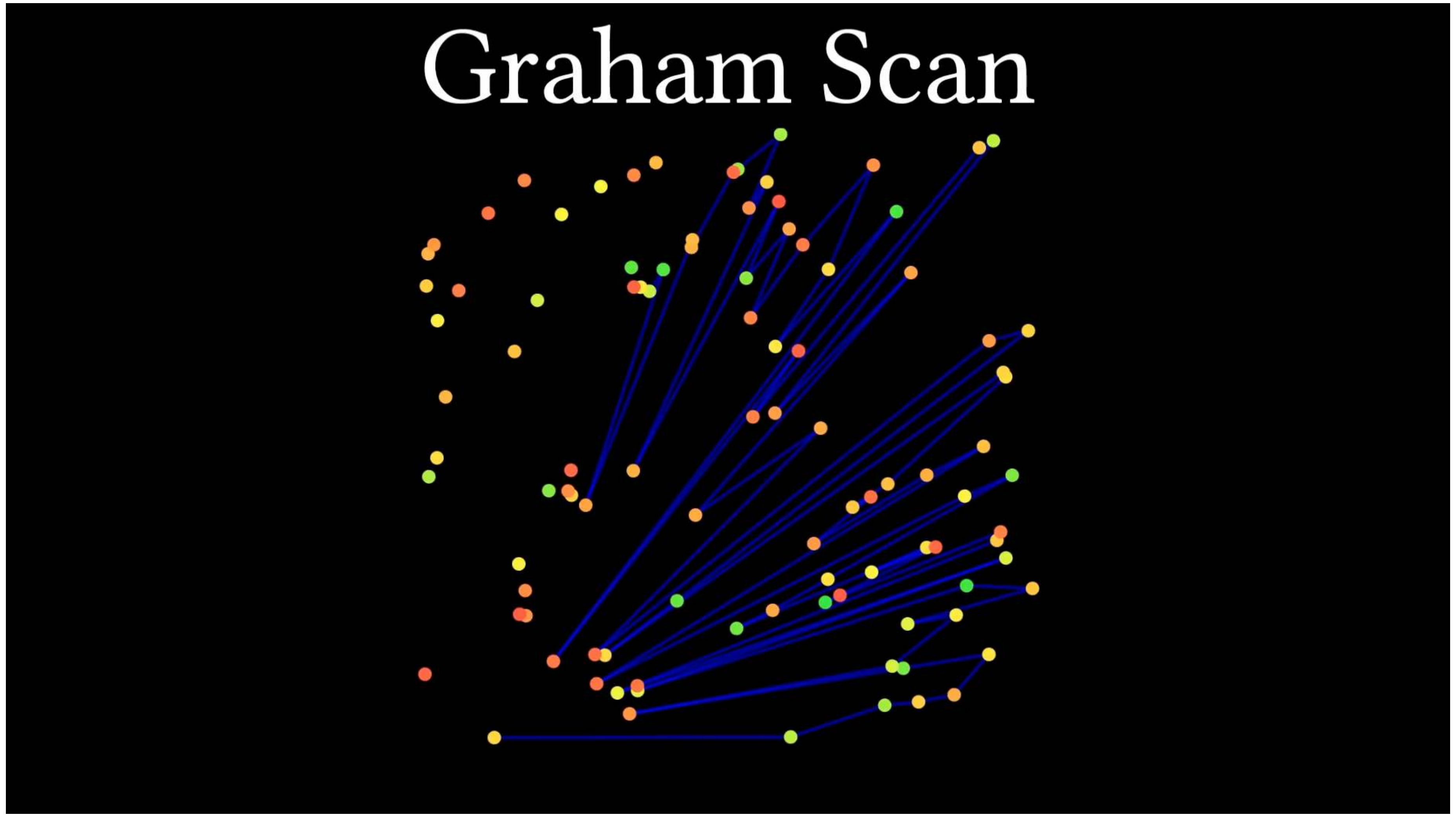
  for all children pairs **do**

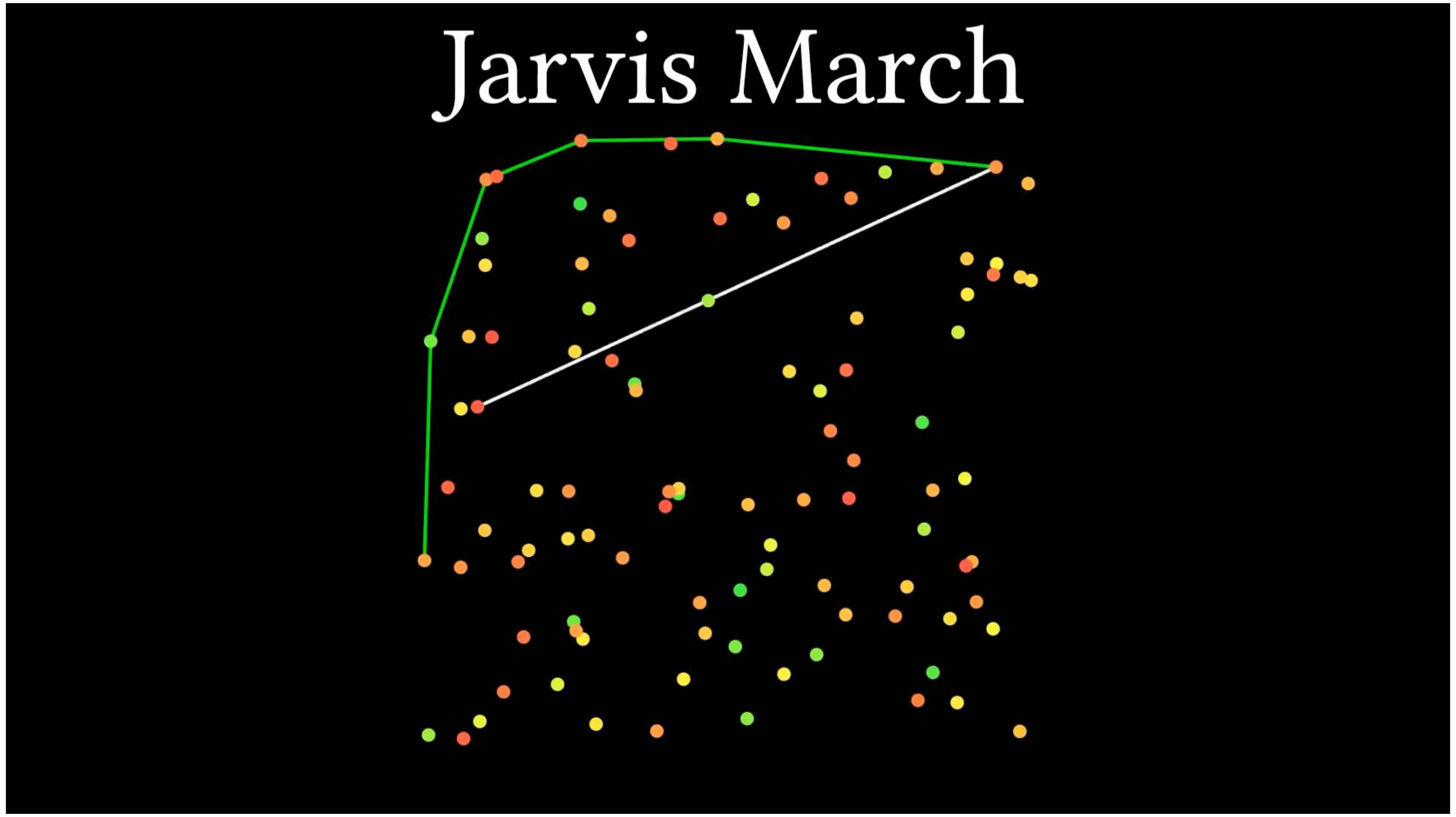
    traverse(Xi, Yj)



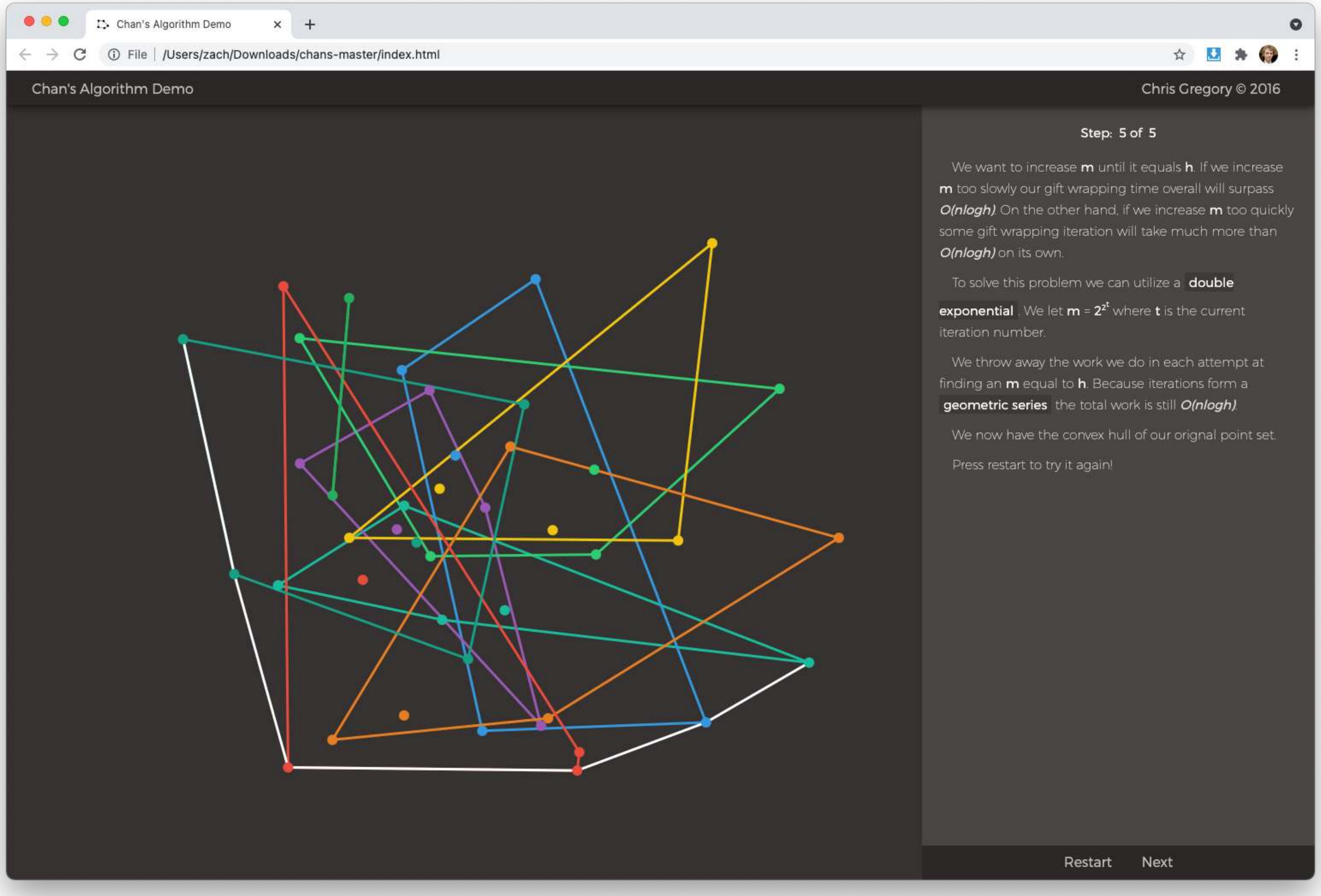
Applications using Distance Fields

Demos of Convex Hull Algorithms in 2D





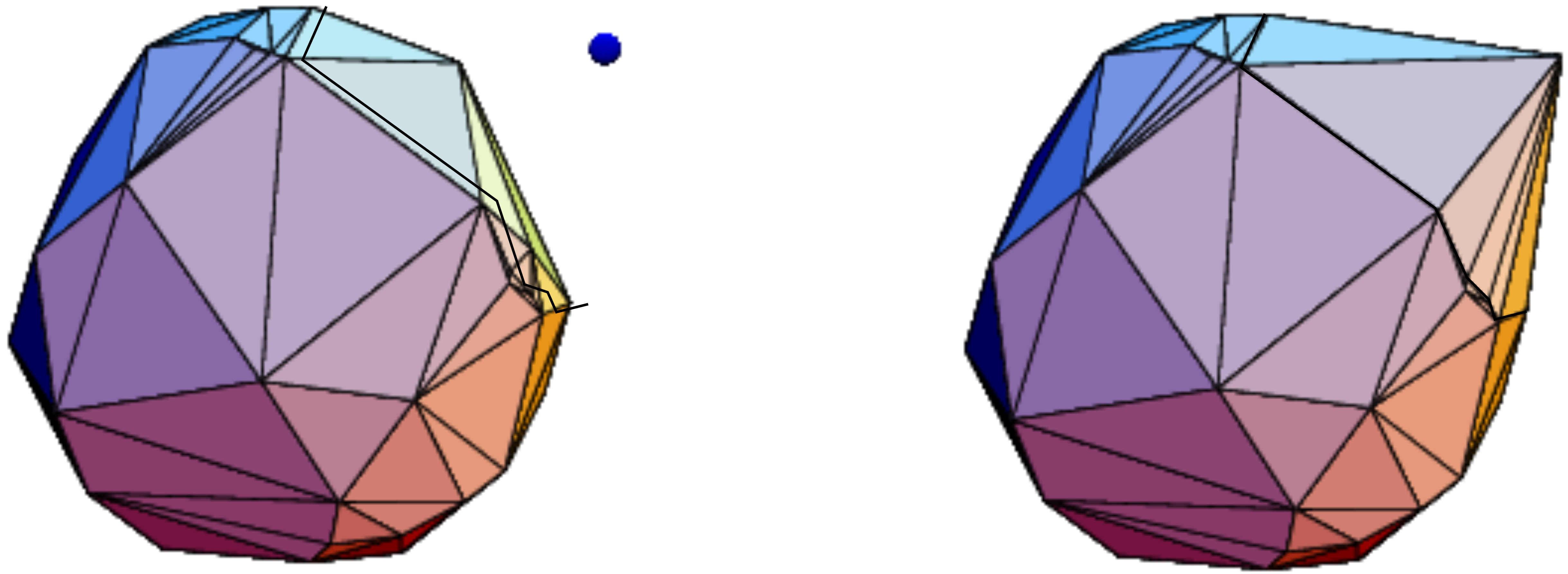
Demos of Convex Hull Algorithms in 2D

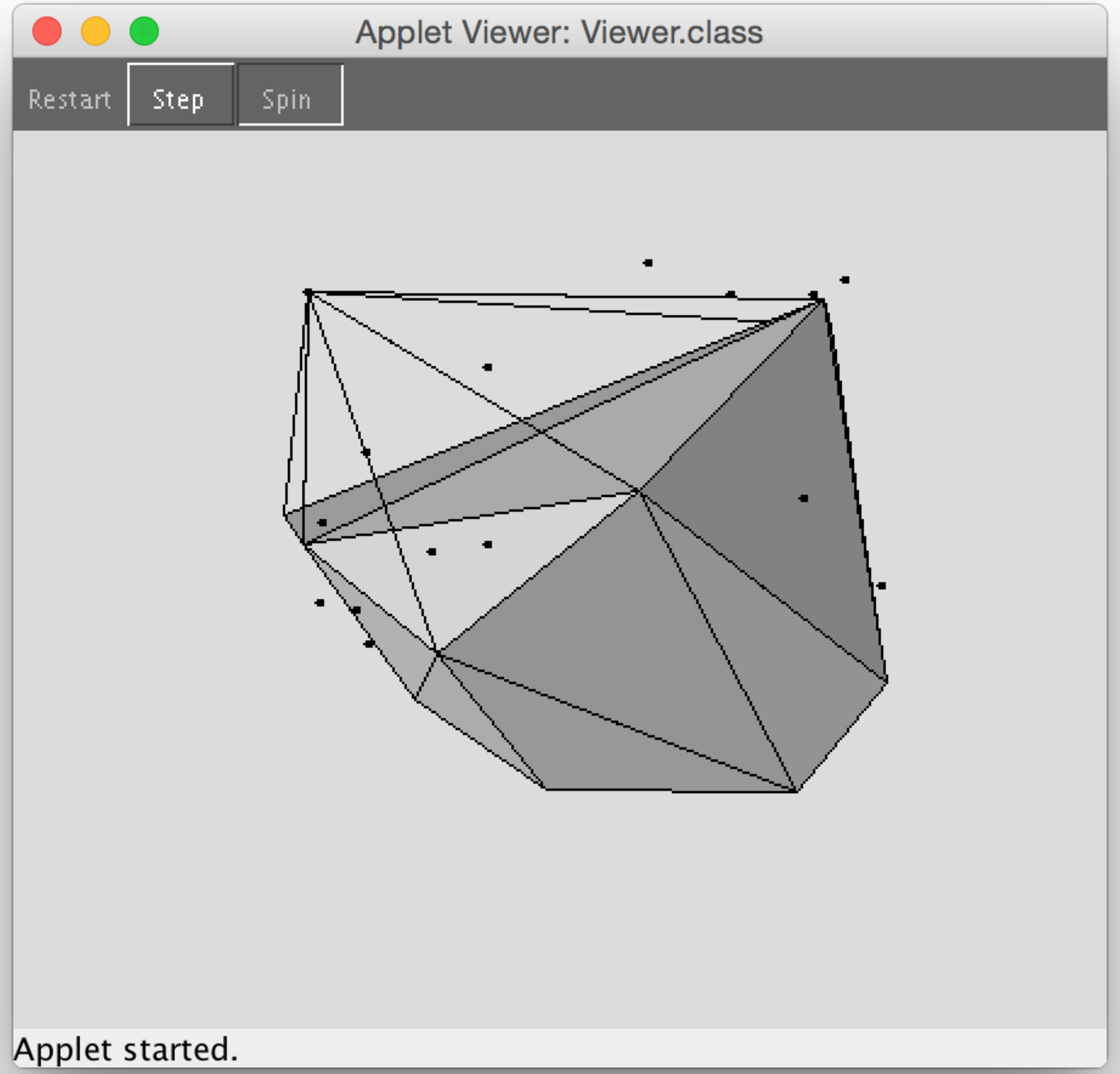


https://github.com/gregorybchris/chans

Convex Hull in 3D

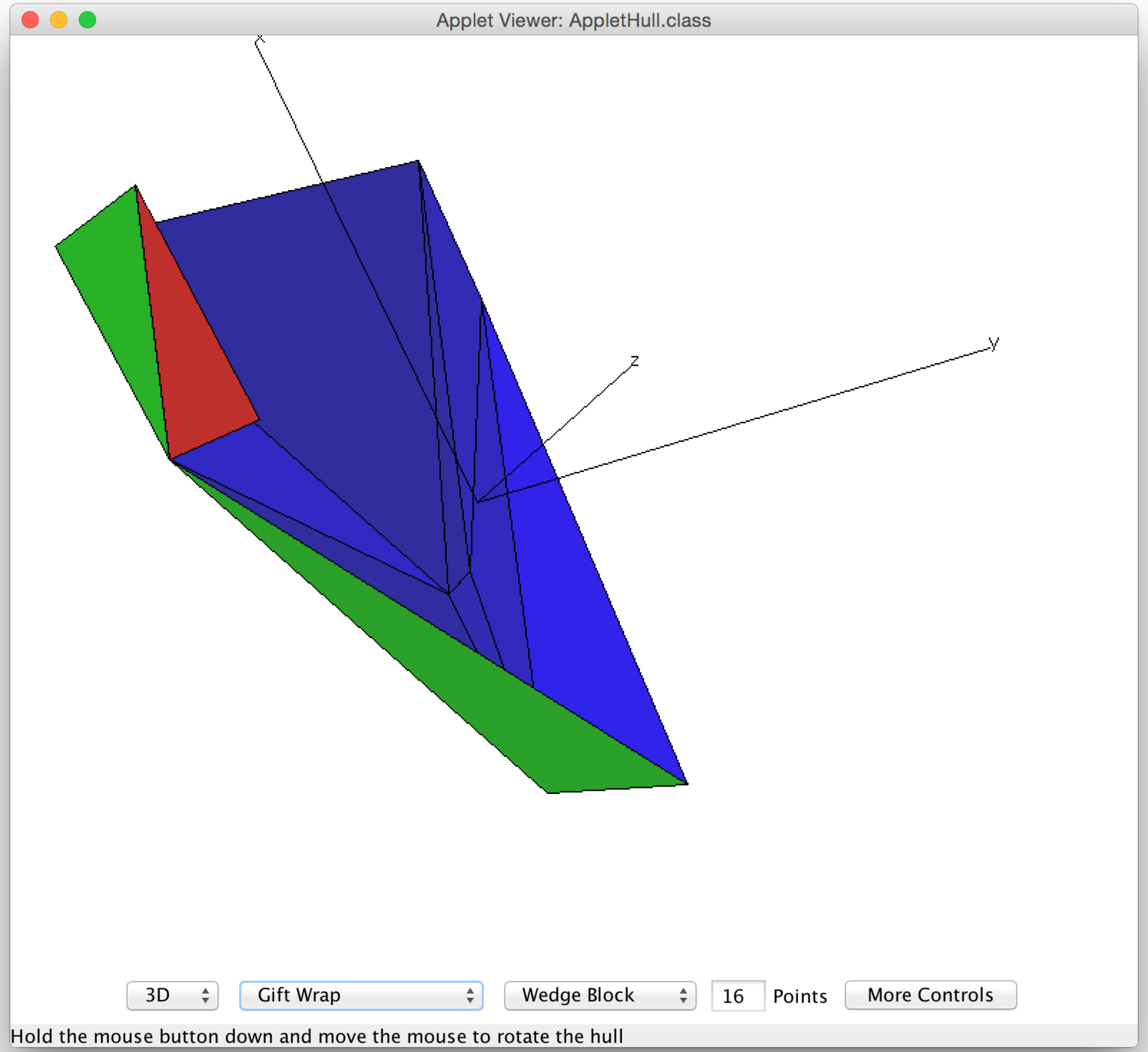
- One step of the incremental Clarkson-Shor algorithm:





Clarkson-Shor-Algorithm (randomized incremental)

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

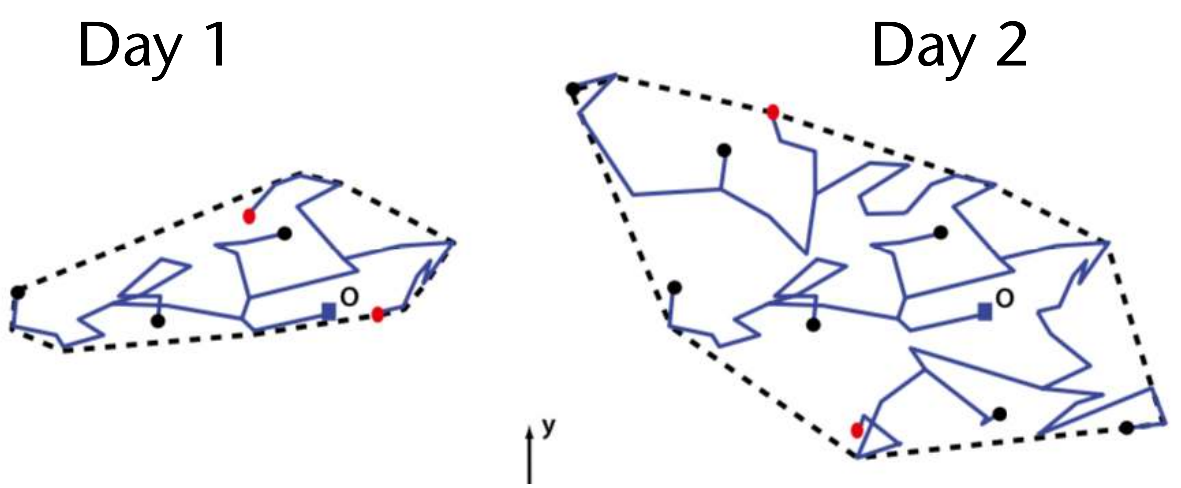


Different algorithms, e.g., gift wrapping

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

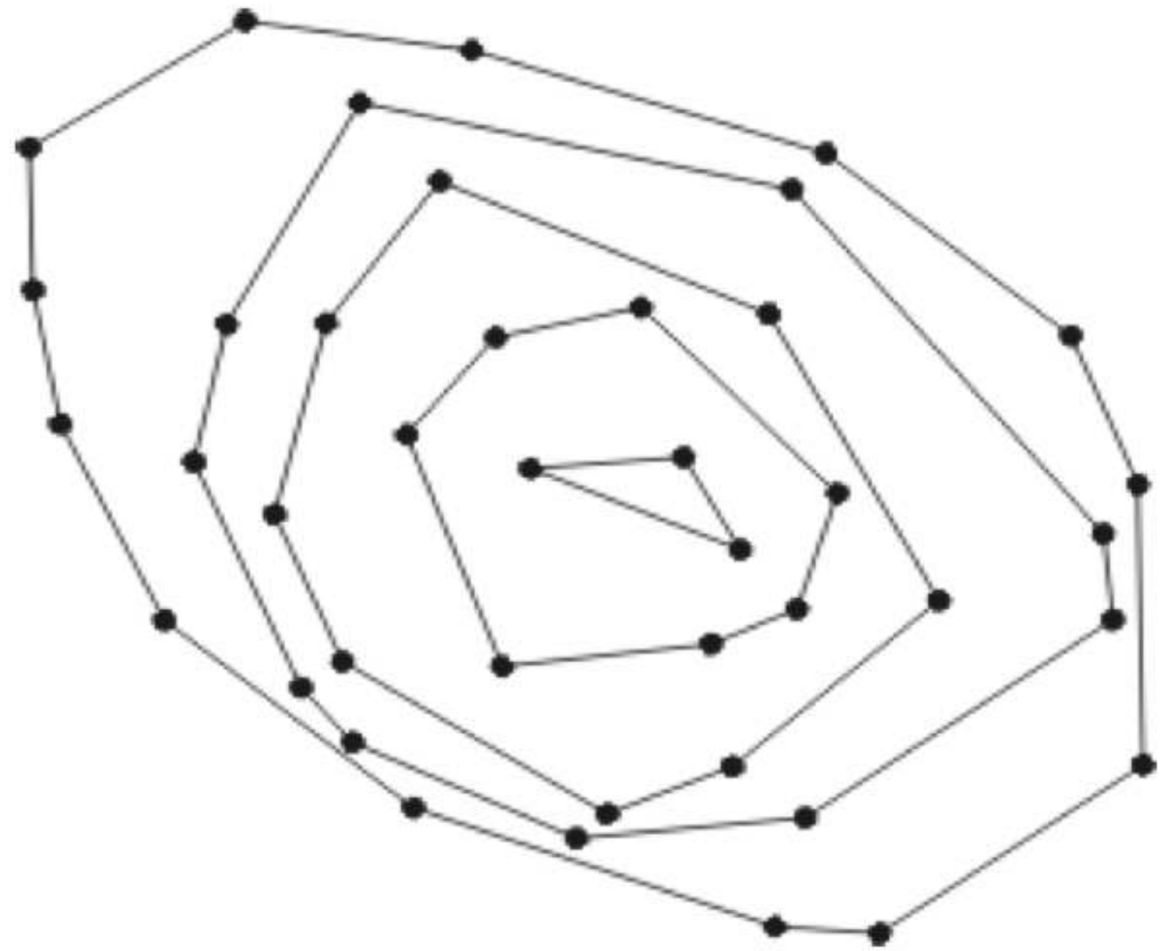
Applications of the Convex Hull

- Biology:
  - How much area does an animal occupy/need? ⟶ take the convex hull of all the points where it has been observed
  - Spatial extent of an outbreak in animal epidemics ⟶ convex hull of locations of all infected animals



- In physics engines:
  - Use the convex hull of objects as bounding volumes in *broad phase*
  - Calculate distance between CH's, or a separating plane
- Robot path planning:
  - Put convex hull around complex obstacle
  - Shortest path from S to T is either a straight line, or includes a part of the CH

Onion Peeling



- Ordering points by degree of "outsidedness":
  - Construct sequence of convex hulls (*onion peeling*)
- Can be used to
  - Estimate source of an event; points are sensors with readings above a threshold
  - Outlier detection and removal
- Finding the diameter of a set *S* of points:
  - Diameter = distance of farthest pair *p*,*q*
  - *p*,*q* must be on the convex hull
  - Walk around CH using a pair of antipodal points
- Diameter can be used for clustering: minimize the maximal diameter over all clusters

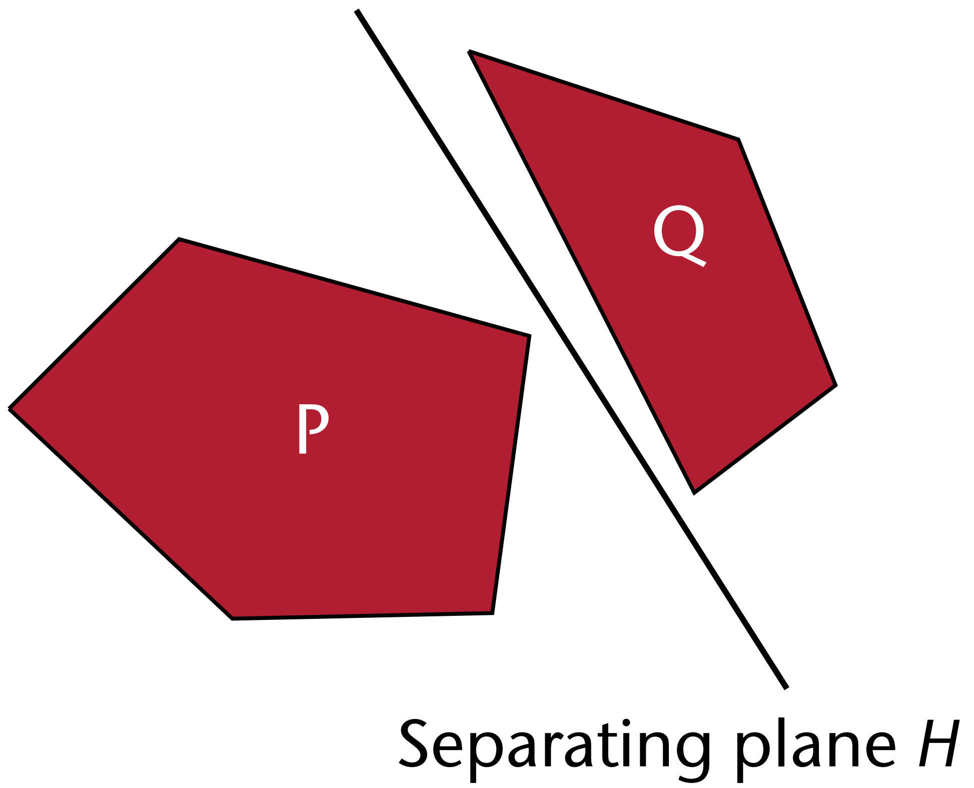
Extremely Fast Collision Detection for Convex Objects

- A condition for "non-collision":

  P and Q are "linearly separable" $:\Leftrightarrow$

  $$\exists \text{ half-space } H : P \subseteq H \wedge Q \subseteq H^c$$

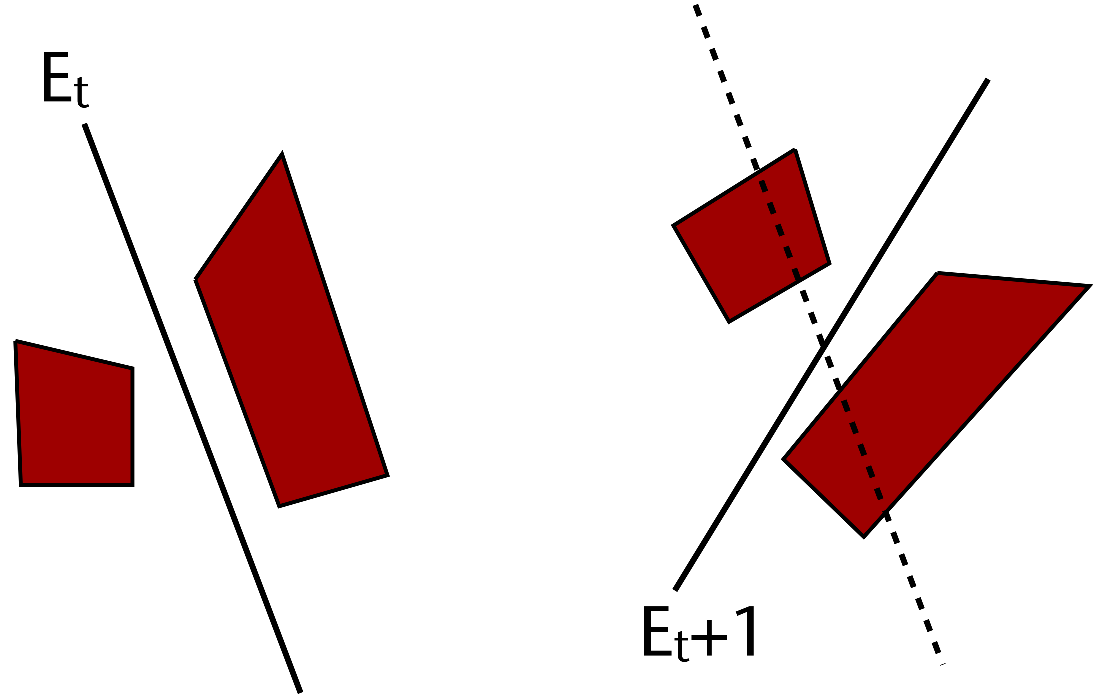
  (i.e., "P is completely on one side of H,
  Q completely on the other side")



- Preprocessing: for each coll.obj., compute its convex hull
- Runtime: try to find a separating plane quickly

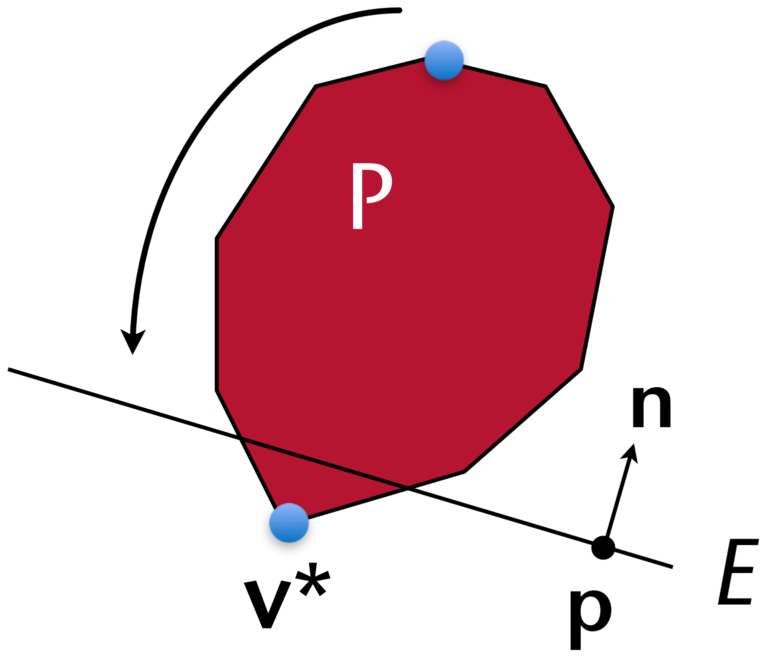
The "Separating Planes" Algorithm for Convex Coll.Det.

- The idea: utilize temporal coherence → if E_t was a separating plane between P and Q at time t, then the new separating plane E_{t+1} is probably not very "far" from E_t (perhaps it is even the same)



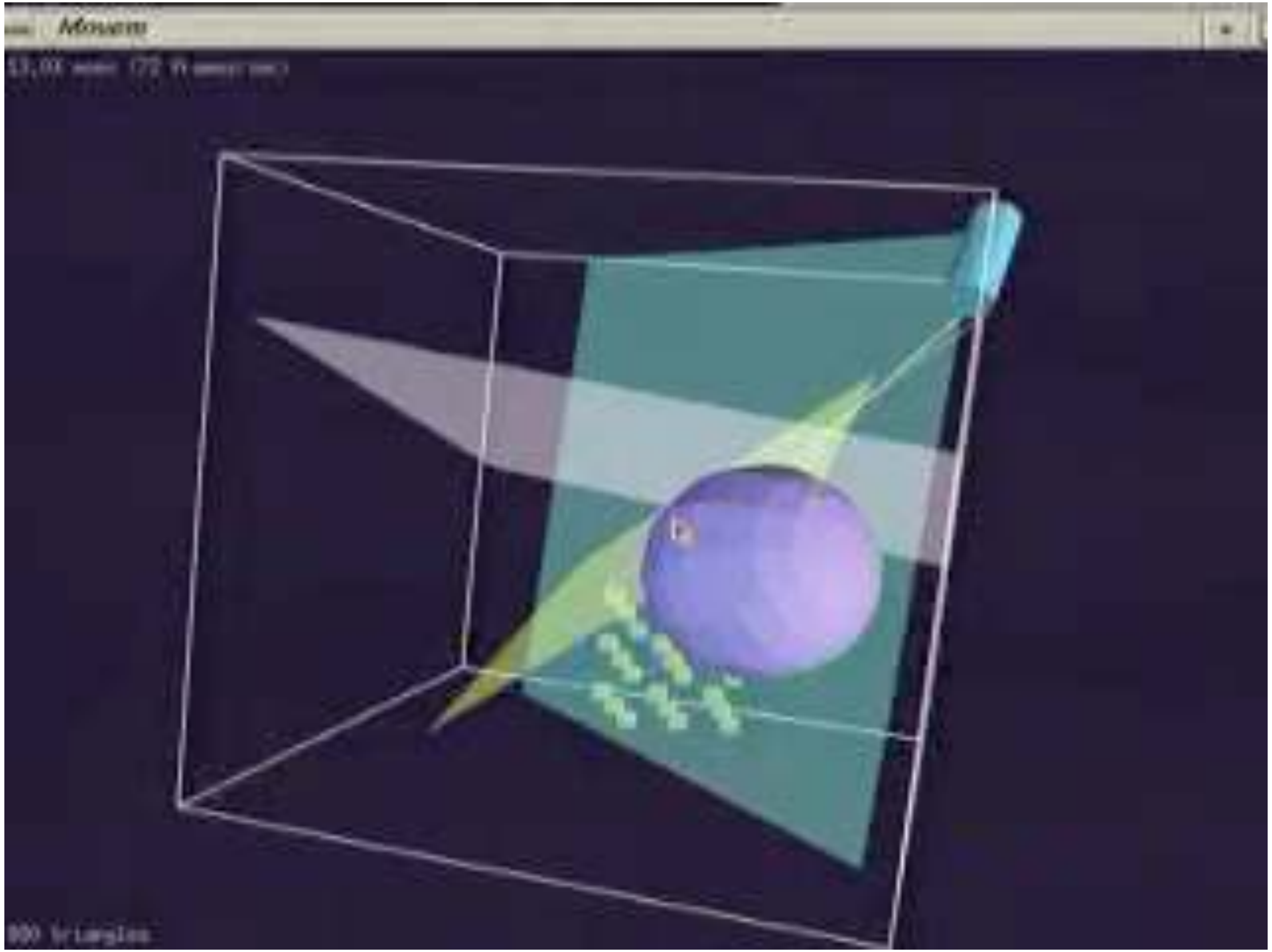
- Check candidate plane by steepest decent on the convex hull (from vertex to vertex)

$$(\mathbf{p}, \mathbf{n}) \text{ is separating plane } \Leftrightarrow$$
$$\forall \mathbf{v} \in P : f(\mathbf{v}) = (\mathbf{v} - \mathbf{p}) \cdot \mathbf{n} > 0$$

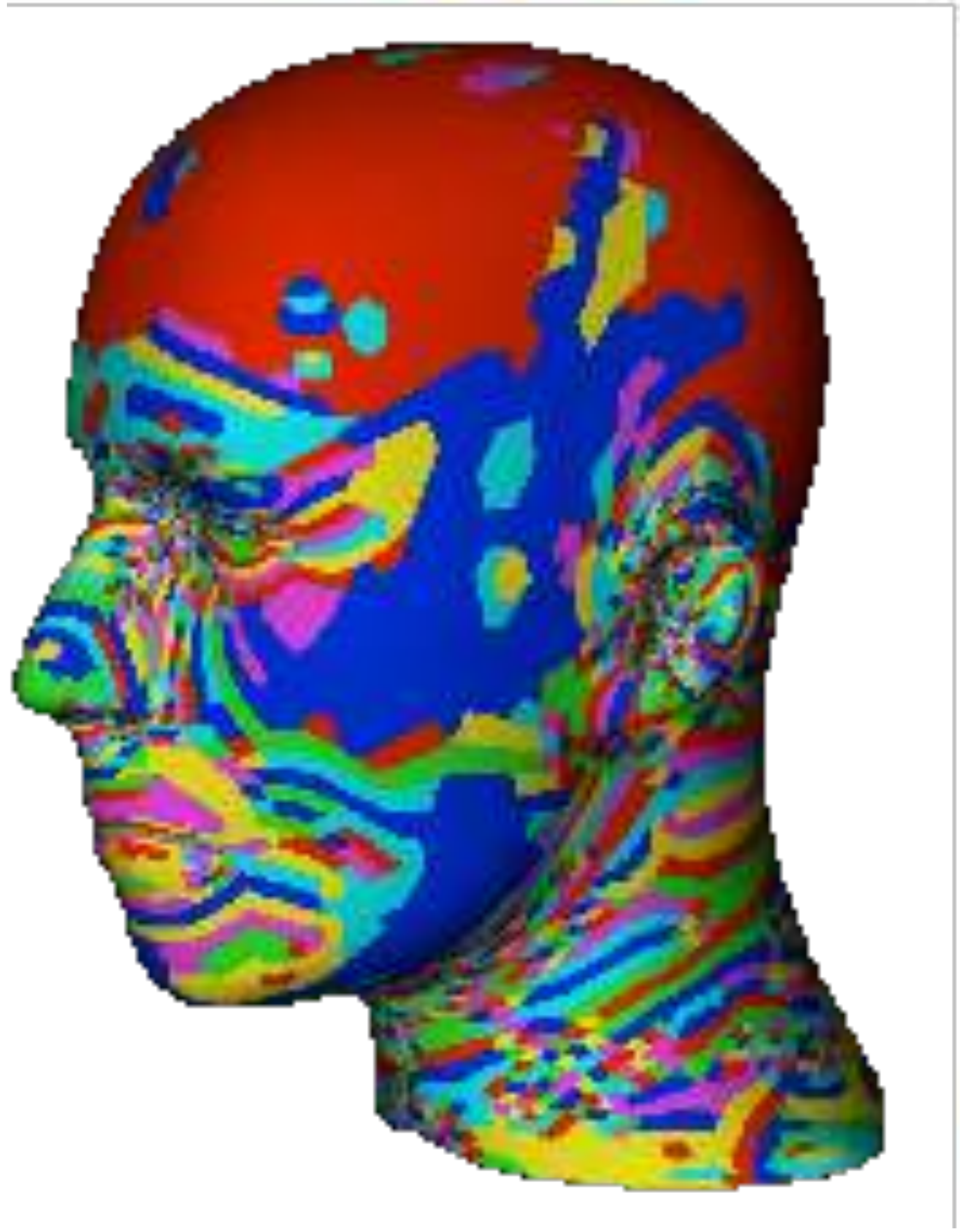


- For details: see *Advanced Comp. Graphics*

Visualization



Convex Surface Decomposition

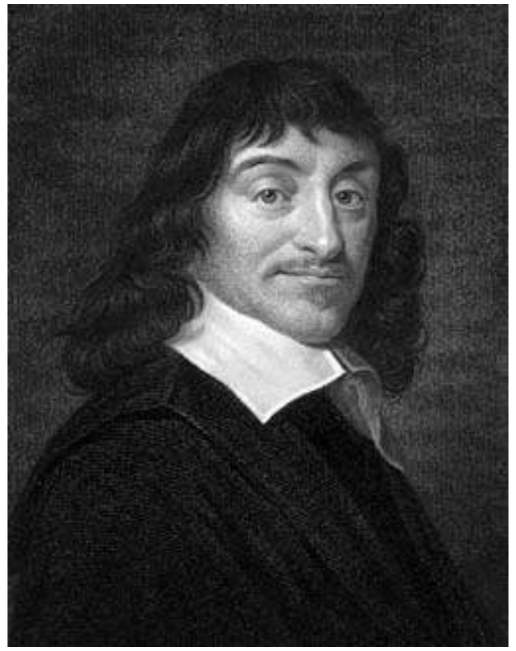


Decomposition into convex surface patches

Convex pieces at a medium level of the hierarchy (green = orig. surface, red = free surface, yellow = "contained")

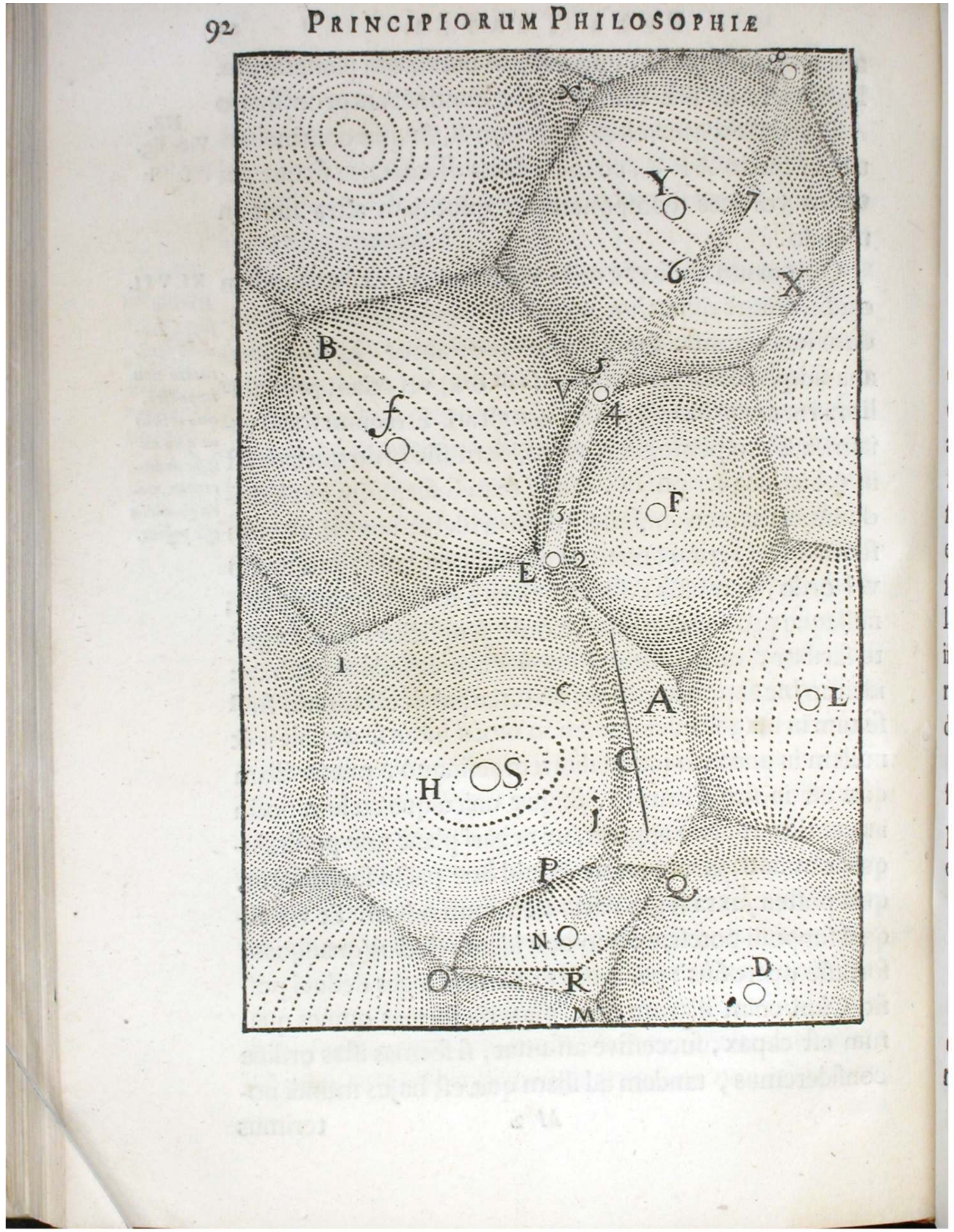
Voronoi Diagrams

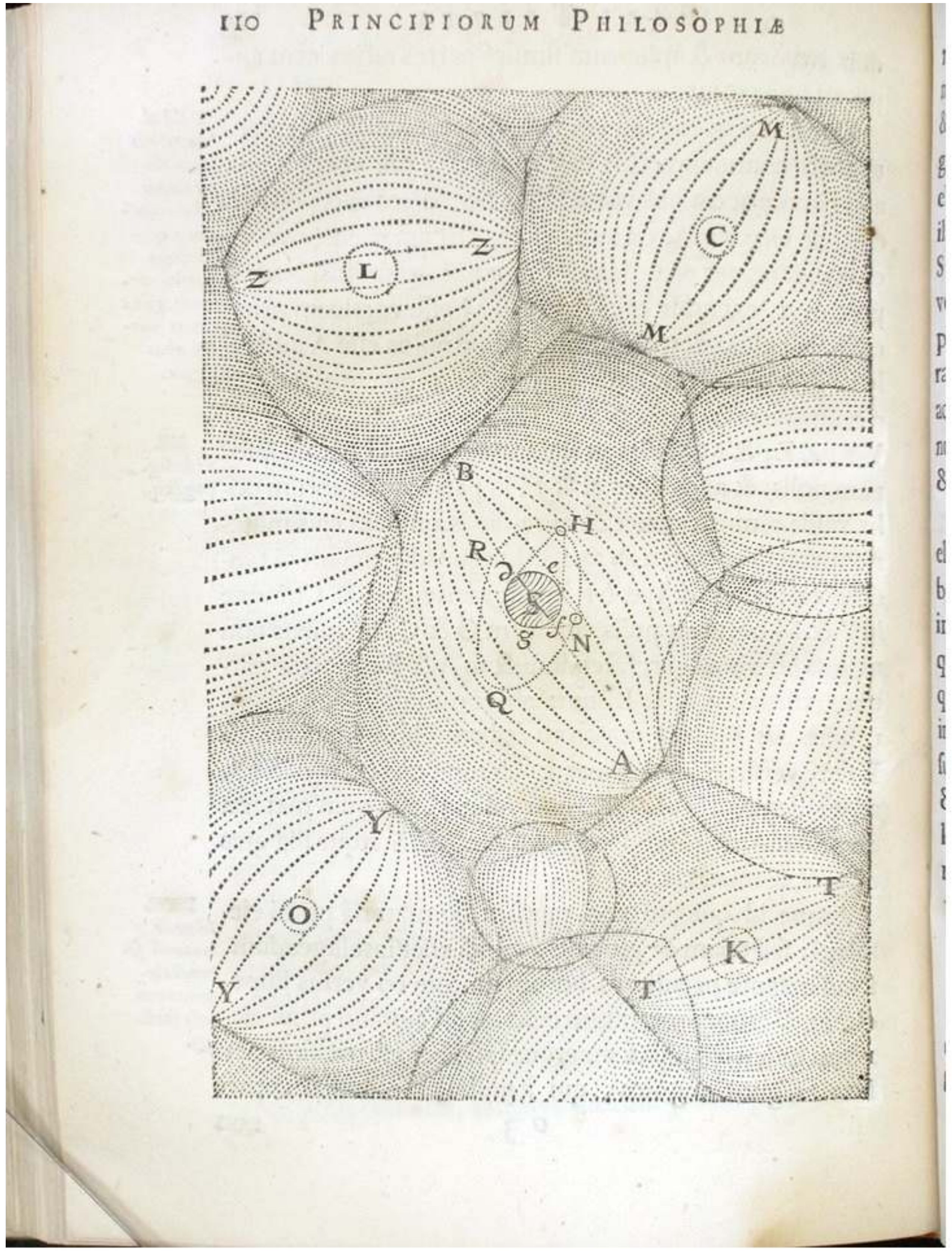
- One of the first mentions are in René Descarte's (Cartesius') *Principiorum Philosophiae*, 1644:
  - Imagined that the universe is filled with matter that is attracted to the stars and swirls around them



- Georgy F. Voronoi (Георгий Ф. Вороной, 1868–1908)
  - Born Ukraine (part of Russian empire at the time)
  - Professor in Warsaw
  - Student: Delaunay

Descartes' Vortices





Delaunay (1890 – 1980)

- Student of Voronoi (and Grave)
- One of the 3 best Russian mountaineers around 1930
- Russian spelling: Борис Николаевич Делоне
  - At that time, French (and German) was the language of science!

- Not to be confused with the painter Robert Delaunay!
    - 1885 – 1941 ; really French

Champs de Mars. La Tour rouge. 1911

Homage à Bleriot, 1914

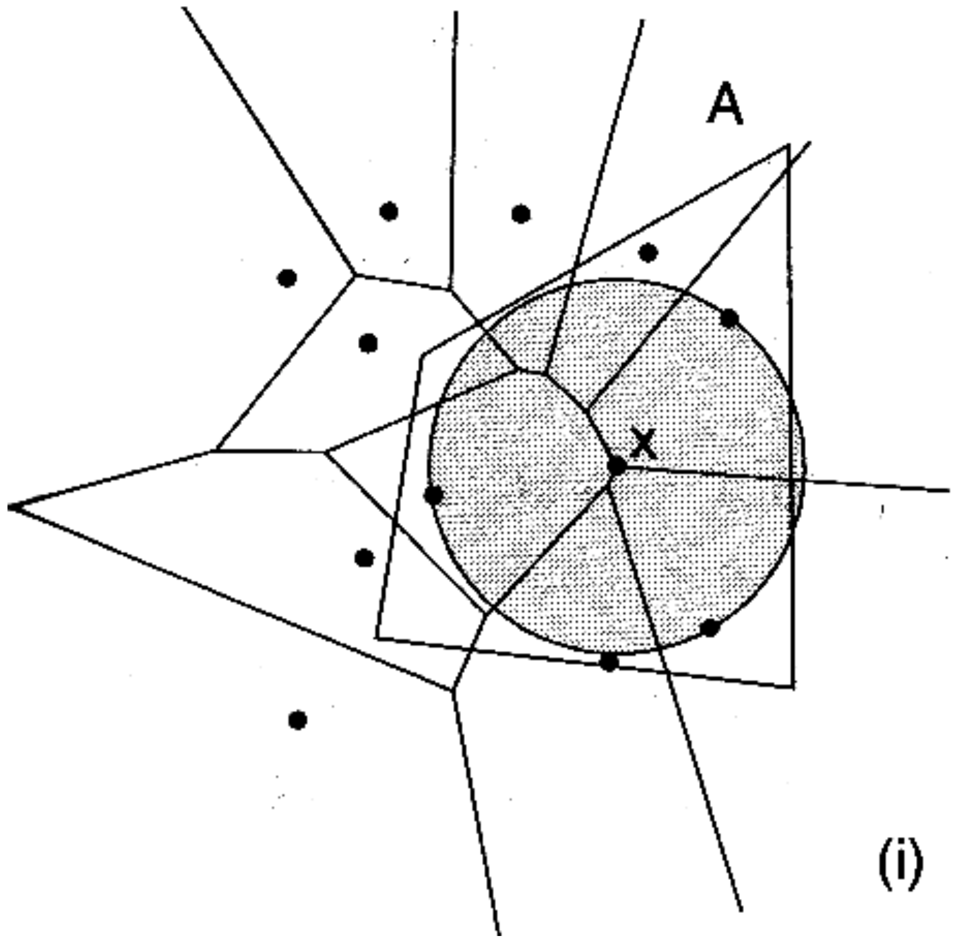
Independent Discoveries in Other Fields

Descartes	Astronomy	1644	“Heavens” (= Voronoi regions)
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

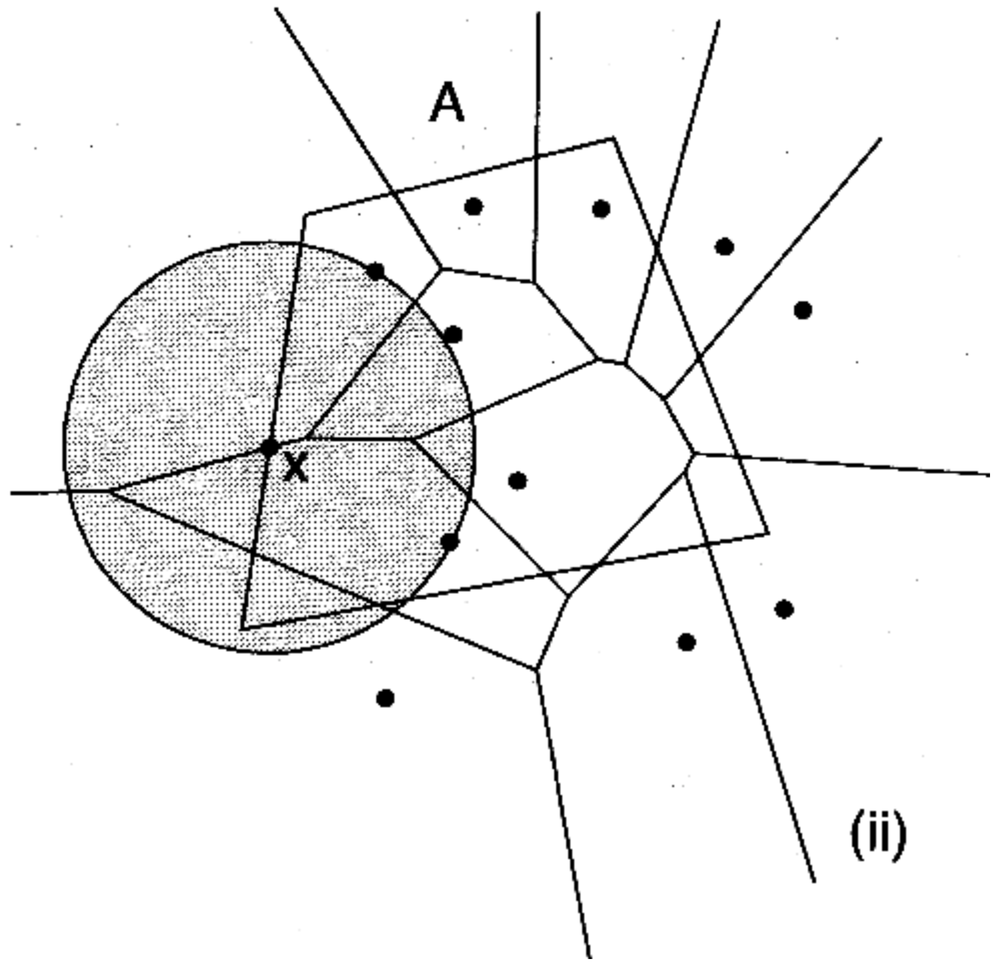
Application: Maximal Empty Circles Constrained by a Polygon

Task: find location of maximal circle such that

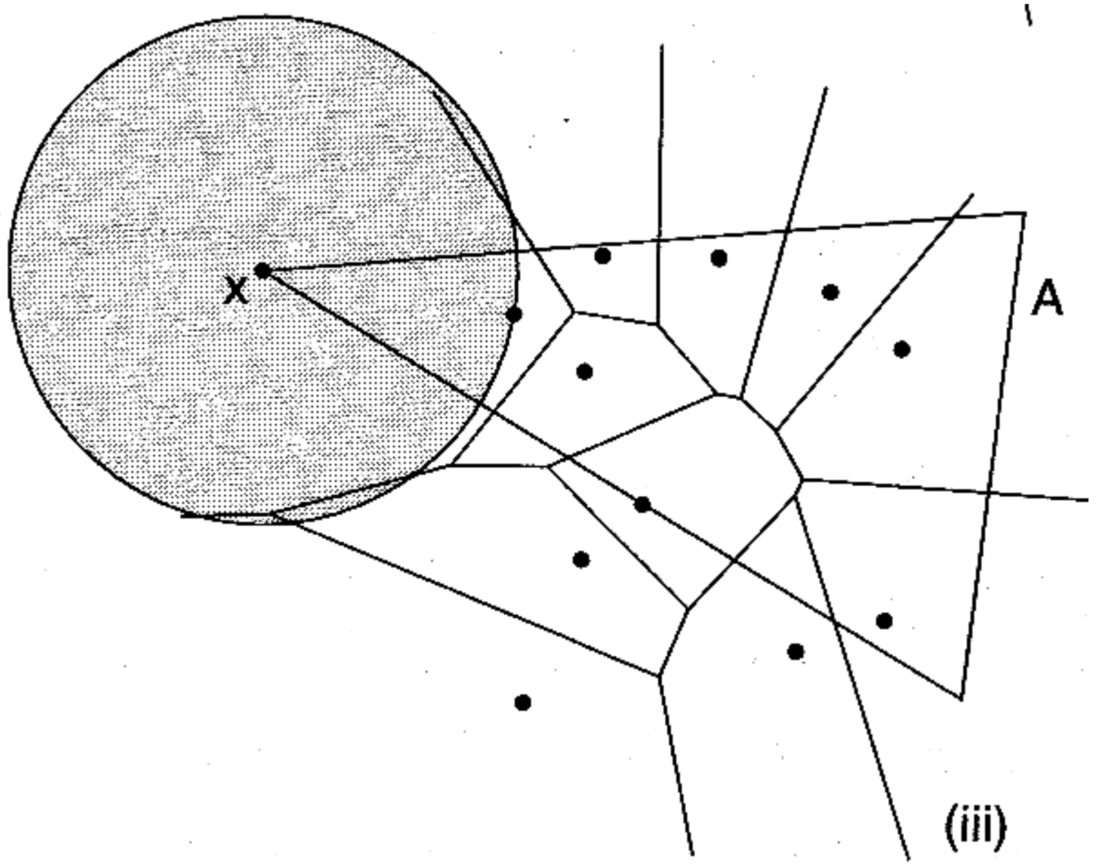
1. its center is inside polygon A
2. it does not contain any of the points



(i)



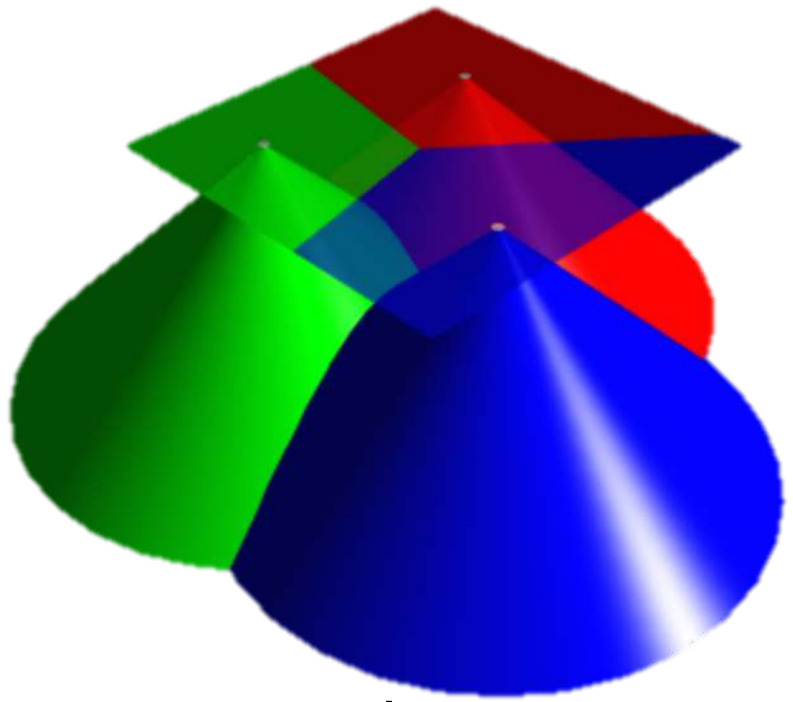
(ii)



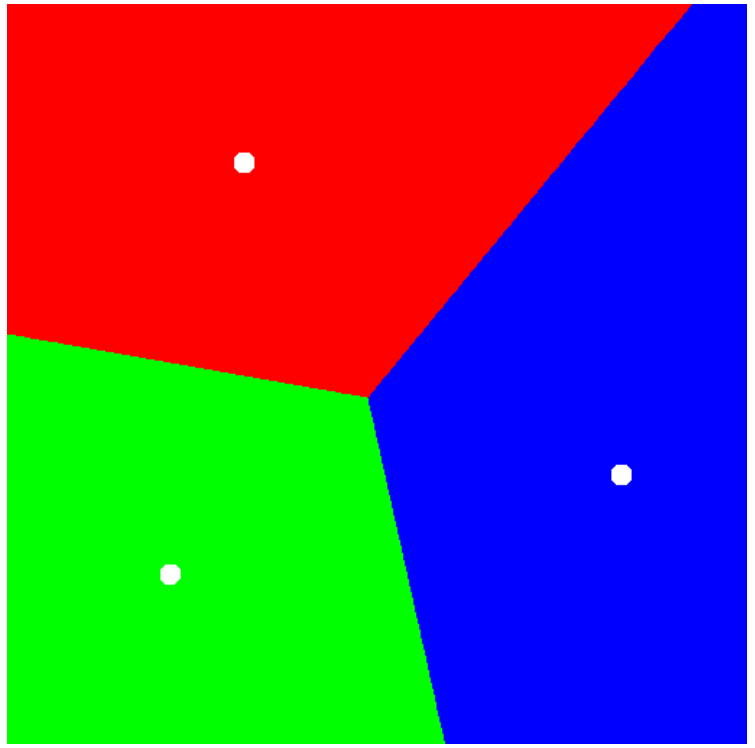
(iii)

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

- Observation:
  - Place a cone at every Voronoi site in the plane with 90° angle at the apex
  - Distance of a point X from Voronoi site = height of cone below X
- Method:
  - For each site, render a cone with different color (= site ID)
  - Borders in color buffer = Voronoi edges
  - Value in Z-buffer = distance from site
- This technique was already mentioned by Dirichlet & Voronoi

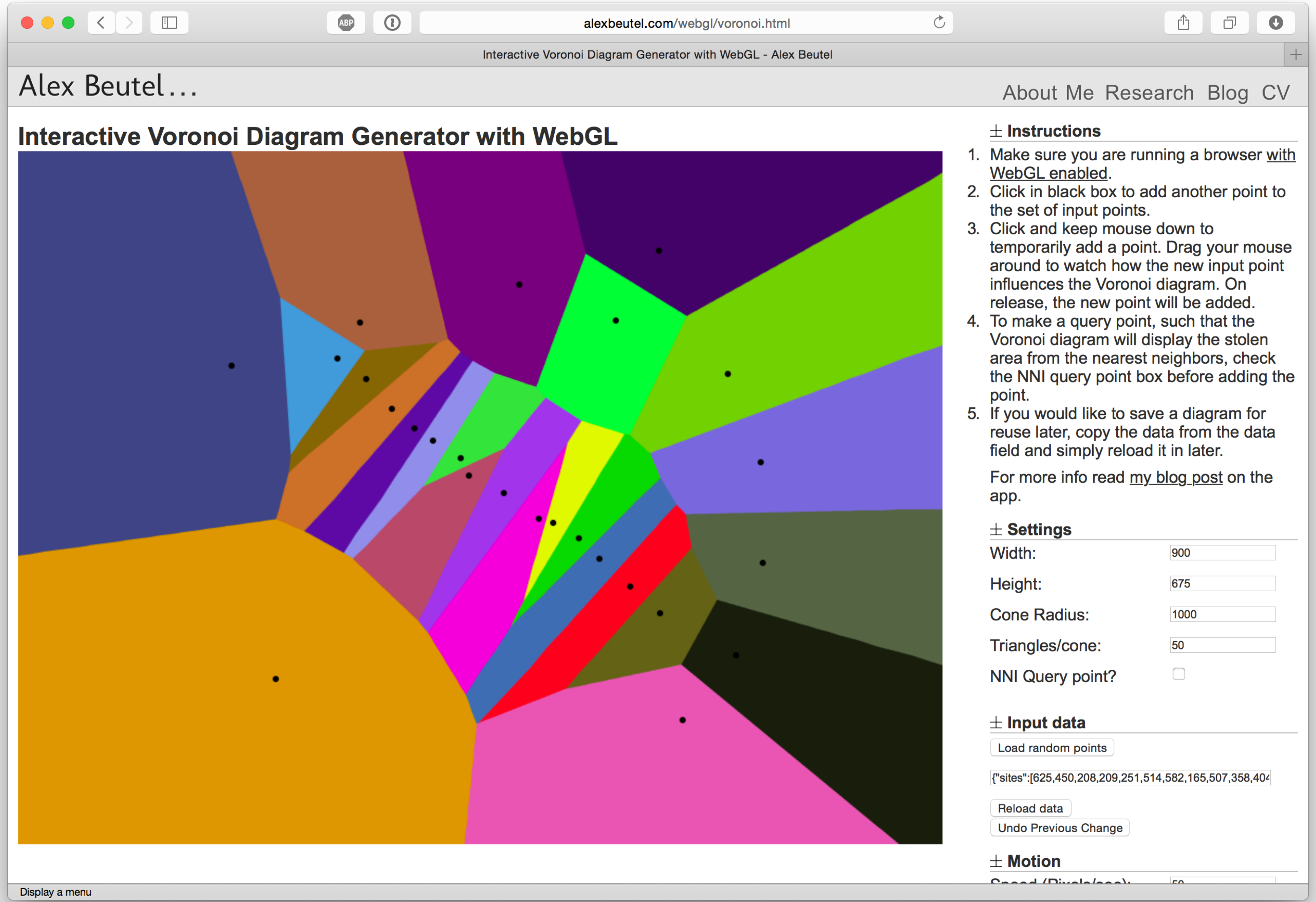


Side view



Top view

Demo



Hint at natural coordinates. Mention Mean Value Coordinates (which are better)

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

Complexity in Higher Dimensions

- The Voronoi diagram over n points in d-dim. space comprises, in each dimension j, $0 \le j \le d\text{-}1$, a number, f_j , of j-dimensional facets. Those numbers are in the order of

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

Generalizations of the Voronoi Diagram

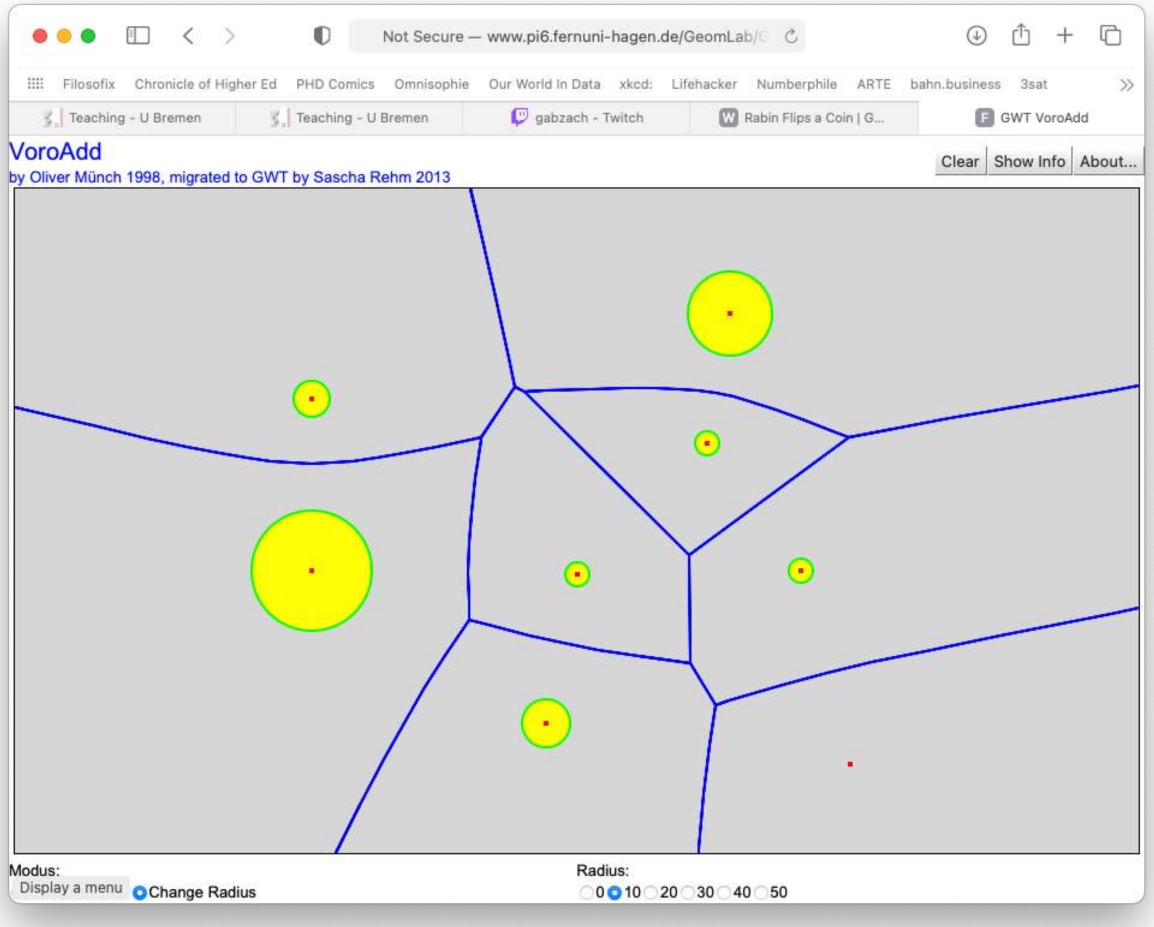
- Other distance functions
- Other objects as sites/generators
- Higher dimensions
- Other equivalence classes
- …

Voronoi Diagrams with Weights

- Generalize the distance function between point $\mathbf{x}$ and site $\mathbf{p}_i$
- Additive weights:

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

  - Bisectors are hyperbolic arcs (and lines)
  - A.k.a. Appolonius diagram
- Example



https://www.fernuni-hagen.de/ks/forschung/geom_lab.shtml

- Multiplicative weights:

$$d(\mathbf{x}, \mathbf{p}_i) = \frac{1}{w_i}\|\mathbf{x} - \mathbf{p}_i\|$$

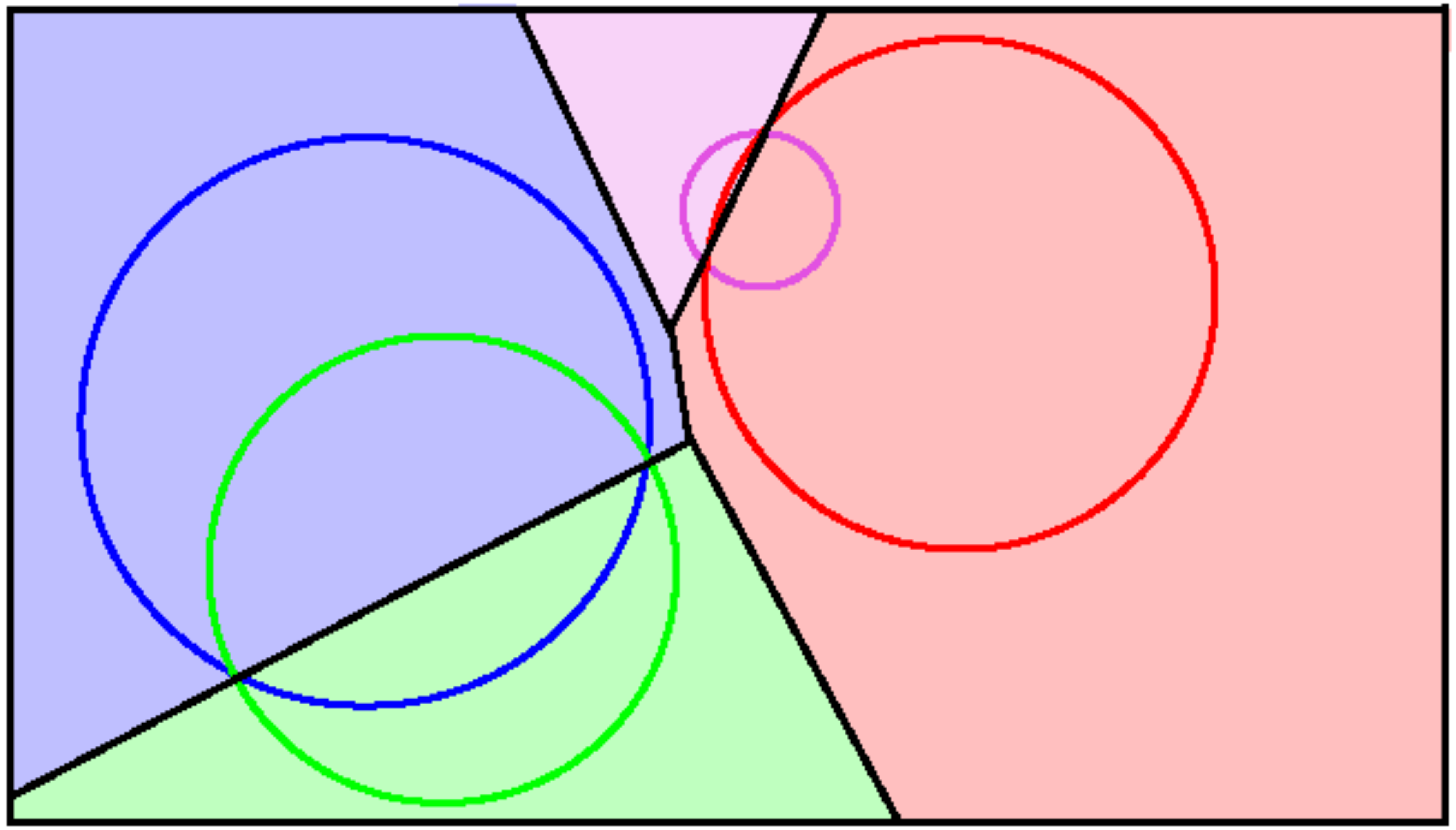
  - Bisectors are circular arcs (and straight lines)

The Power Diagram

- Different distance function:

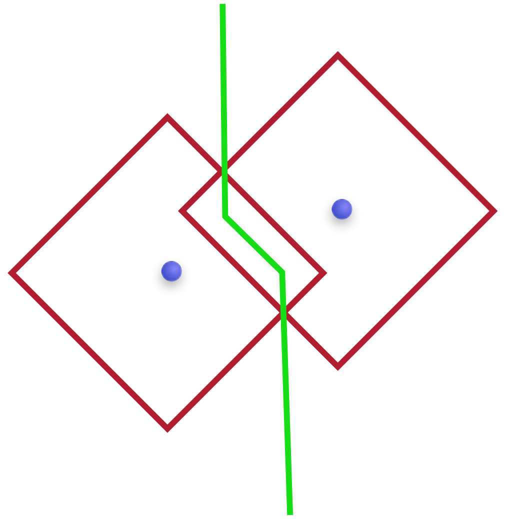
$$d(\mathbf{x}, \mathbf{p}_i) = (\mathbf{x} - \mathbf{p}_i)^2 - r_i$$

- Here, bisectors are lines!
- Example:

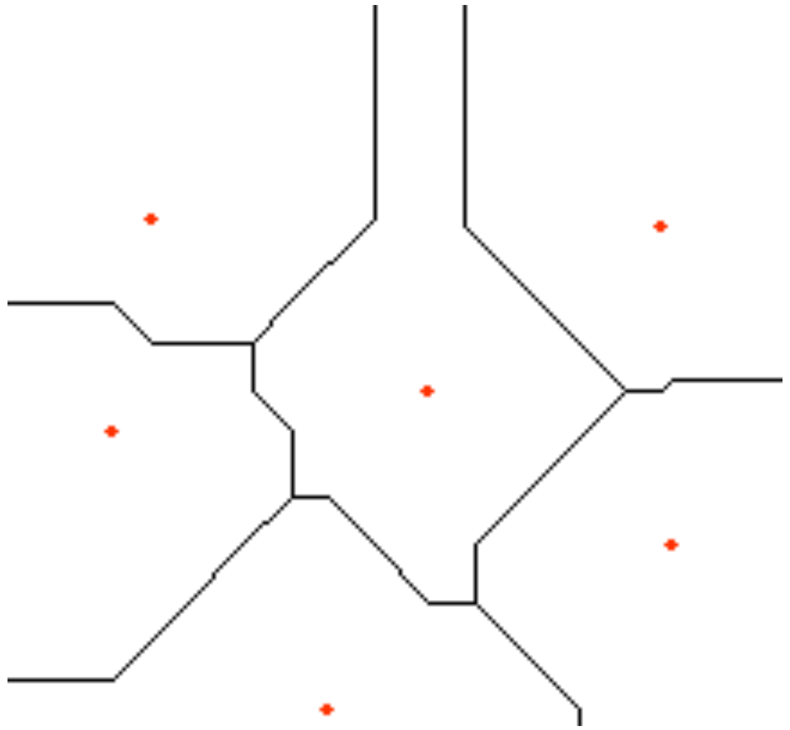


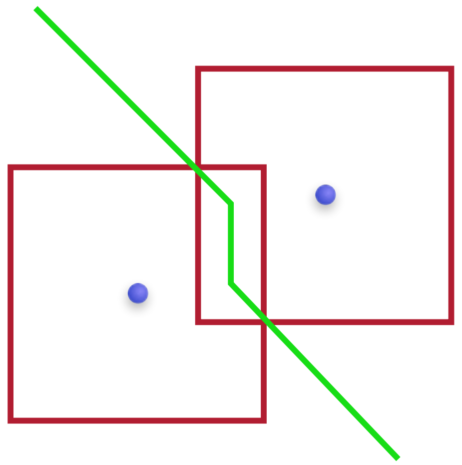
Other Distance Metrics

- Voronoi diagram using L_1 and L_∞ norm:

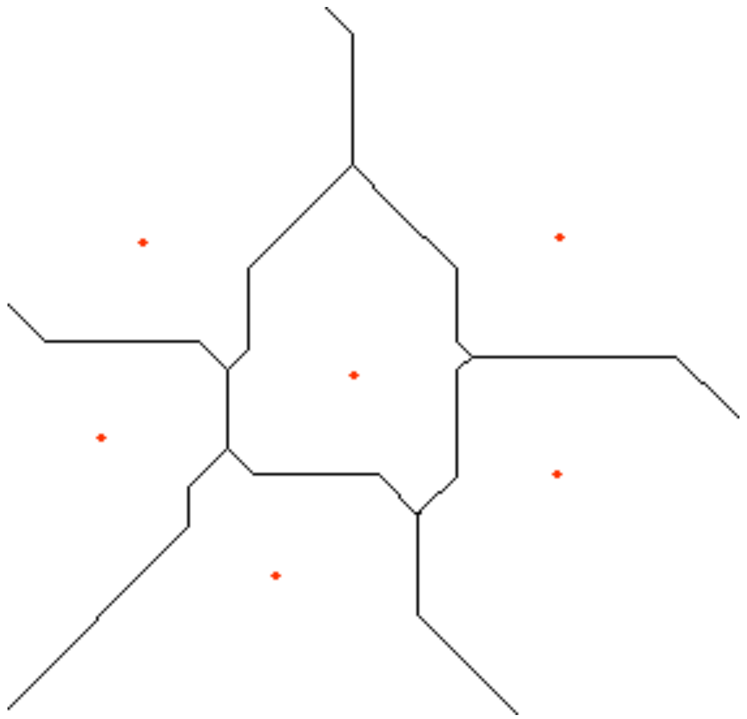


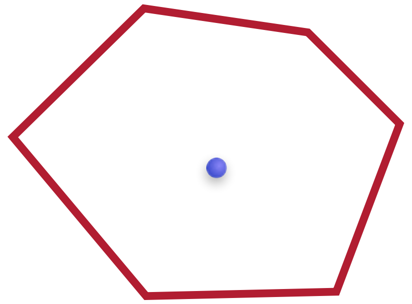
L_1-norm
(Manhattan norm)





L_∞ - norm
(maximum-norm)

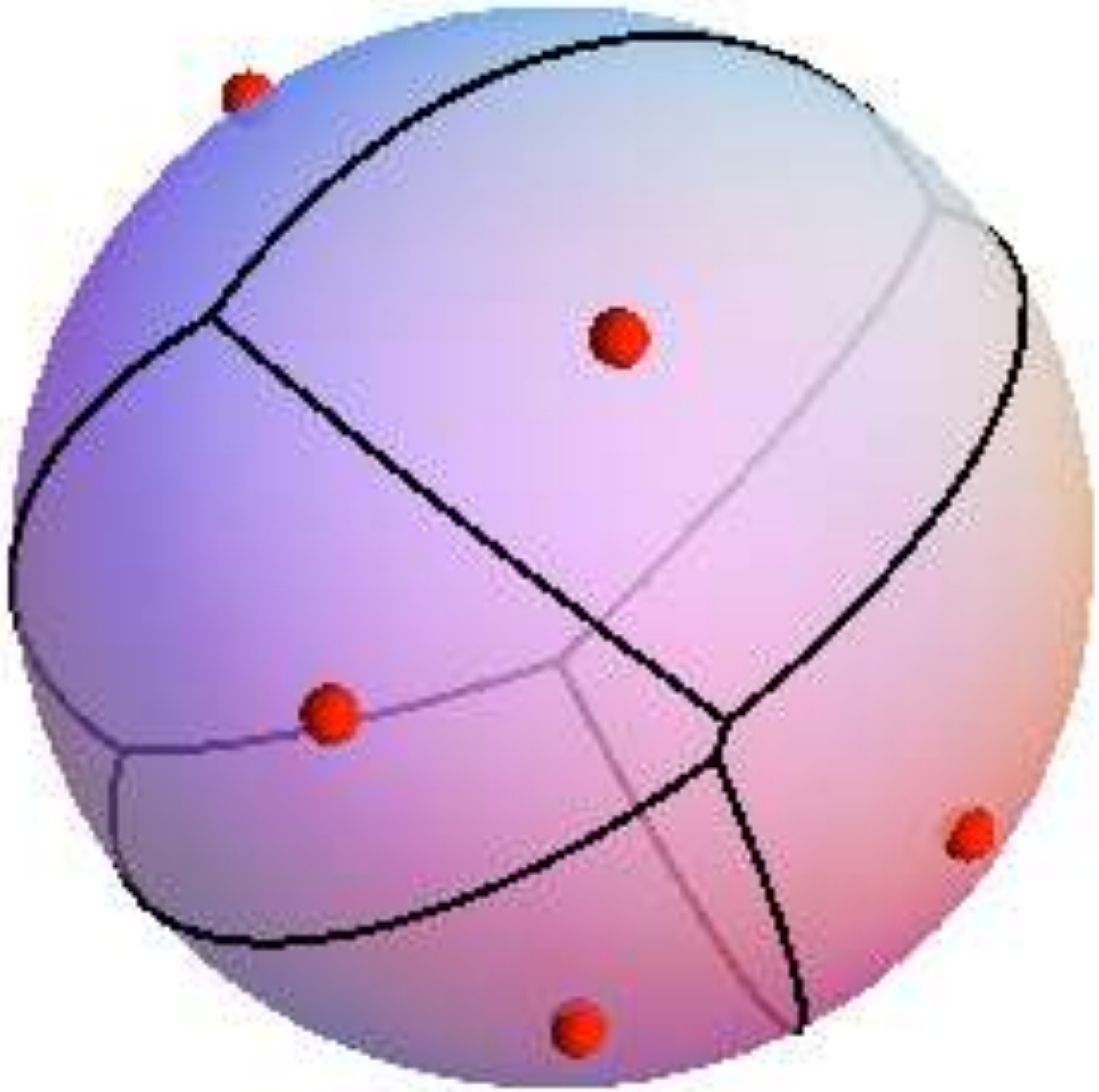


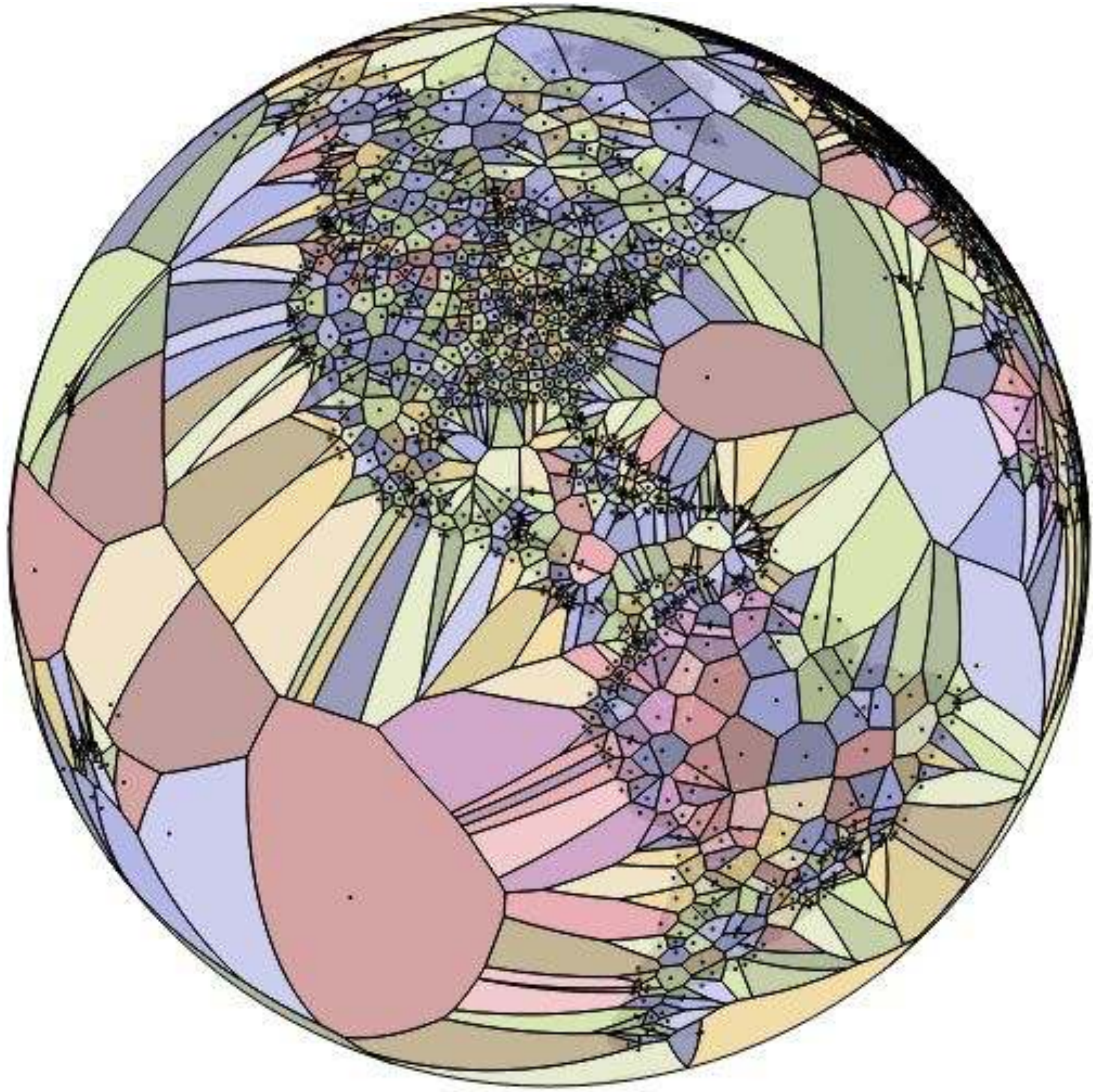


Convex norm
(can be defined over any convex polygon!)

Voronoi Diagrams on Other Two-Manifolds (e.g. Sphere)

- On the sphere, bisectors are great circles





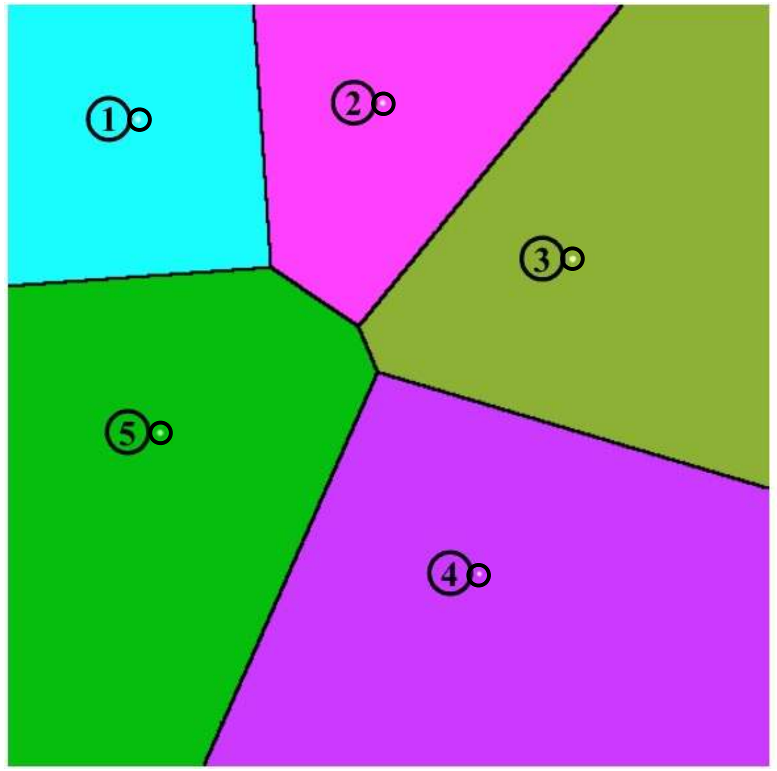
Higher-Order Voronoi Diagrams

- Definition:
  In a Voronoi diagram of *k*-th order, $V_k(S)$, all points in space belong to the *same* Voronoi region that have the *same* set of k nearest neighbors in S.

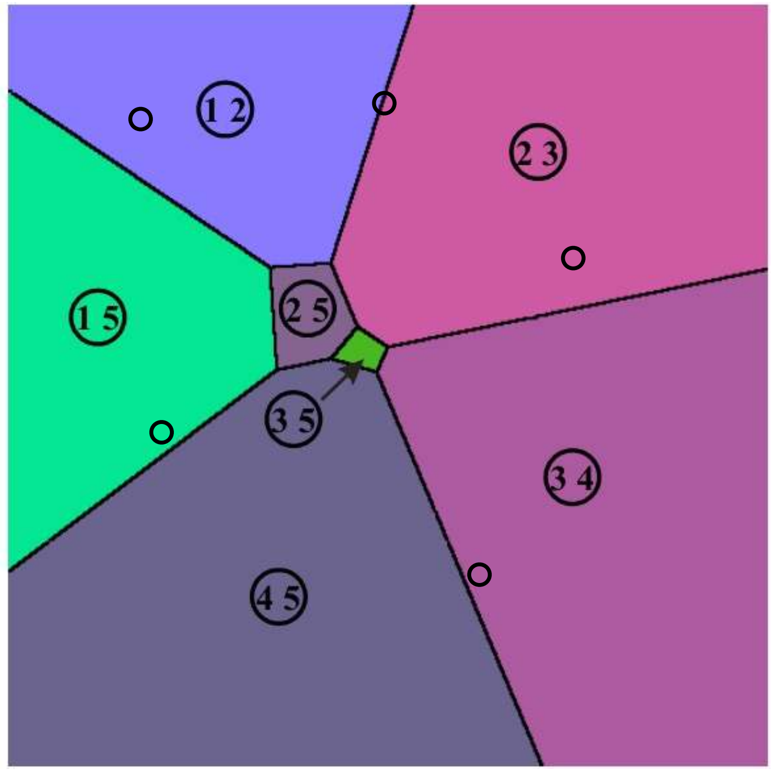
- Differences to the classical Voronoi diagram:
  - A "bisector" can contribute to *several*, *different* Voronoi edges!
  - A Voronoi region no longer necessarily contains its generators (Voronoi sites)

Examples

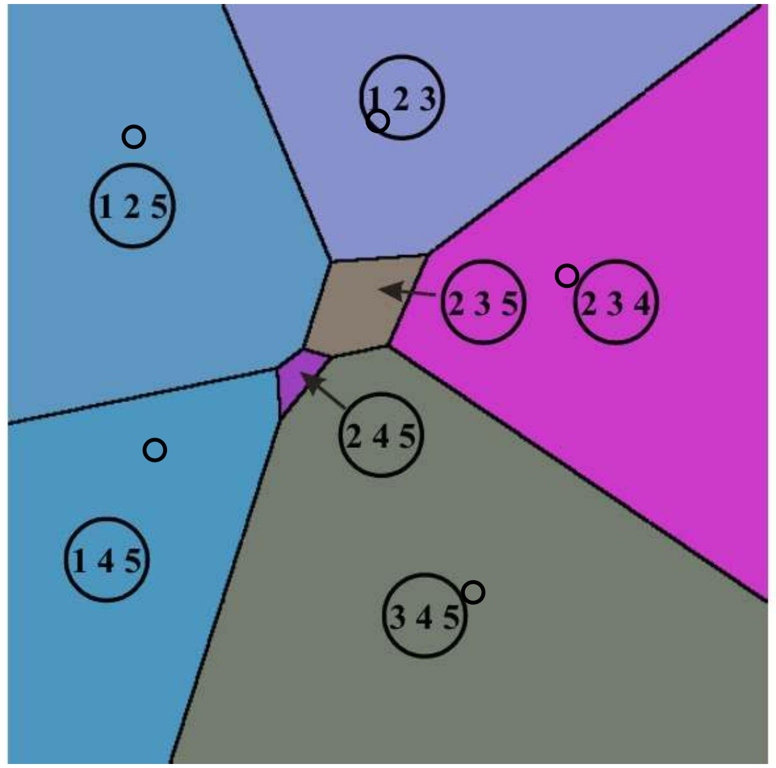
1-st order



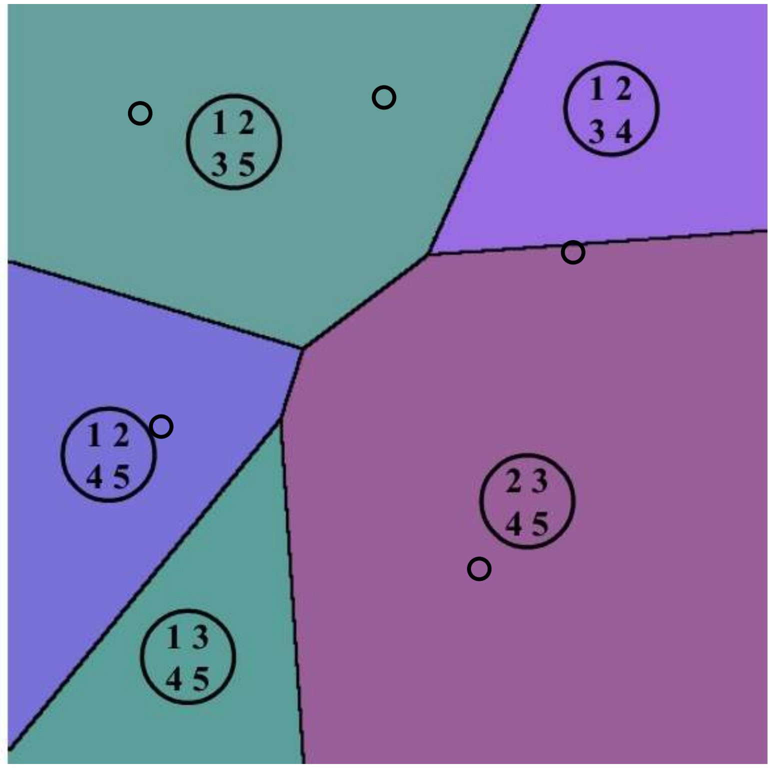
2-nd order



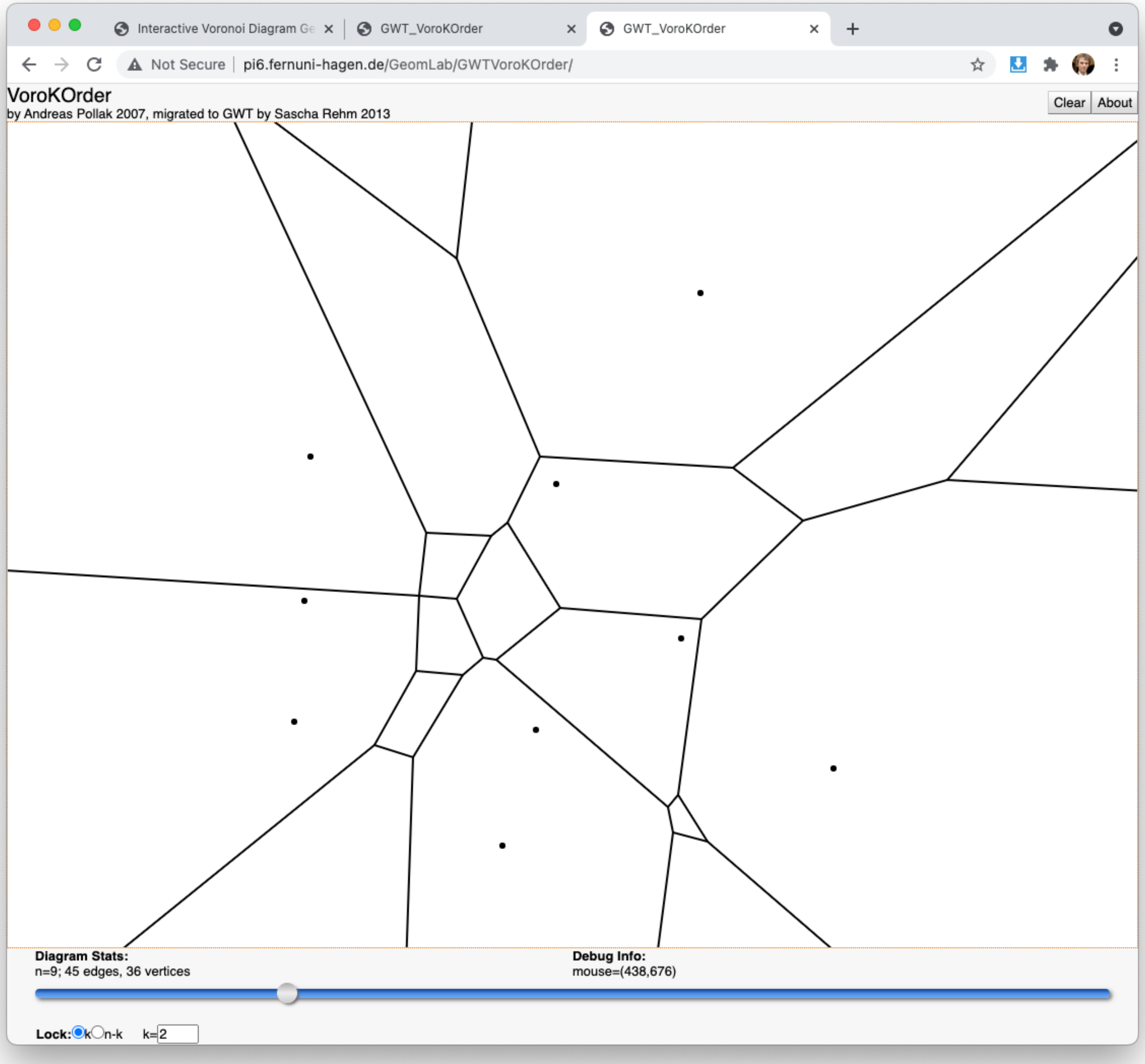
3-rd order



4-th order



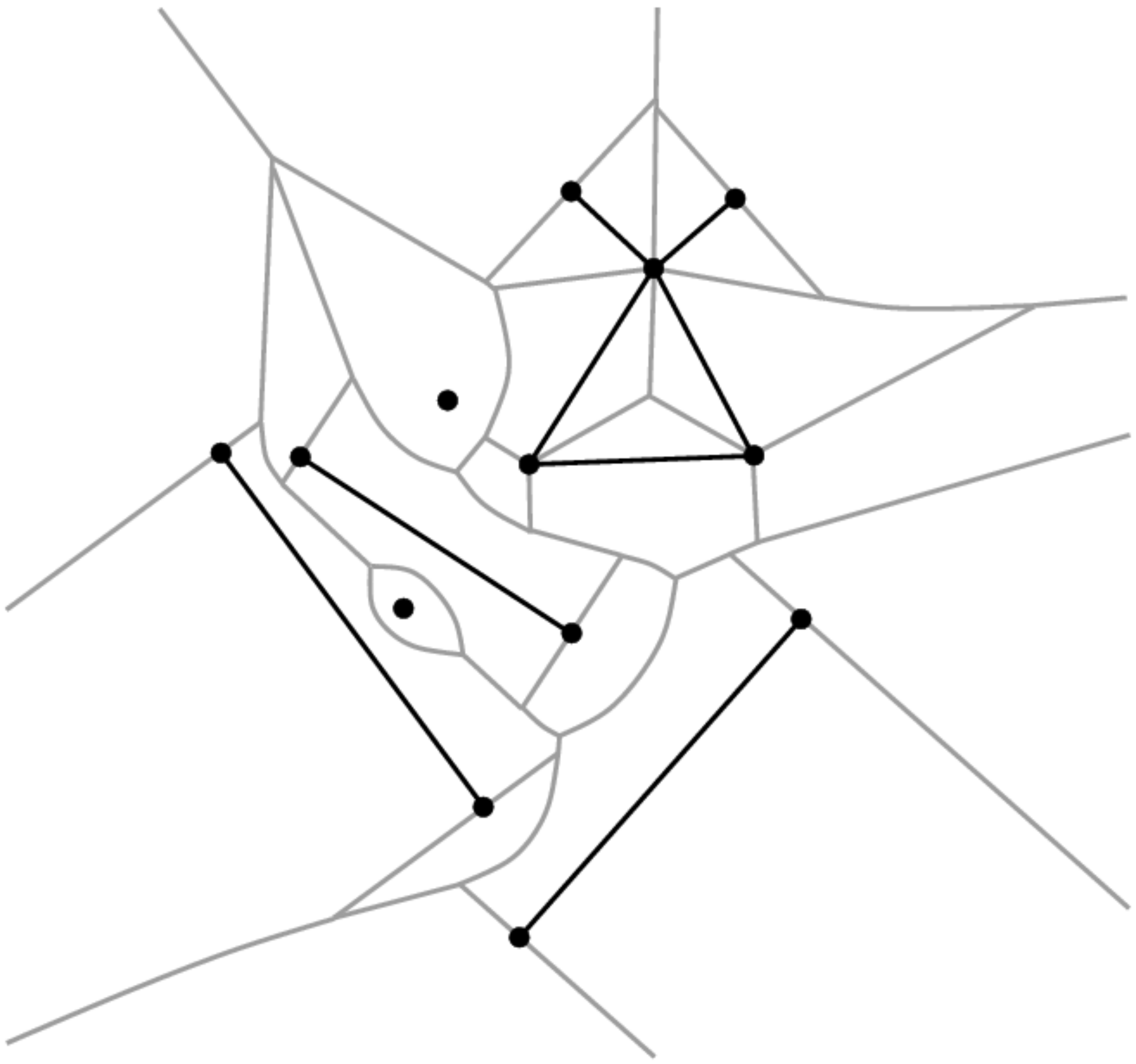
Demo



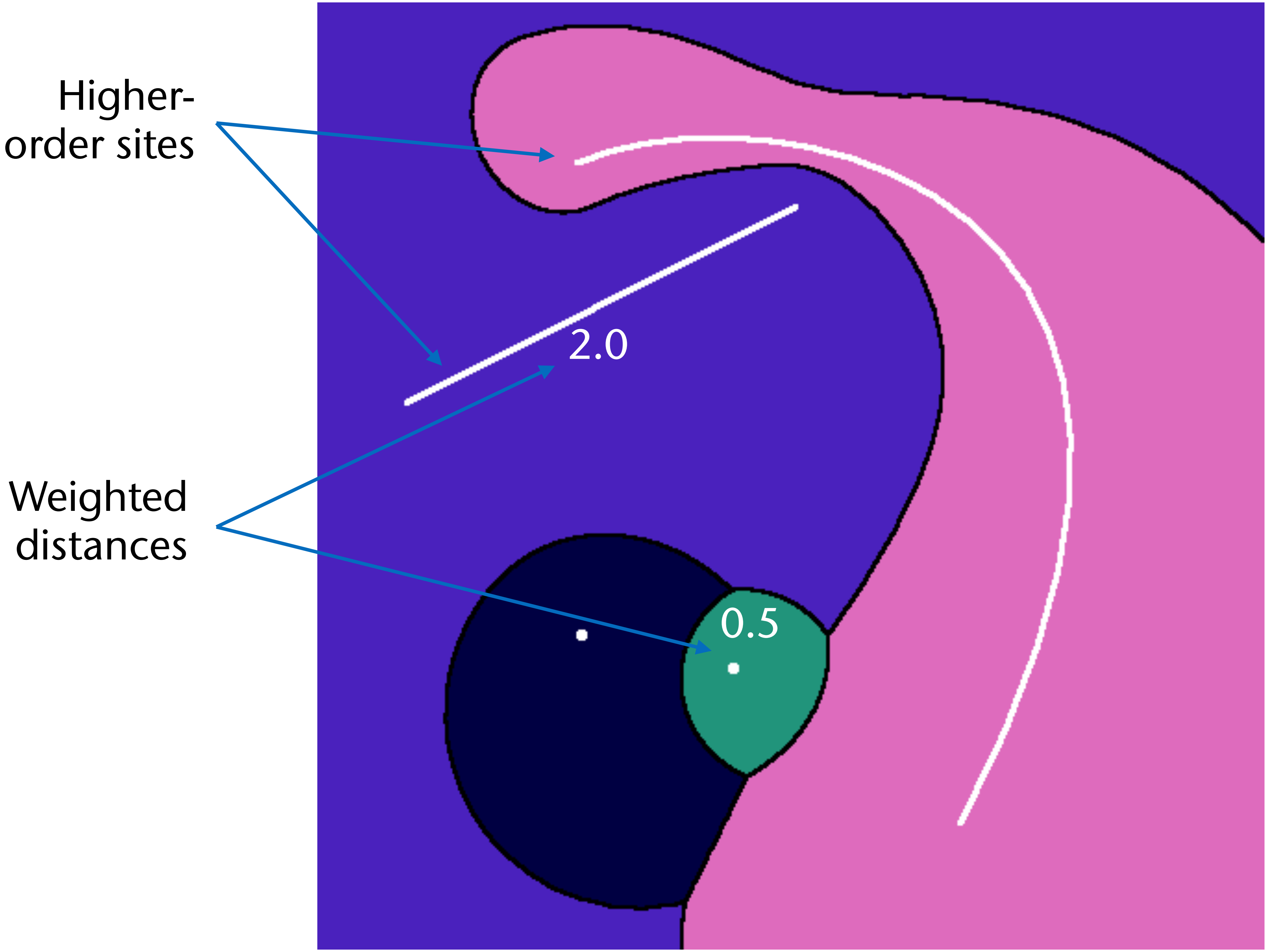
https://www.fernuni-hagen.de/ks/forschung/geom_lab.shtml

Voronoi Diagrams over Line Segments

- Sites (generators) are now points and line segments
- Bisectors = lines and parabolas
- Example:

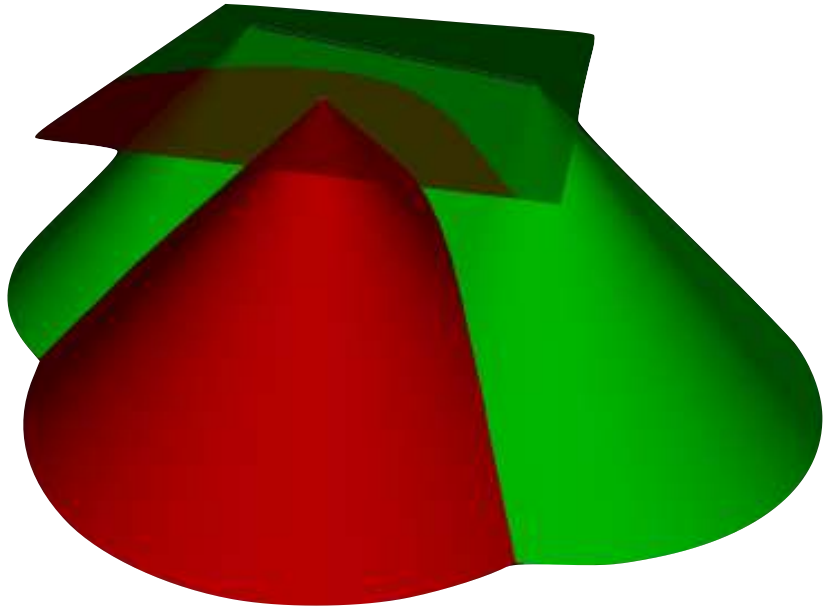


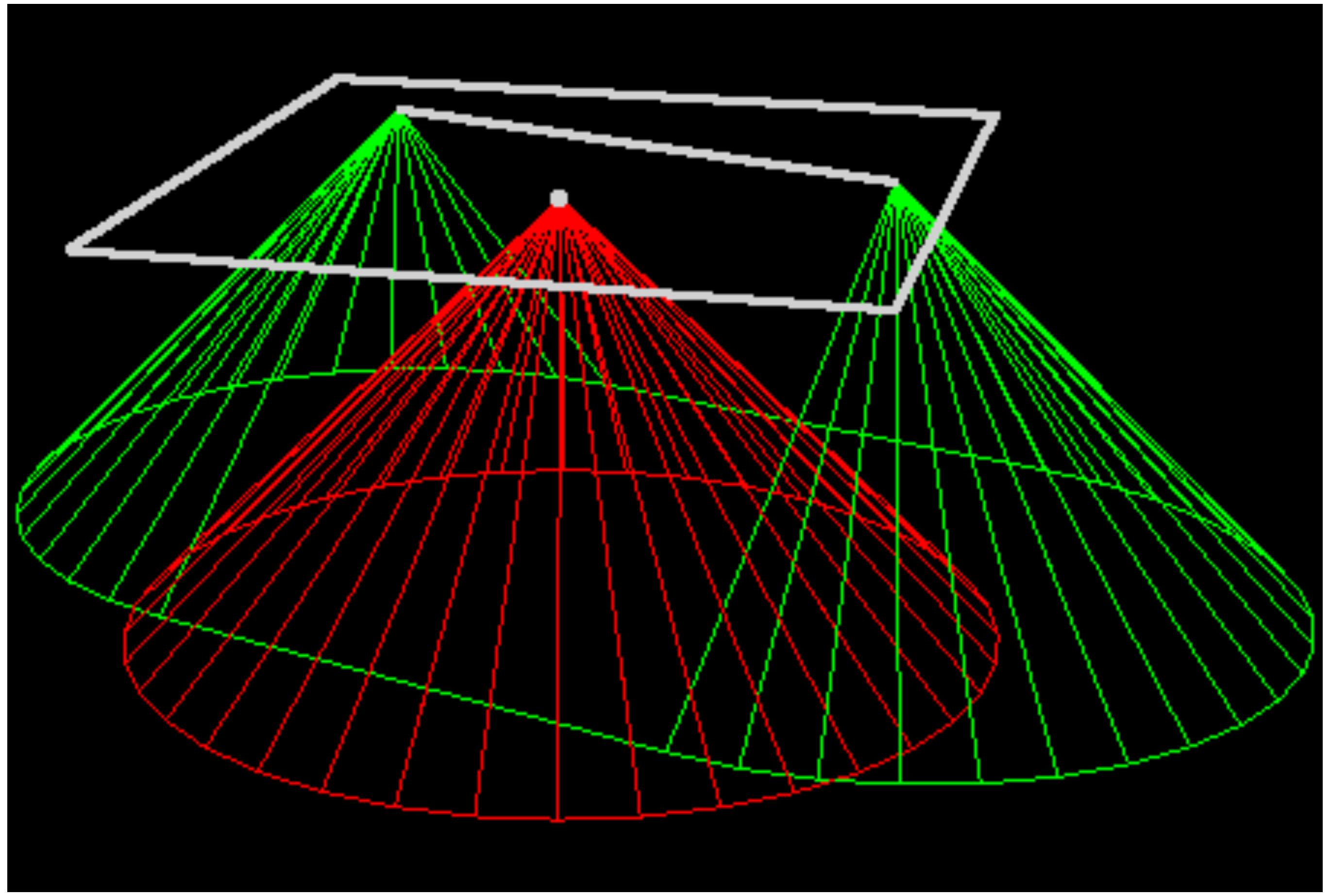
Example with Weighted Sites and Higher-Order Sites



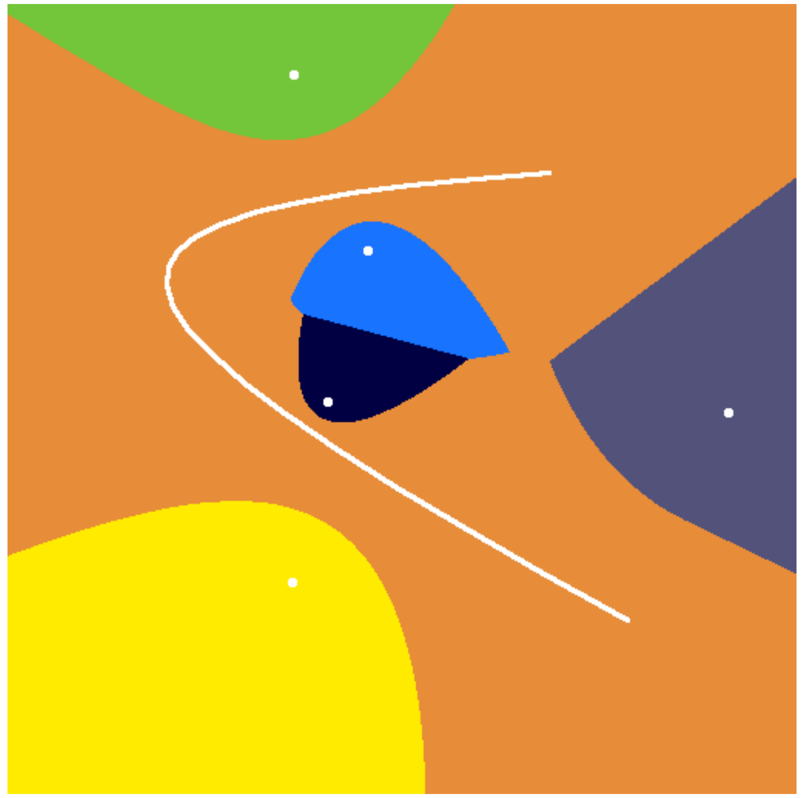
The "Cones Trick" for Higher-Order Sites

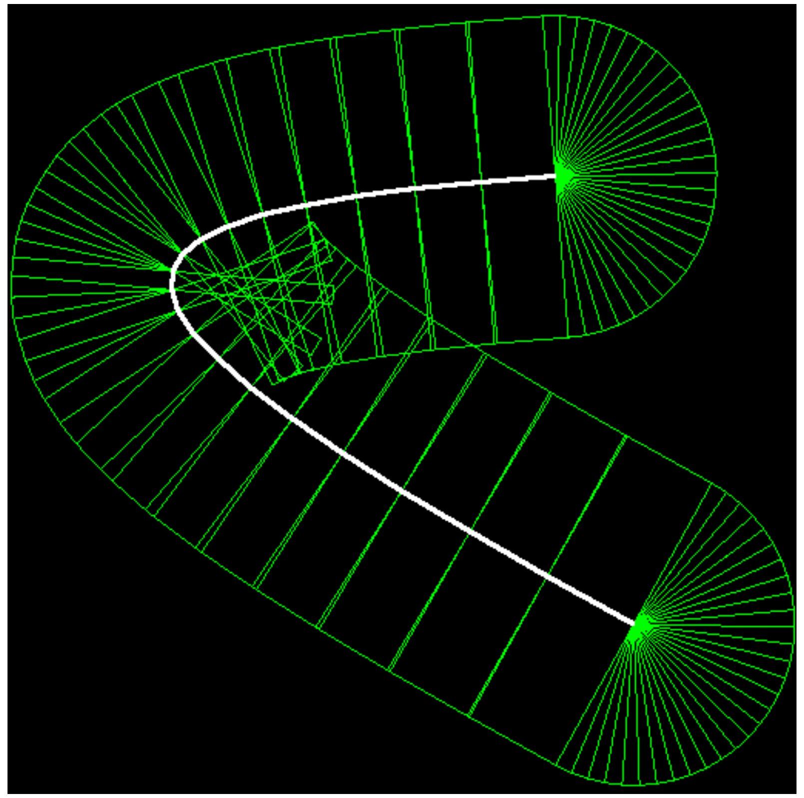
- Observation: the surface in 3D, generated by
$$f(x,y) = (x, y, d(x,y))$$
where $d(x,y)$ = distance from the Voronoi site is a swept cone, where the apex is swept over all points of the generator
- Idea: approximate distance function by a mesh

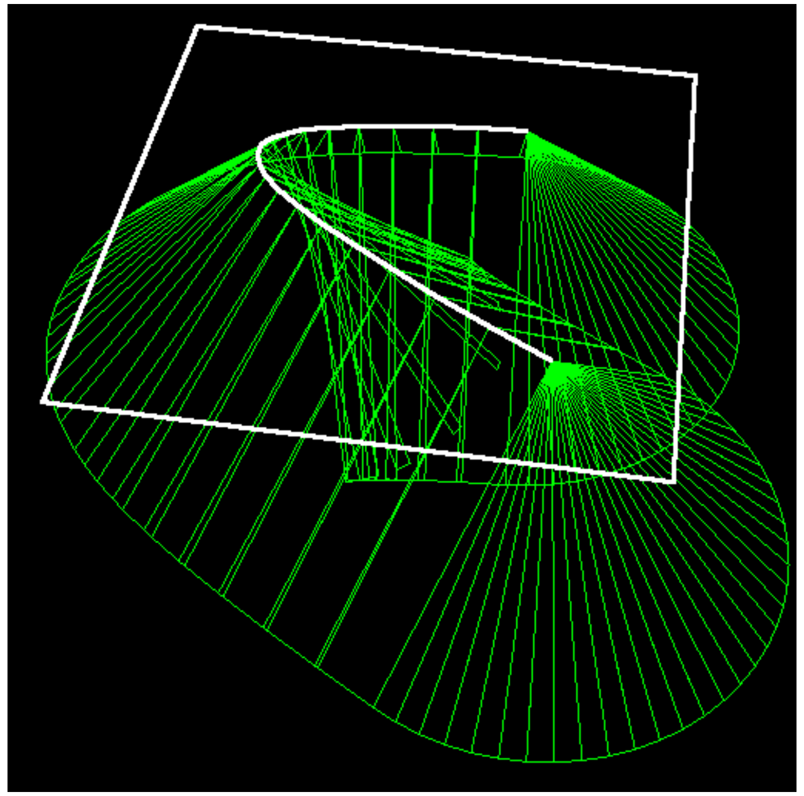


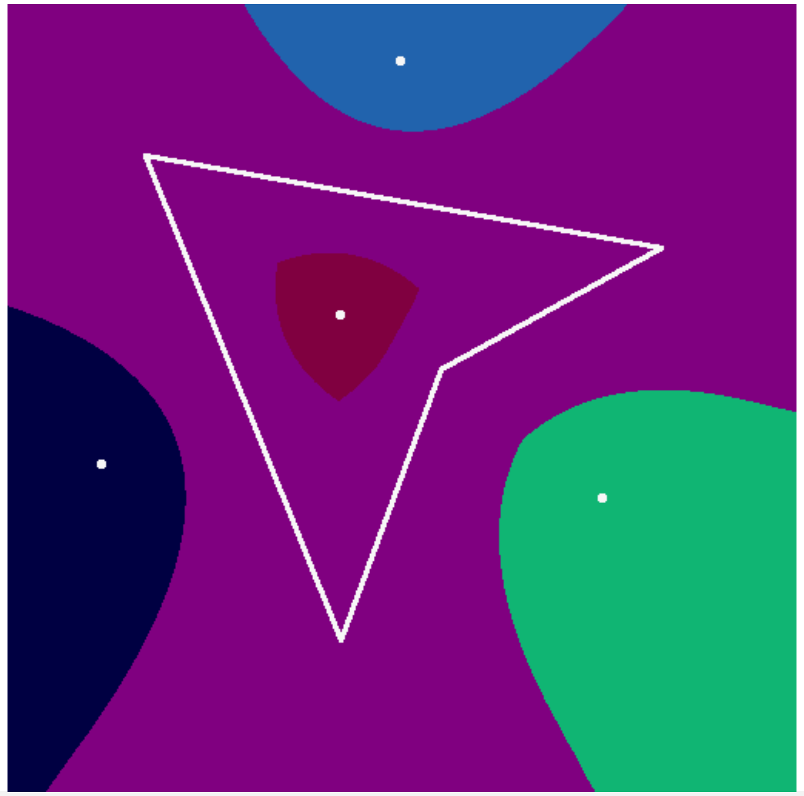


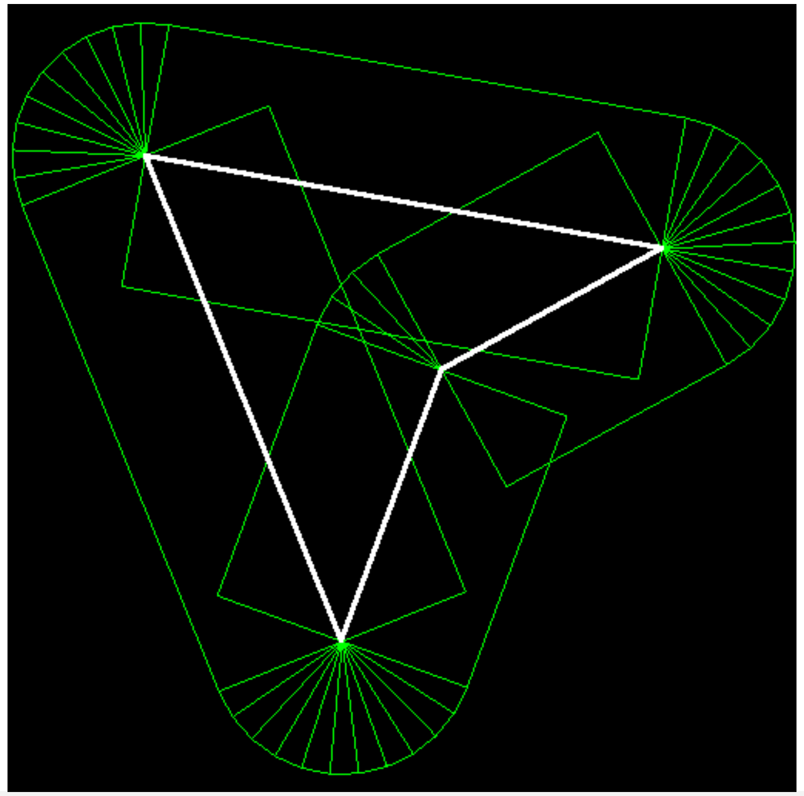
More Example Swept Cones (Distance Meshes)

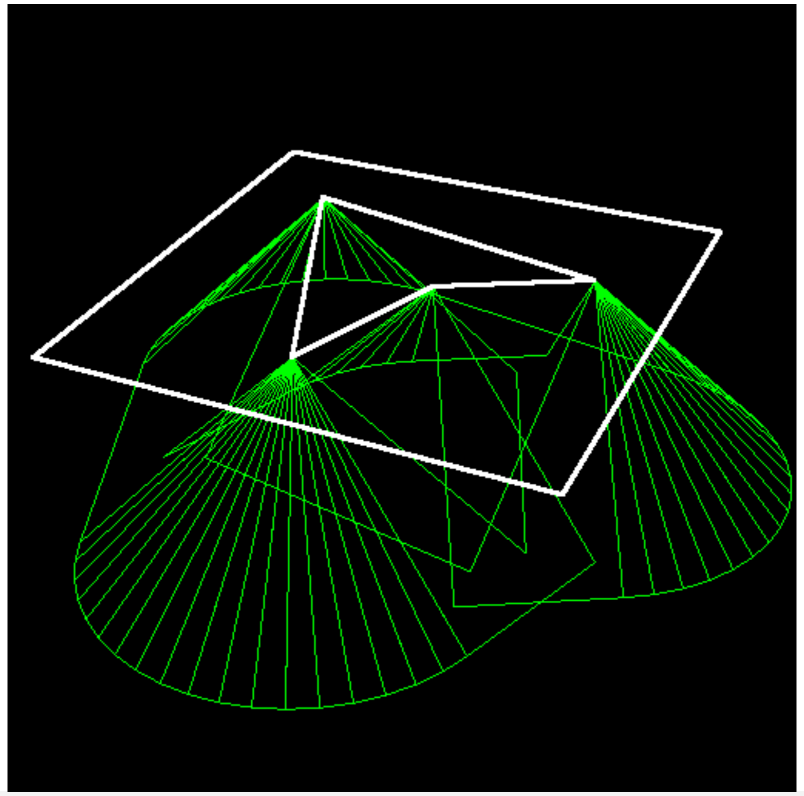




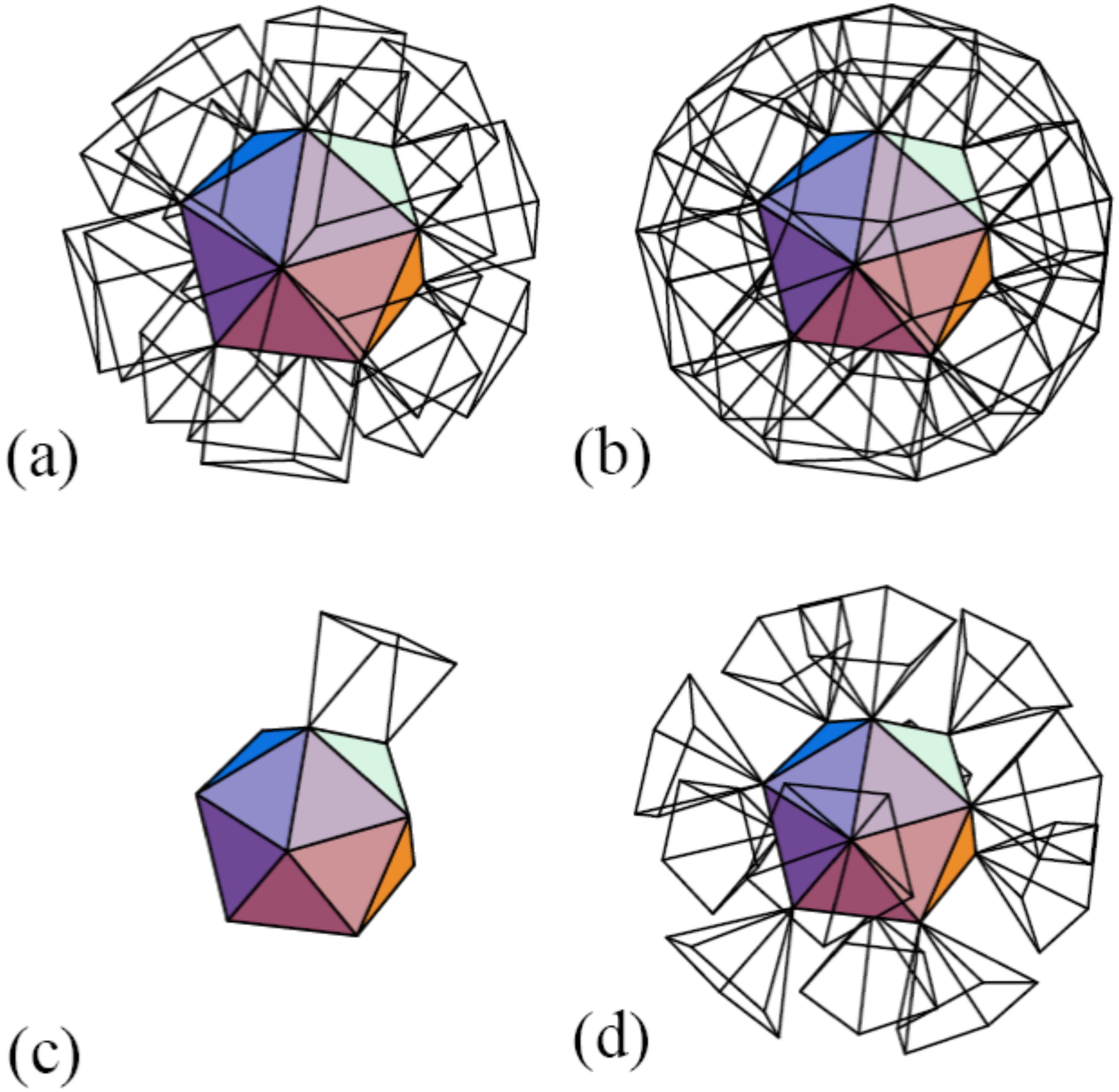








The Outer Voronoi Regions over a Convex Polyhedron



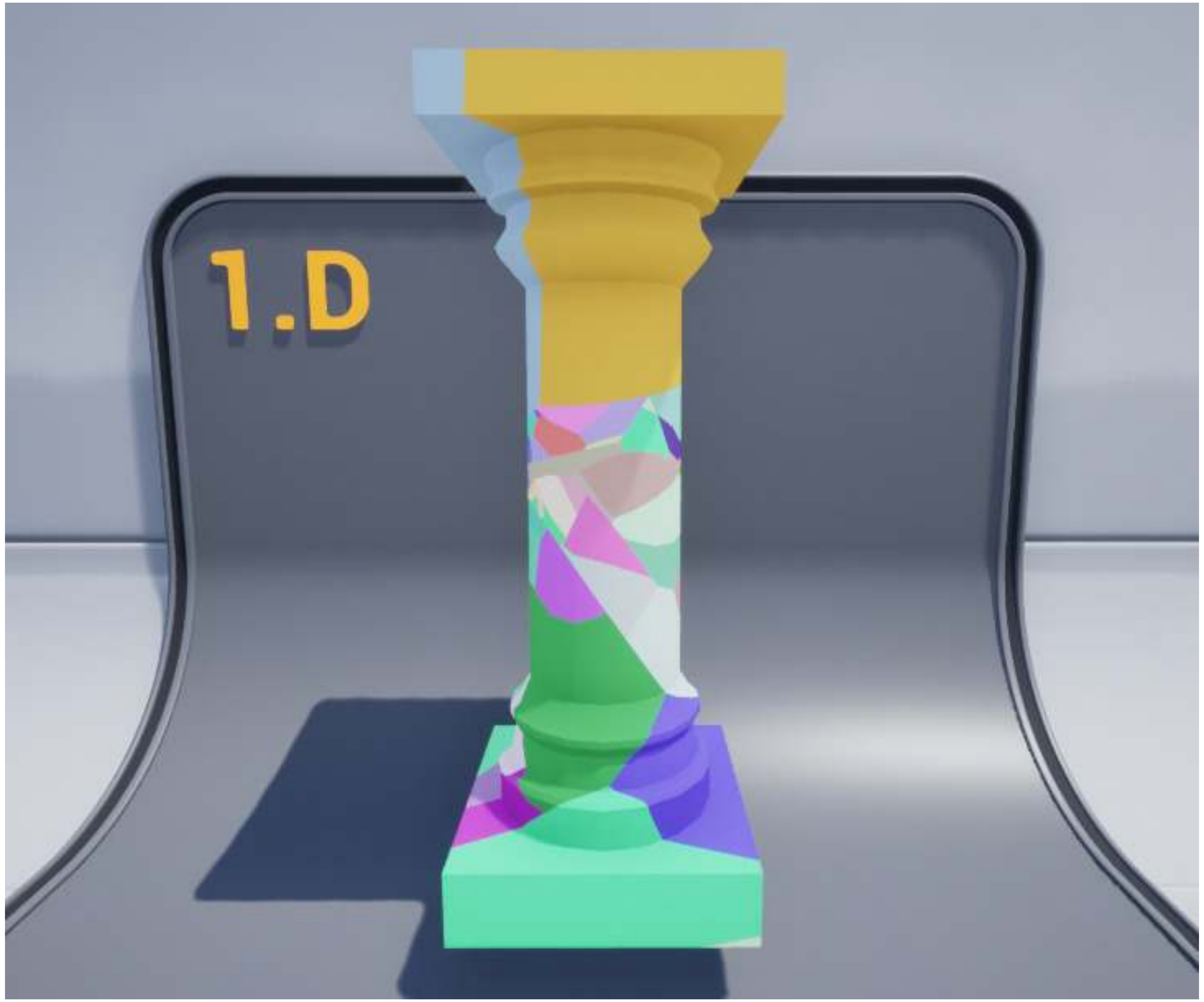
The external
Voronoi regions of …

(a) faces
(b) all features
(c) a single edge
(d) vertices

Application Areas for Voronoi Diagrams

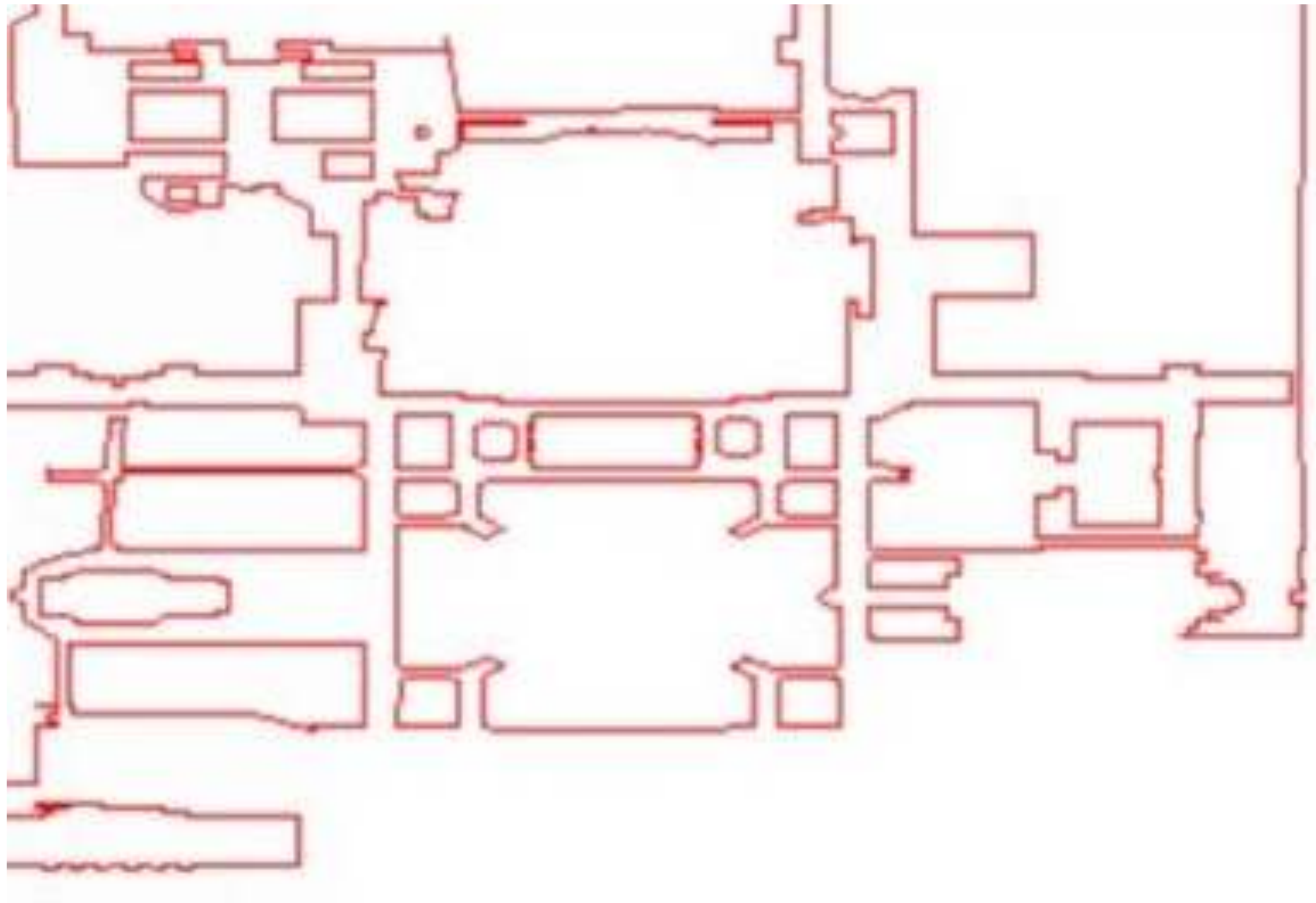
- **Anthropology and Archeology** -- Identify the parts of a region under the influence of different Neolithic clans, chiefdoms, ceremonial centers, or hill forts.
- **Astronomy** -- Identify clusters of stars and clusters of galaxies (Here we saw what may be the earliest picture of a Voronoi diagram, 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

Application: Fracturing (e.g., in Games)



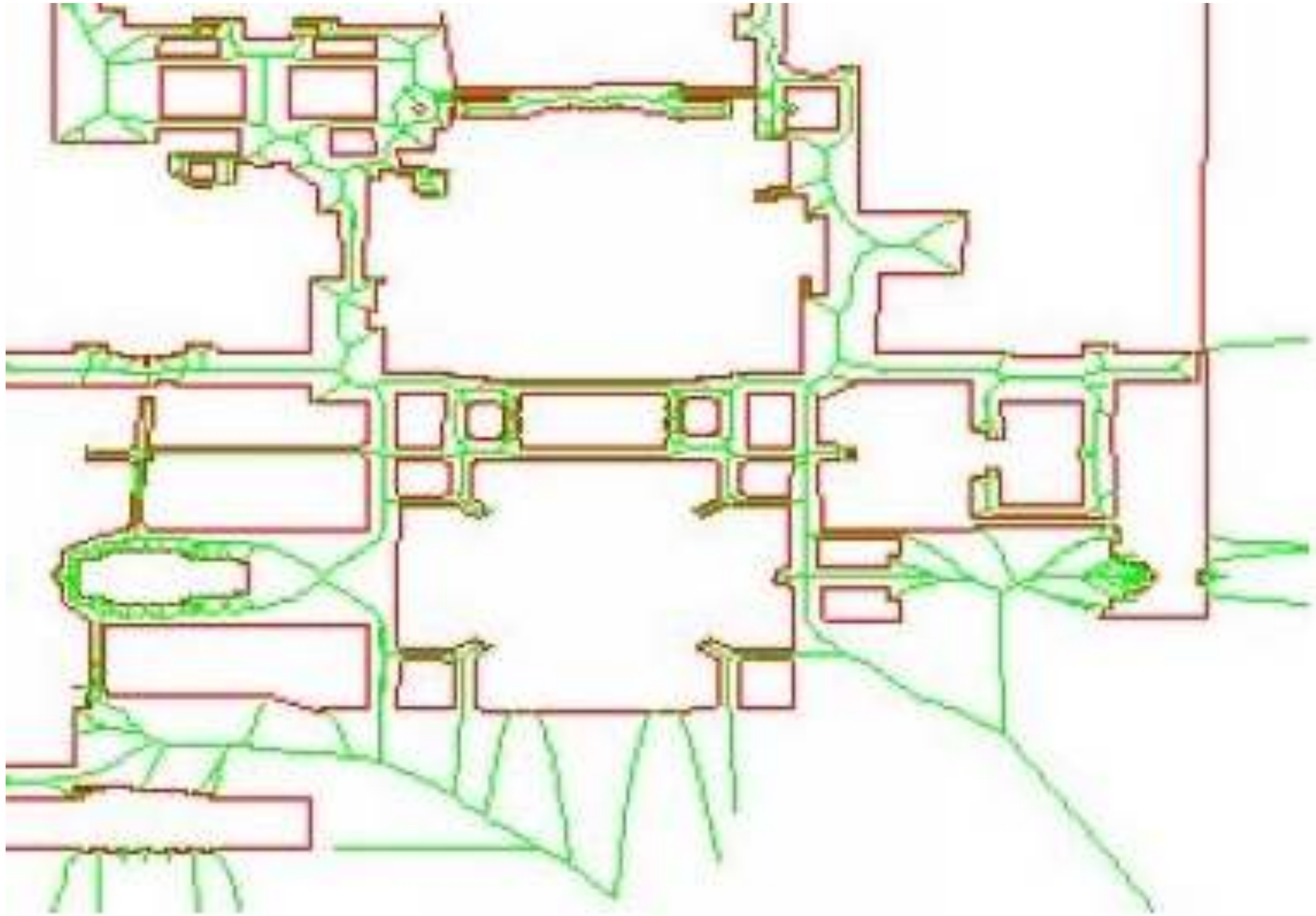
Path Planning

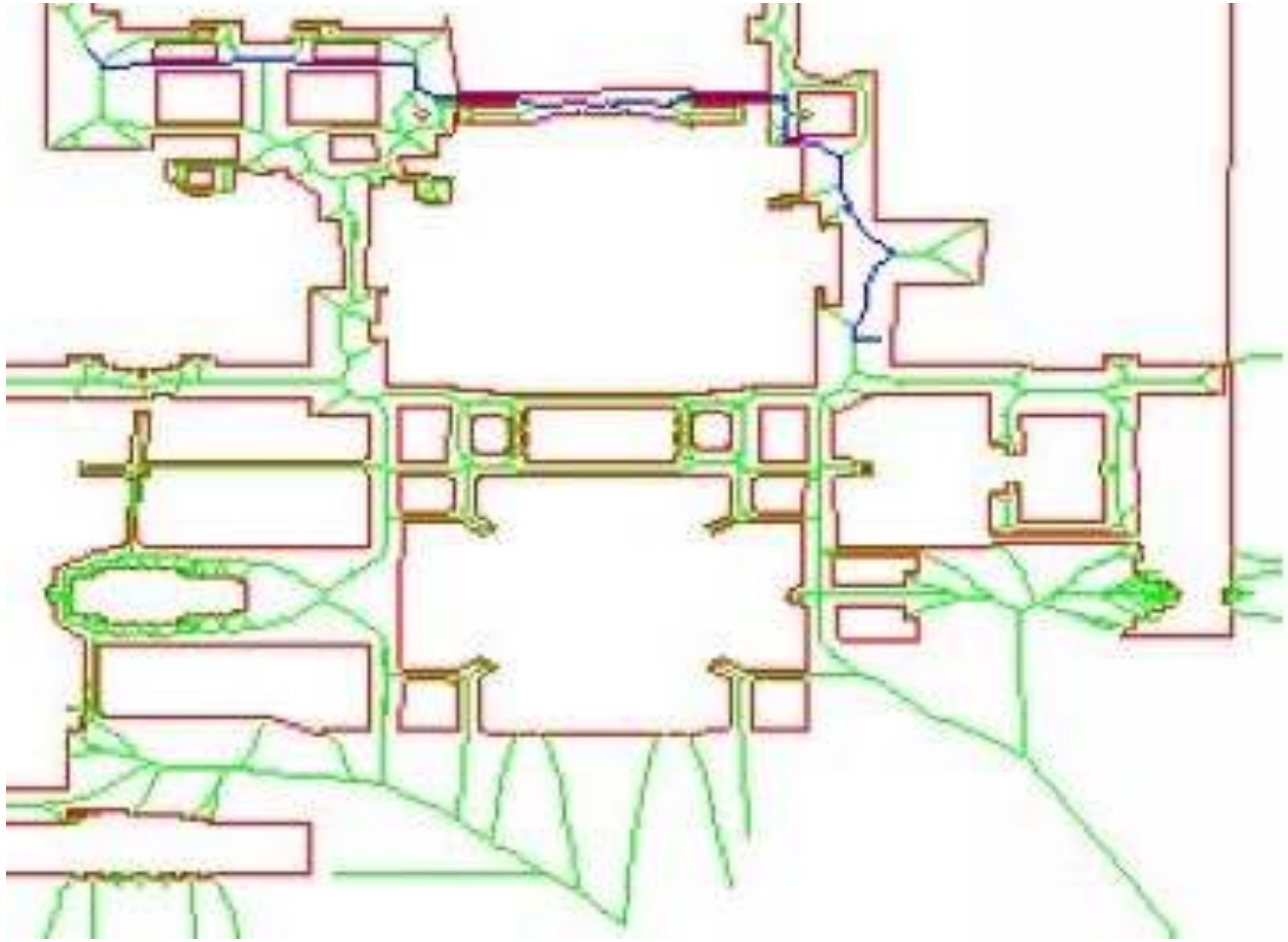
- Given: a floor plan as set of line segments
- Sought: path (e.g. for autonomous vehicle = robot) with maximum distance to walls
- Solution:
  - Construct (generalized) Voronoi diagram
  - Find Voronoi nodes closest to the start and end point, resp.
  - Use Dijkstra's algorithm to find shortest path from start to end nodes through Voronoi diagram



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

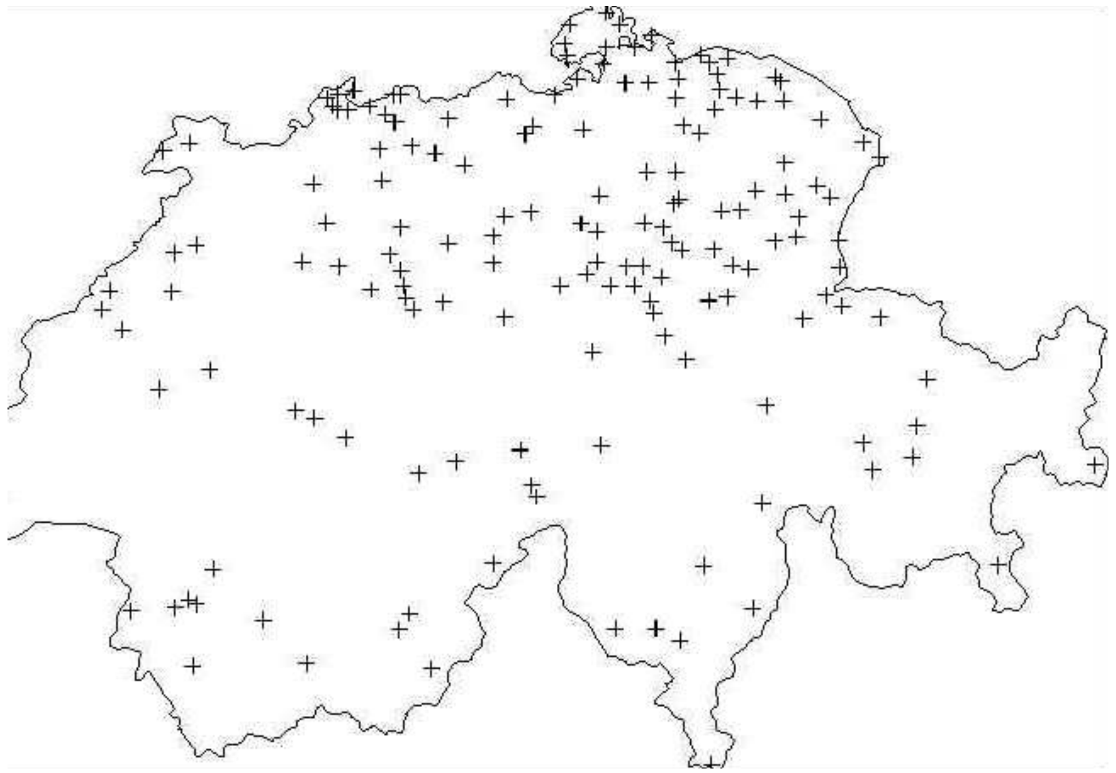
Example

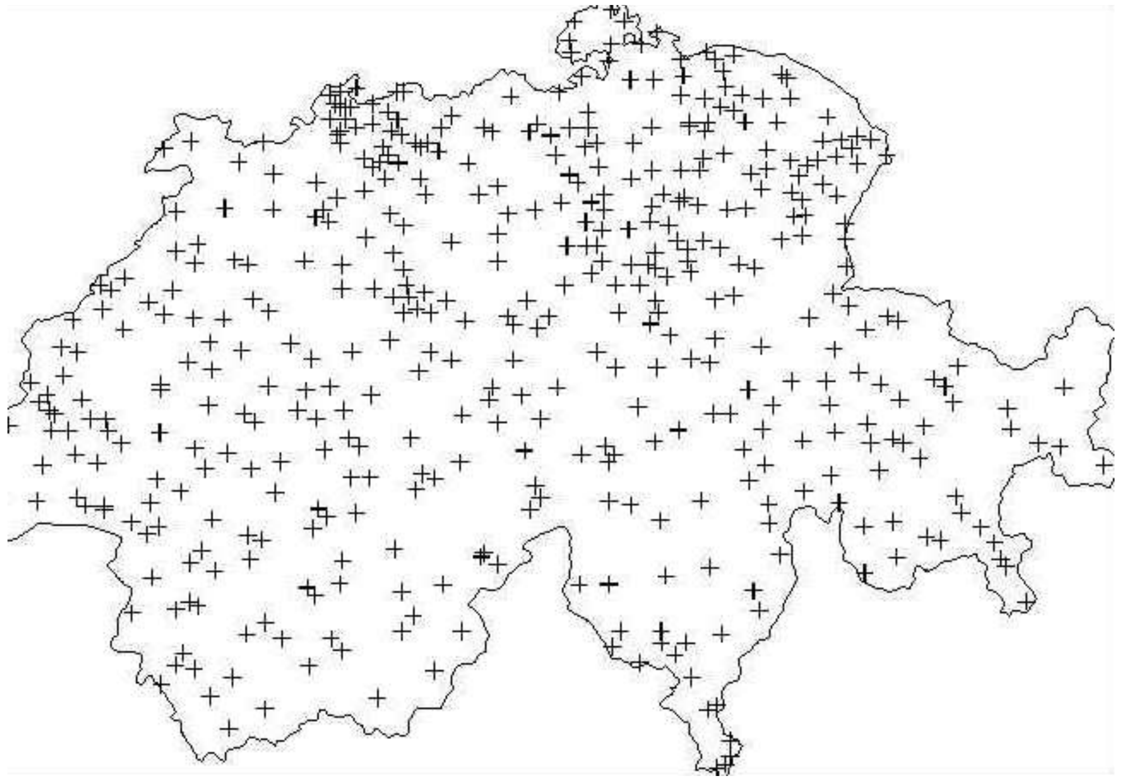




Assessing the Quality of Samplings

- Example: weather stations
- Question: where is the lowest density?
- Ideal sampling → each point would cover an area of
$$\bar{A} = \frac{A}{n}$$
where A = total area
- Usually, there are constraints, e.g., accessibility

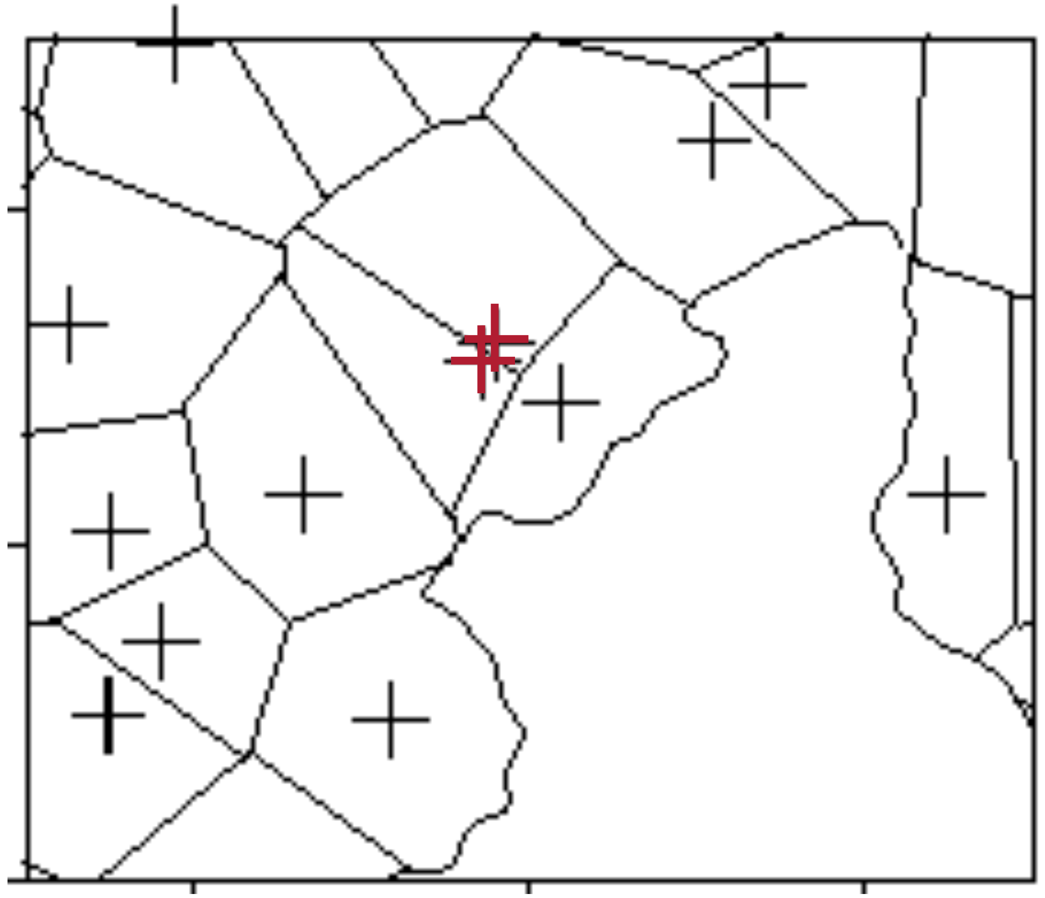


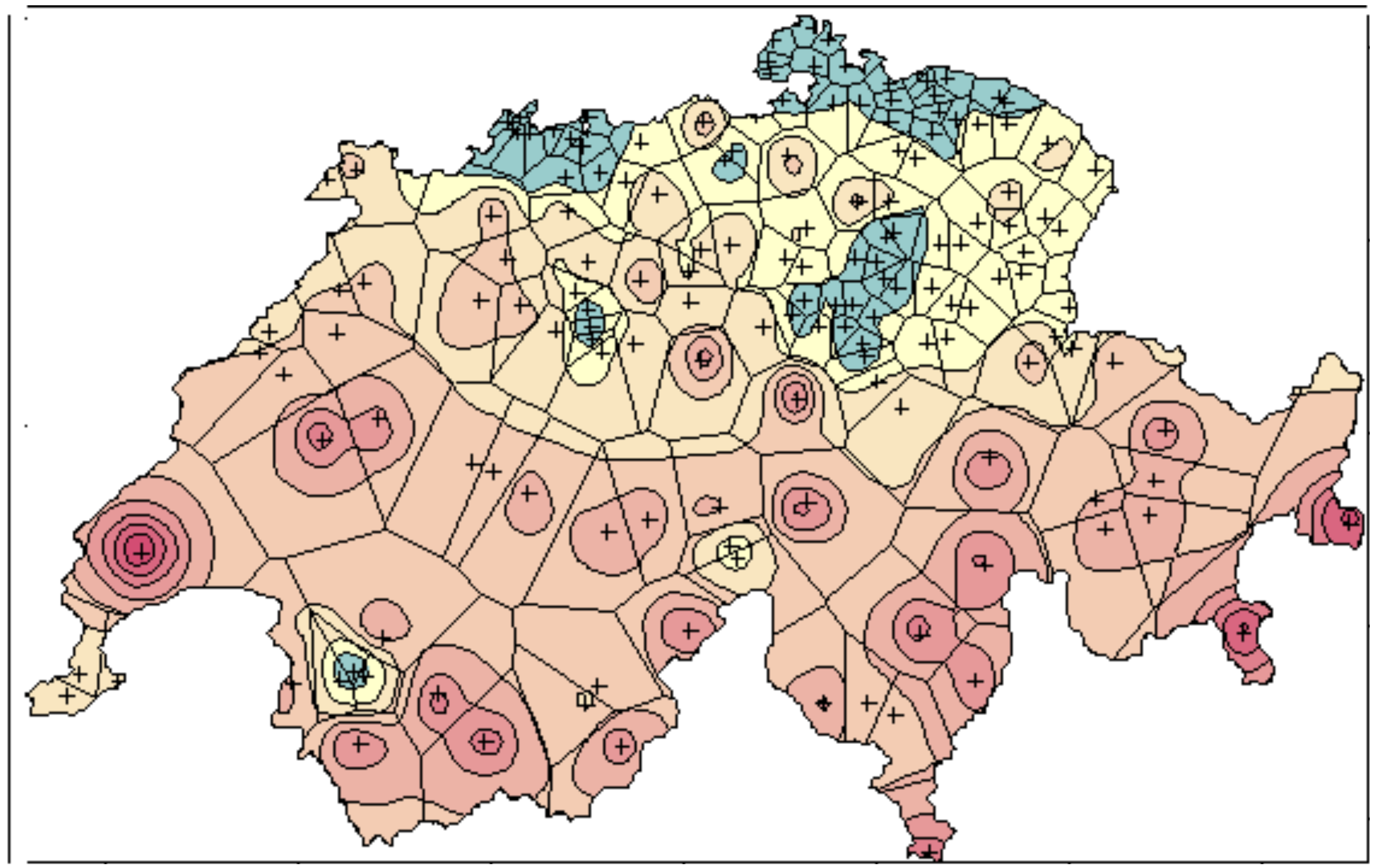


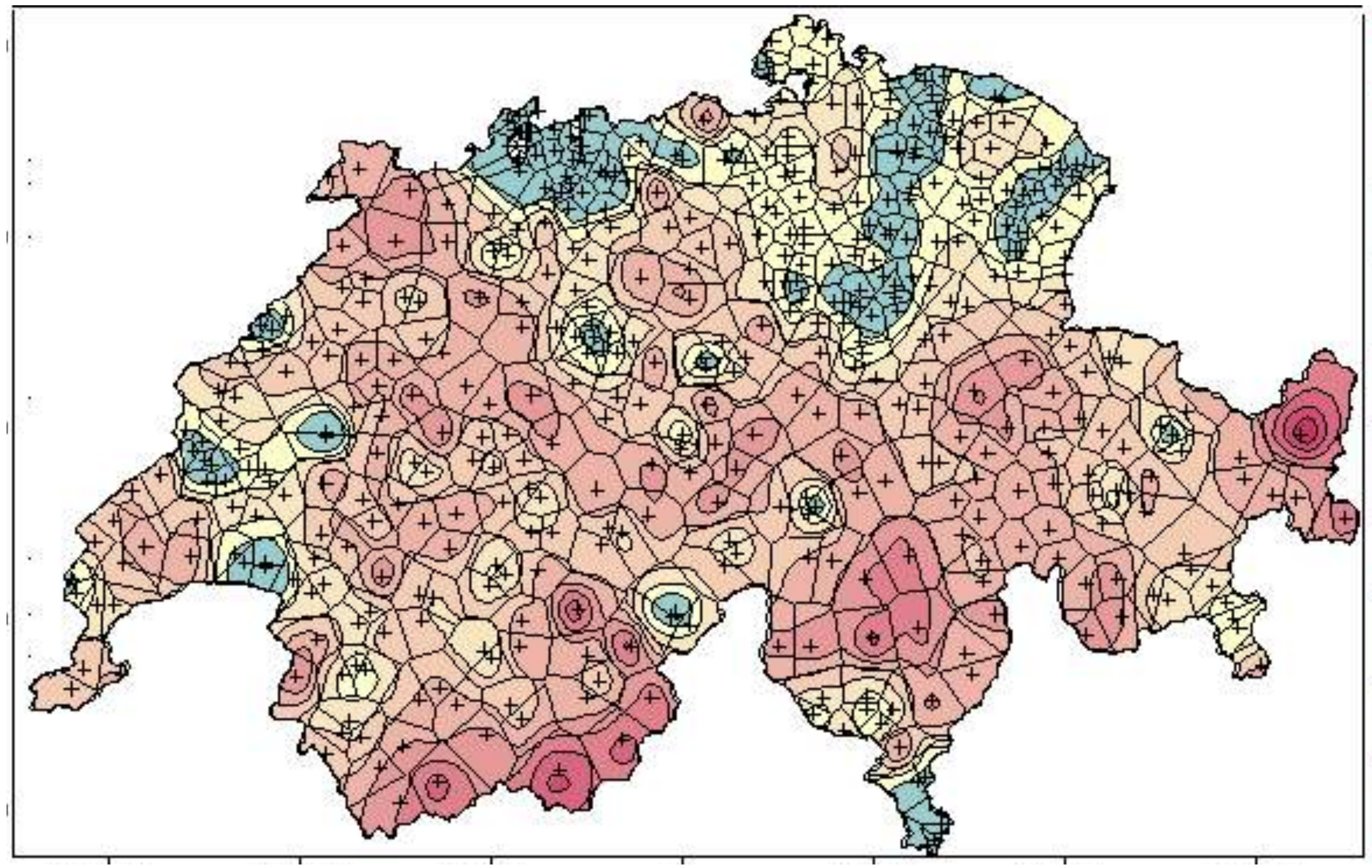
- Solution:
  - Calculate Voronoi and Delaunay diagrams
  - Relative size per cell is

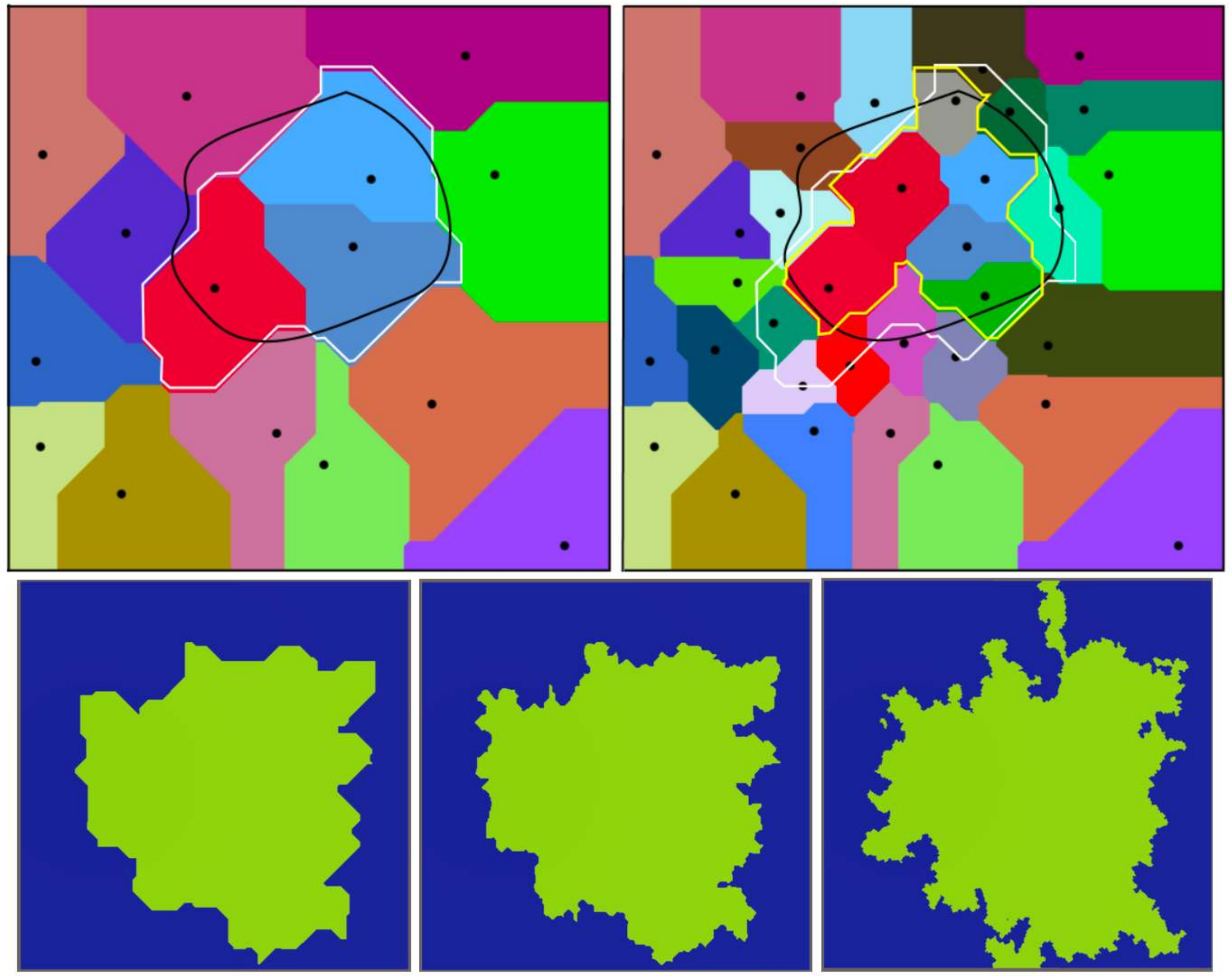
$$A_i = \frac{V_i}{\bar{A}}$$

  - $A_i > 1$ → density too low
  - "Penalize" sample points if they are close together relative to the size of the cell → distance to nearest neighbor





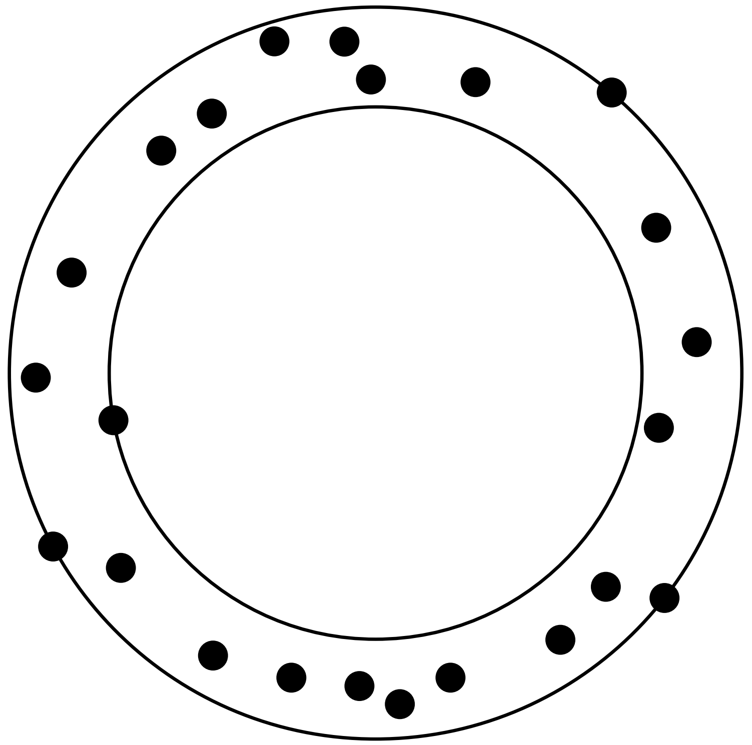




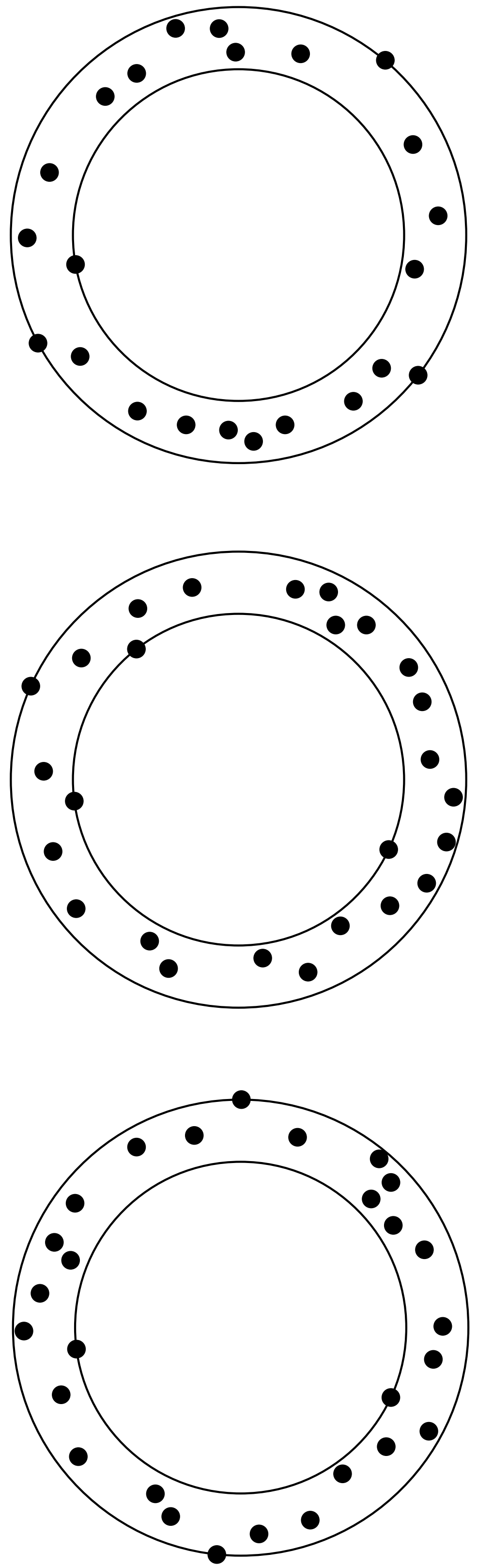
Aus Roland's Diss: Verfeinerung der Küstenlinie

Metrology: Determining the "Sphere-ness" of a Shape

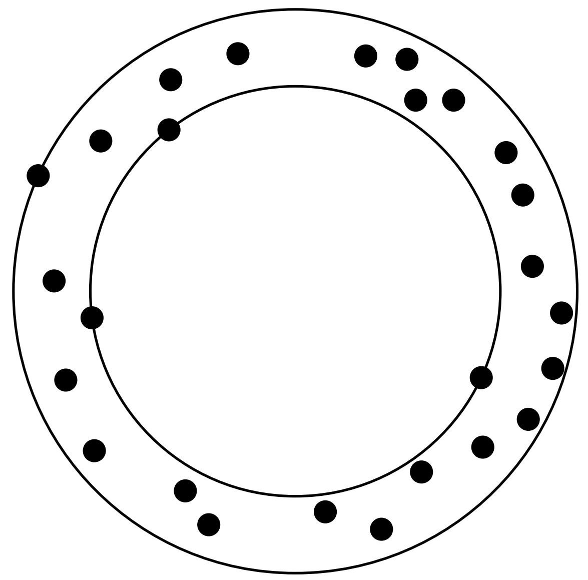
- Application: manufacturing balls for bearings
  - High-precision/high-performance bearings require perfectly spherical balls
- How to determine "sphere-ness"?
- Procedure:
  - Measure coordinates of points on surface
  - Compute smallest annulus containing points S ("smallest" = smallest width)
- Definition annulus:
  region between two concentric spheres (circles)

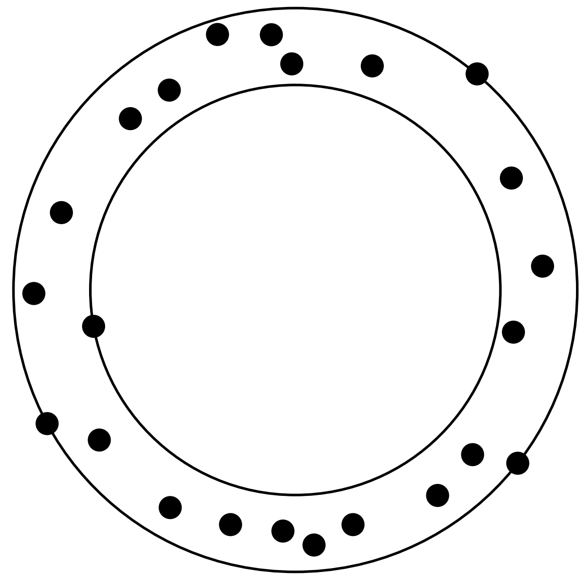


- Cases that can occur in 2D(!):
  1. C_{outer} touches 3 points, C_{inner} touches 1 point
  2. C_{outer} touches 1 points, C_{inner} touches 3 point
  3. C_{outer} touches 2 points, C_{inner} touches 2 points
  - Remember: centers of both spheres are "connected"
  - In all cases, read "*x* points or more"
- In 3D, there are more cases



- Observation: once the center **c** of the annulus is found, the radii follow from S
- Case 2: **c** is closest point to 3 points of $S \longrightarrow$
  c sits on a node of the Voronoi diagram of S
- Case 1: **c** is farthest point to 3 points of $S \longrightarrow$
  c sits on a node of farthest-point Voronoi diagram of S
- Case 3: **c** is *closest* to 2 points of S and *farthest* to 2 points of S $\longrightarrow$
  c sits on an edge of VD and on an edge of farthest VD of S





The Farthest-Point Voronoi Diagram

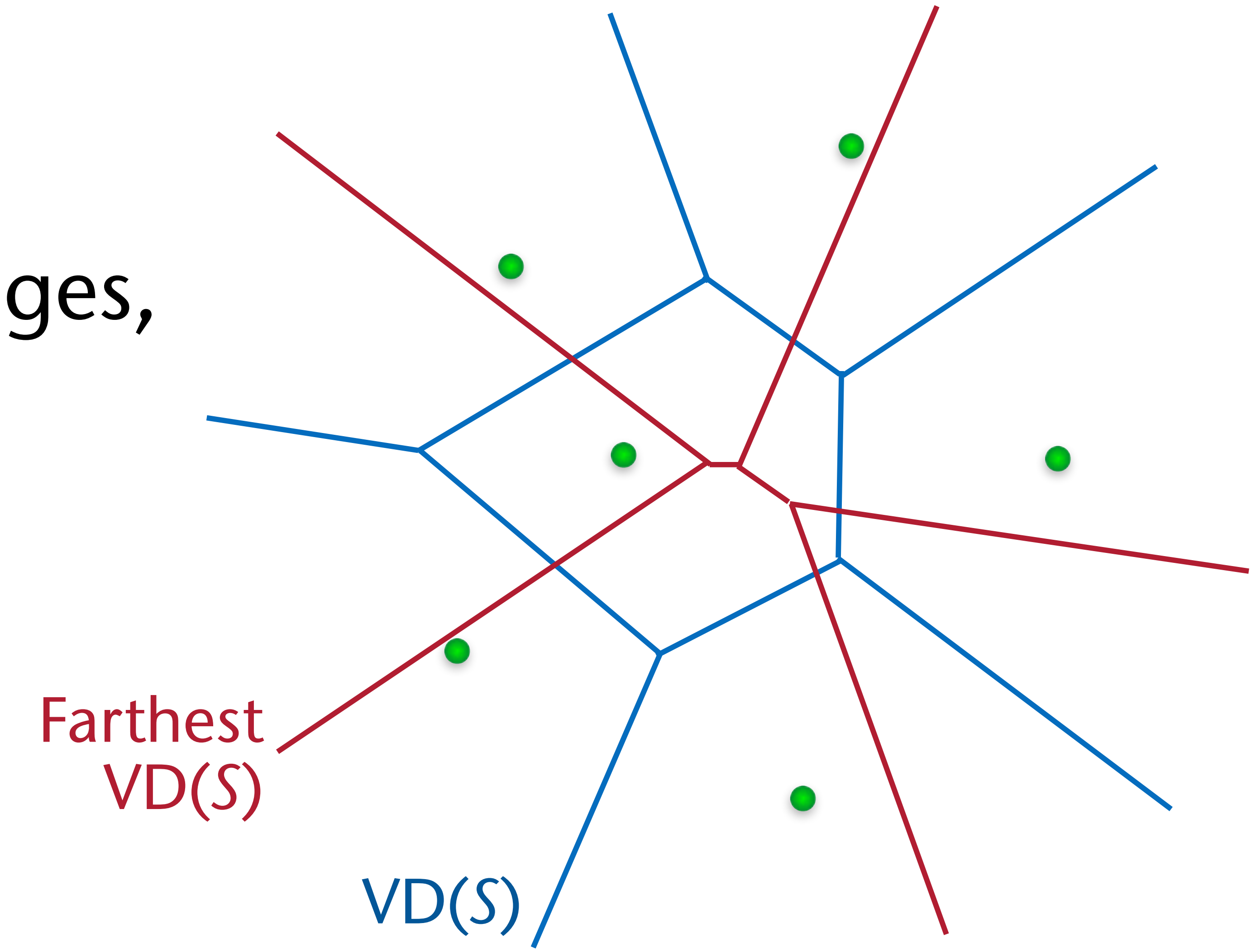
- Just like the VD over *n* points, except ...
- Voronoi region of a point $p \in S$ = intersection of *n*-1 half-spaces where we take the "other" side of the bisectors!
- Thus,

$$R(p) = \{\, x \in \mathbb{R}^d \mid d(x,p) = \max_{q \in S} d(x,q) \,\}$$

- Some properties are similar, some different:
  - Farthest-point Voronoi regions are convex
  - Nodes of the farthest-point VD are farthest away from 3 Voronoi sites/generators (i.e., they have the same and maximal distance from 3 Voronoi sites)
  - Some points $p \in S$ don't have a Voronoi region!

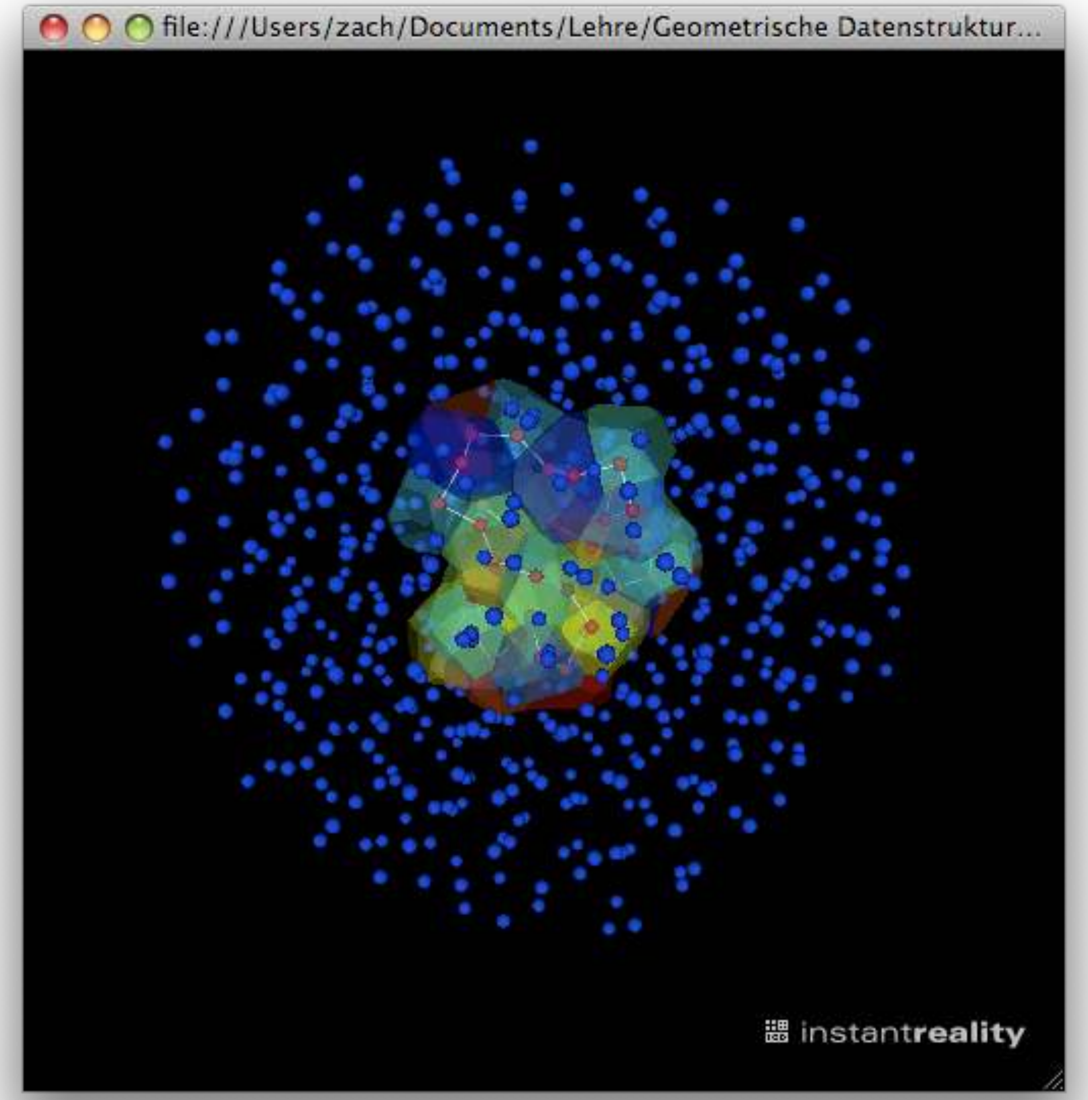
Solution Method to the Problem of Finding the Smallest Annulus

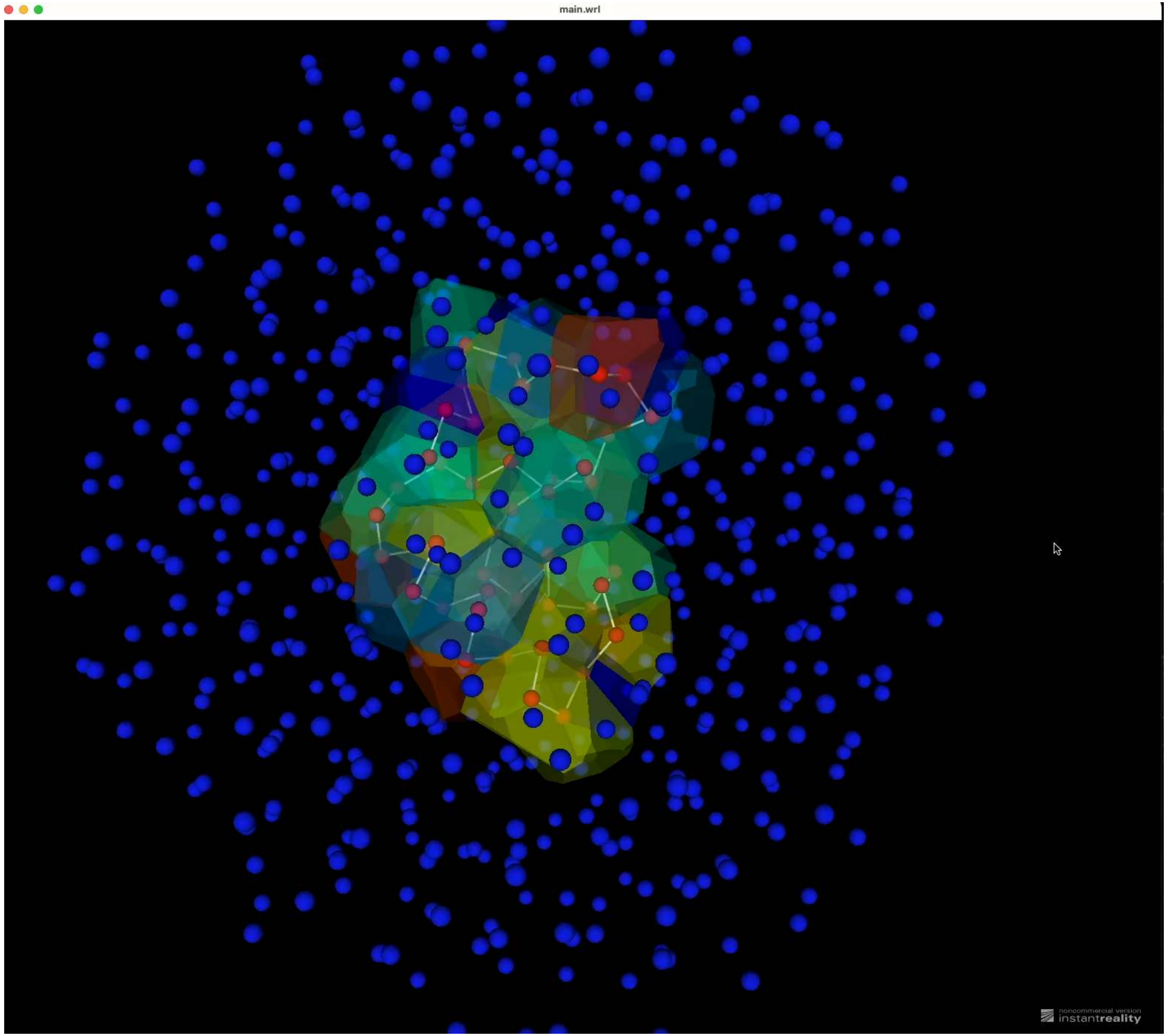
- Compute VD(S), the Voronoi nodes are the candidates of center **c** of C_{inner}, find farthest point of S w.r.t. each **c** $\longrightarrow$ smallest annulus for case 2
- Compute farthest-point VD(S), the Voronoi nodes are the candidates of center **c** of C_{outer}, find closest point of S w.r.t. each **c** $\longrightarrow$ smallest annulus for case 1
- Overlay VD(S) and farthest-point VD(S), compute intersection points of all pairs of edges, each is a candidate **c** for case 3 $\longrightarrow$ pick smallest annulus
- Runtime: $O(\, n + n + n^2 \,)$



Protein Structure Analysis

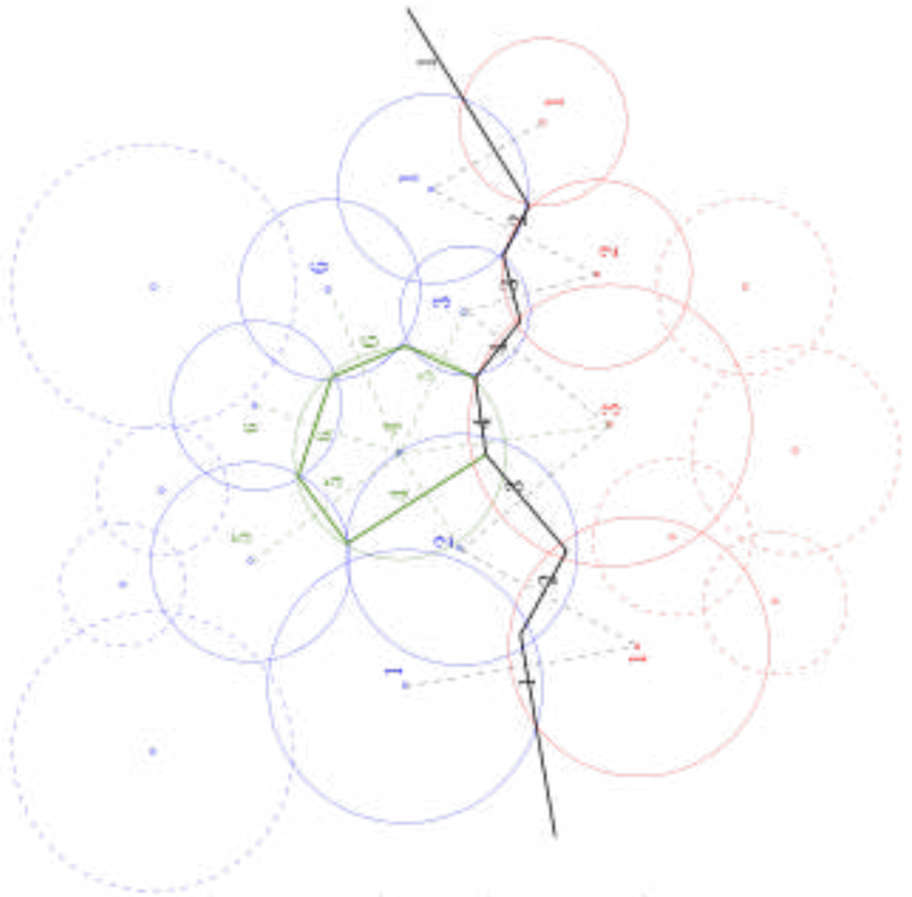
- Question:
  - What does the active surface (= interface) of a molecule look like? how big is it?
  - Which atoms could interact with atoms from the environment?
- One solution:
  - Randomly place atoms around the given molecule
  - Calculate the Voronoi diagram of all points
  - Interface = Voronoi facets between molecule and surrounding atoms

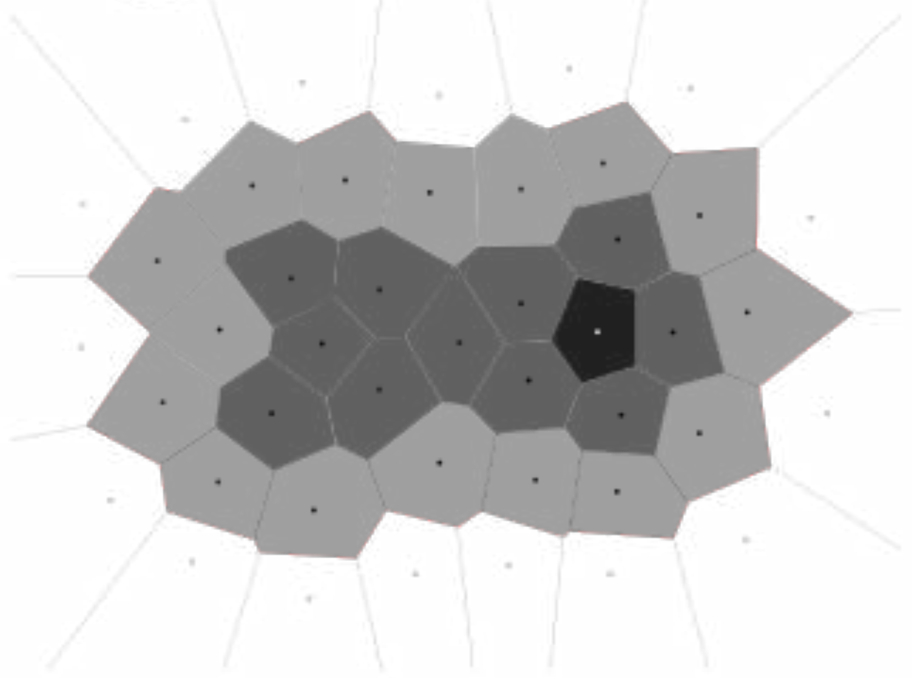



main.wrl
noncommercial version
instantreality

Improvements

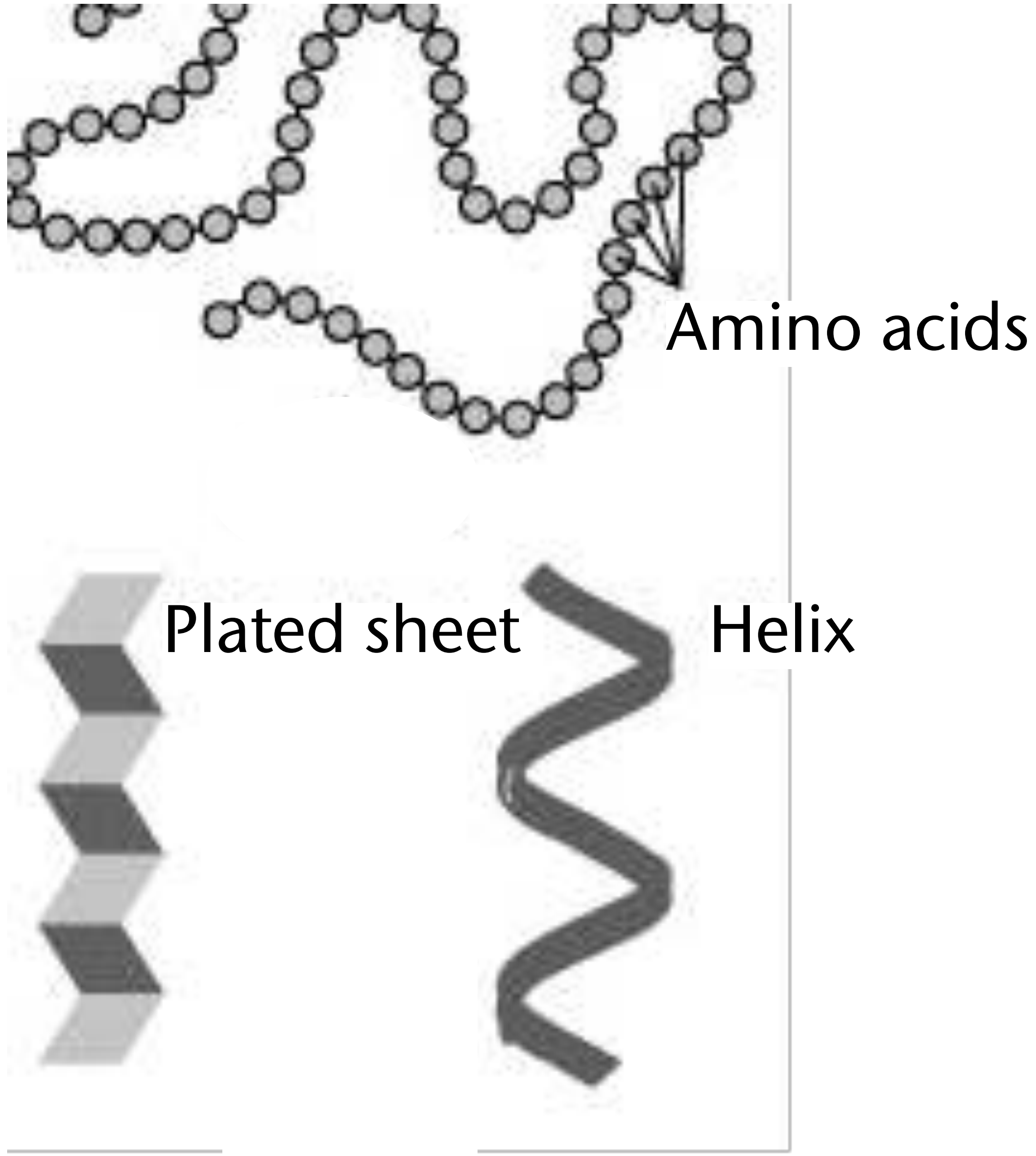
- Use power diagram or Voronoi diagram with additive weights
  - Weight = atomic radius
- Calculate "depth" per atom:
  - Atoms with a Voronoi facet outward = depth 1
  - Traverse Delaunay graph breadth-first from outside to inside
  - The deeper an atom, the smaller its contribution to interactions



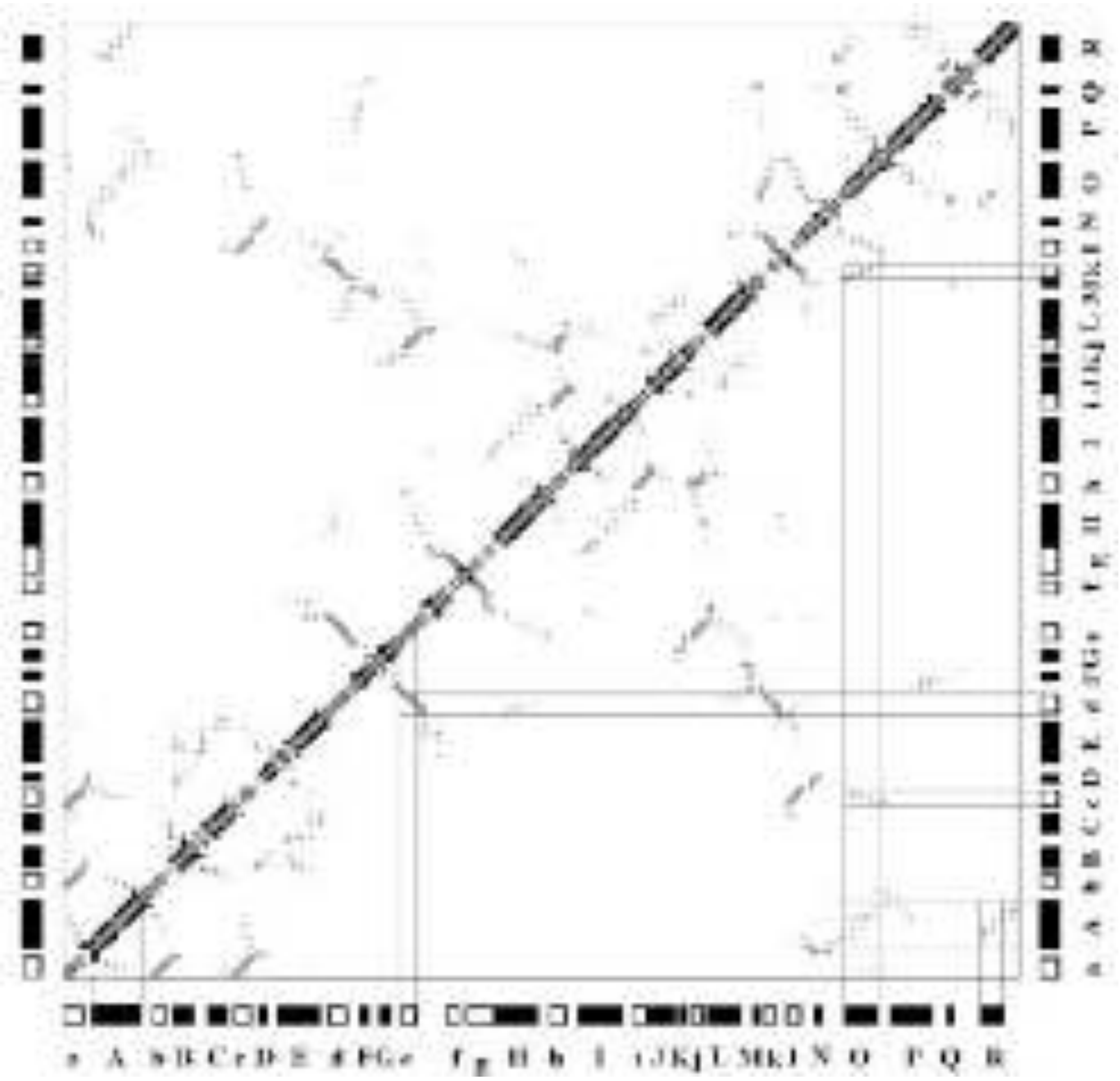


Secondary Structure of Proteins

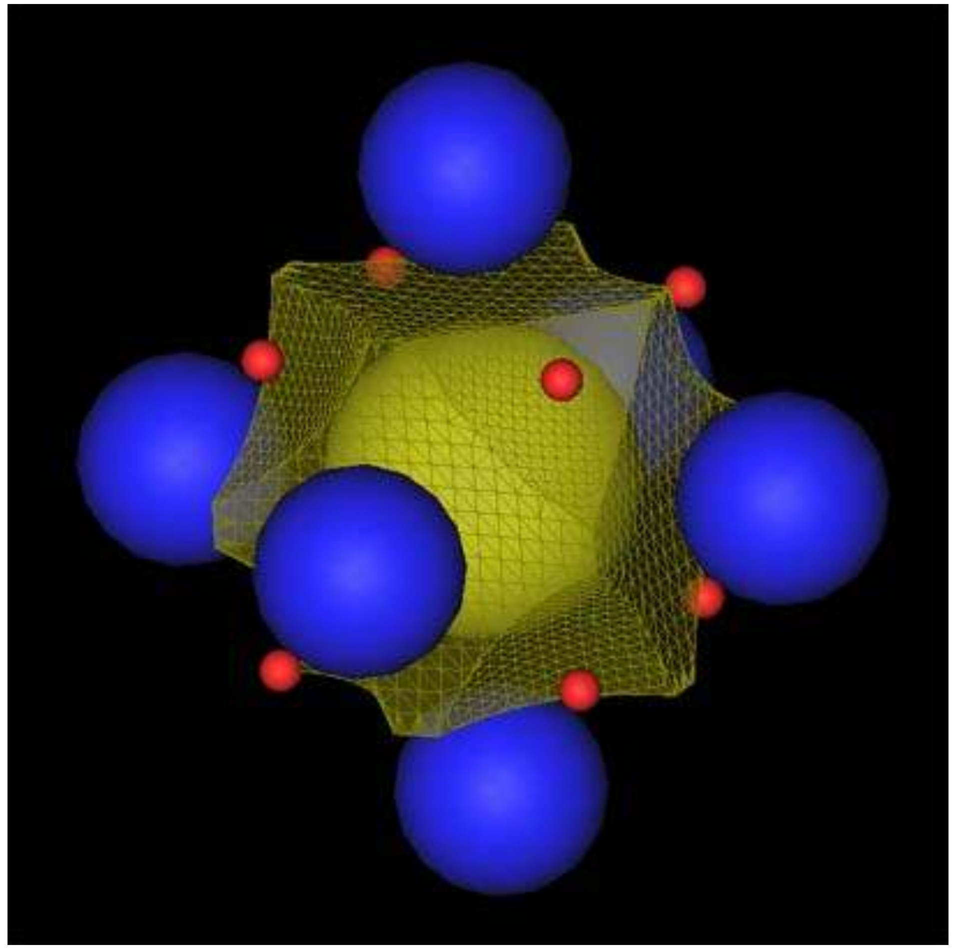
- Long proteins fold into helices, tangles, and surface pieces
- Results in interactions between atoms (bonds) that are not seen in the chemical formula
- Question: given the positions of the atoms, what does the secondary structure look like?
  - Which atoms are "adjacent", which are not
  - How strong is their adjacency?
- Solution: Voronoi diagram
  - Adjacent = common Voronoi facet
  - Strength of the neighborhood = size of the facet

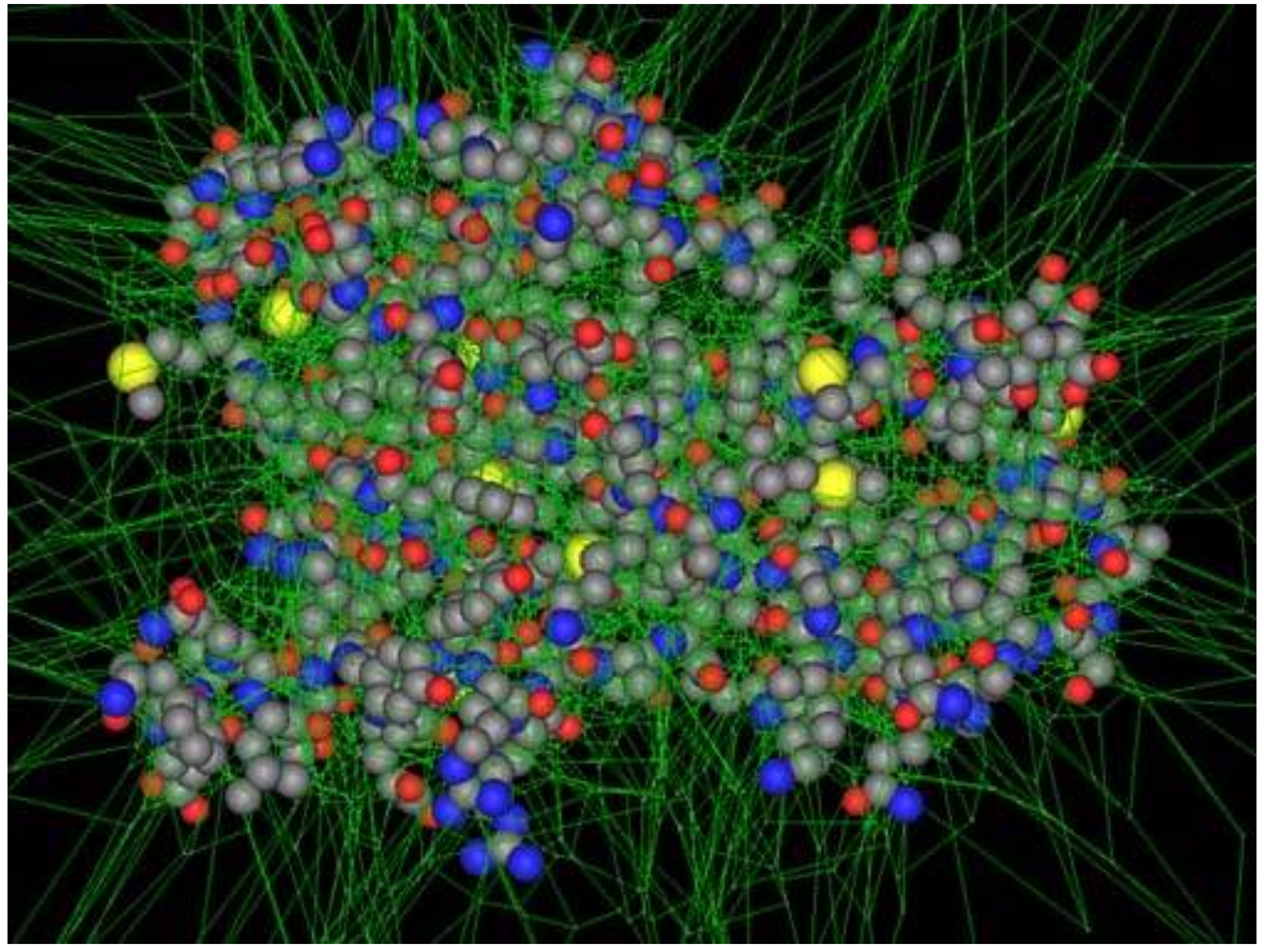


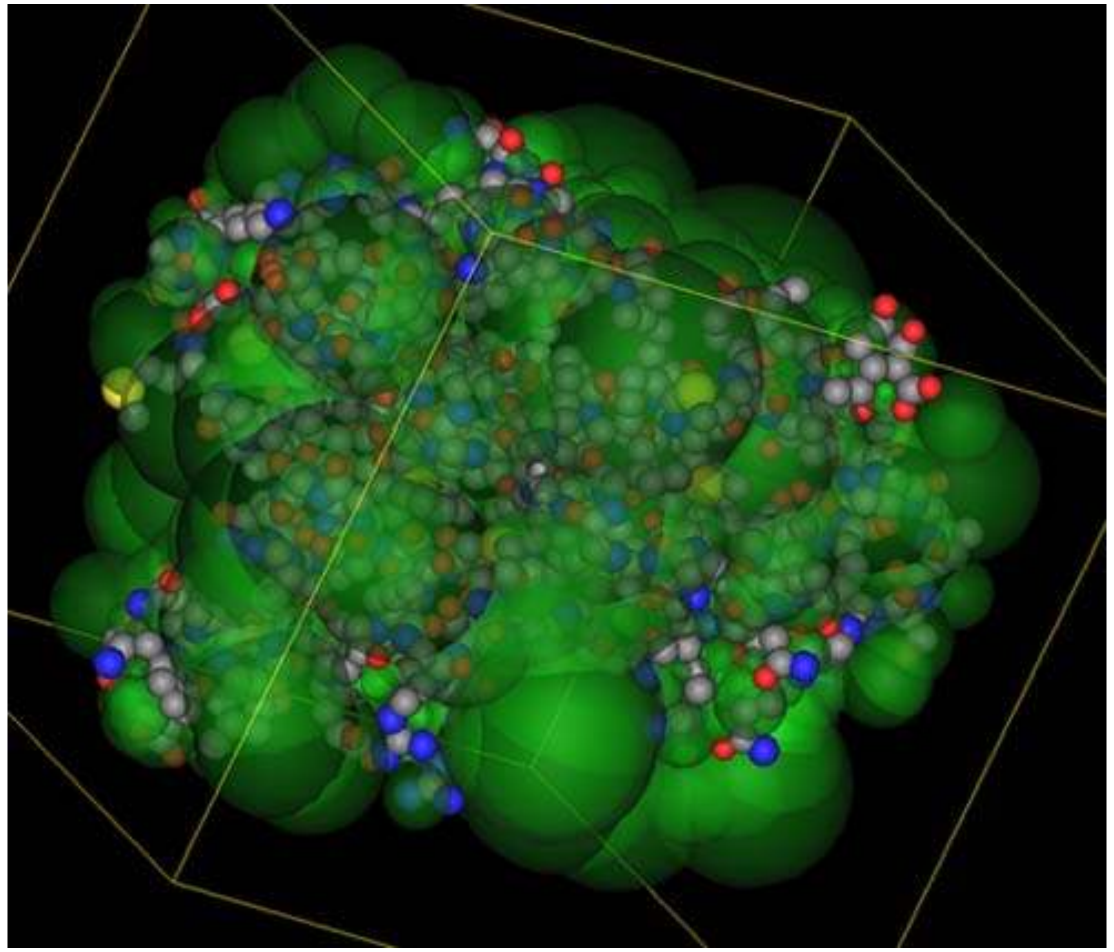
- Result: Adjacency-Matrix (gray/black = weakly/strongly neighboured)



Appolonius Diagrams in 3D



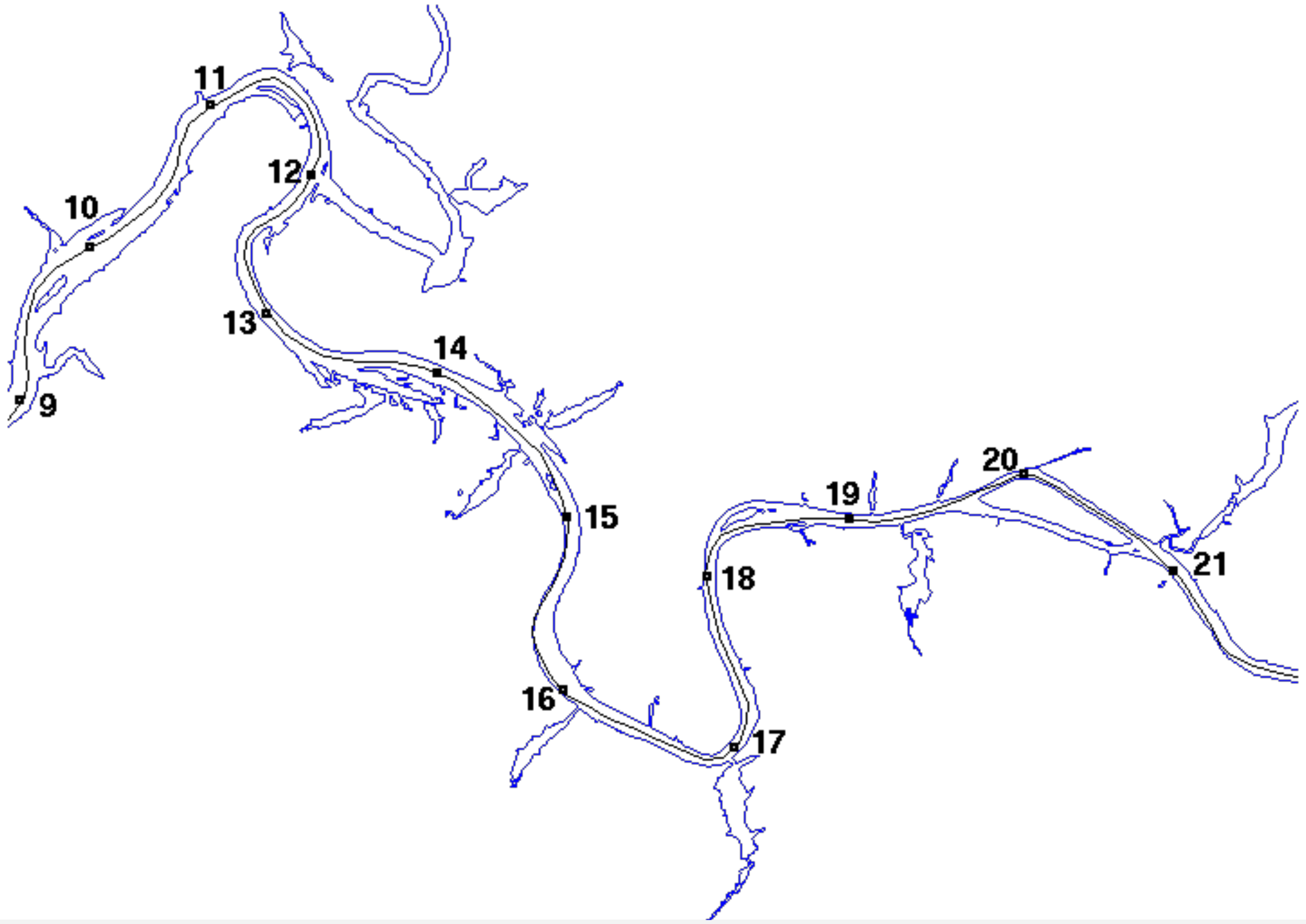




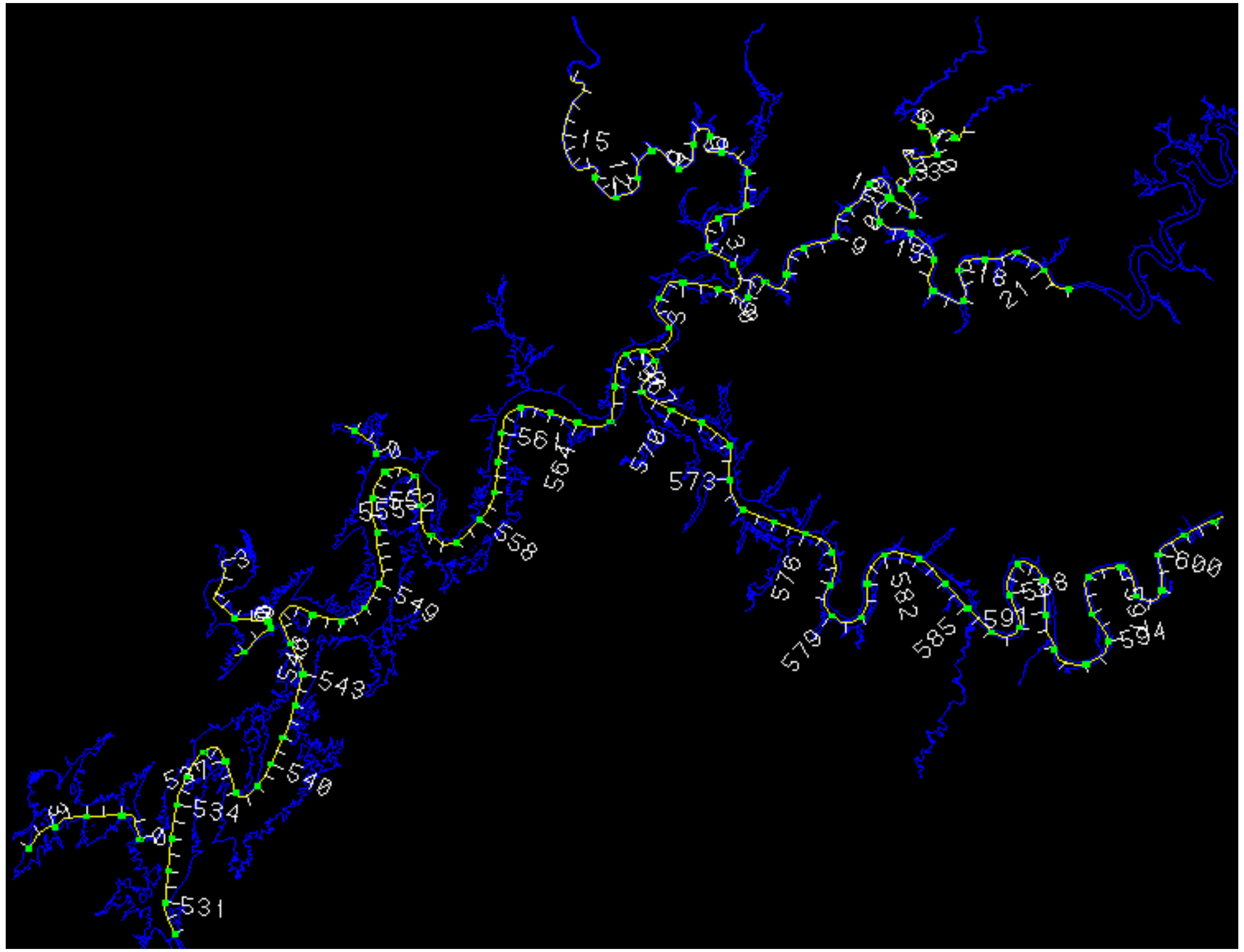
Helps to determine the empty spaces in a molecule

Application: the River-Mile-Coordinate System

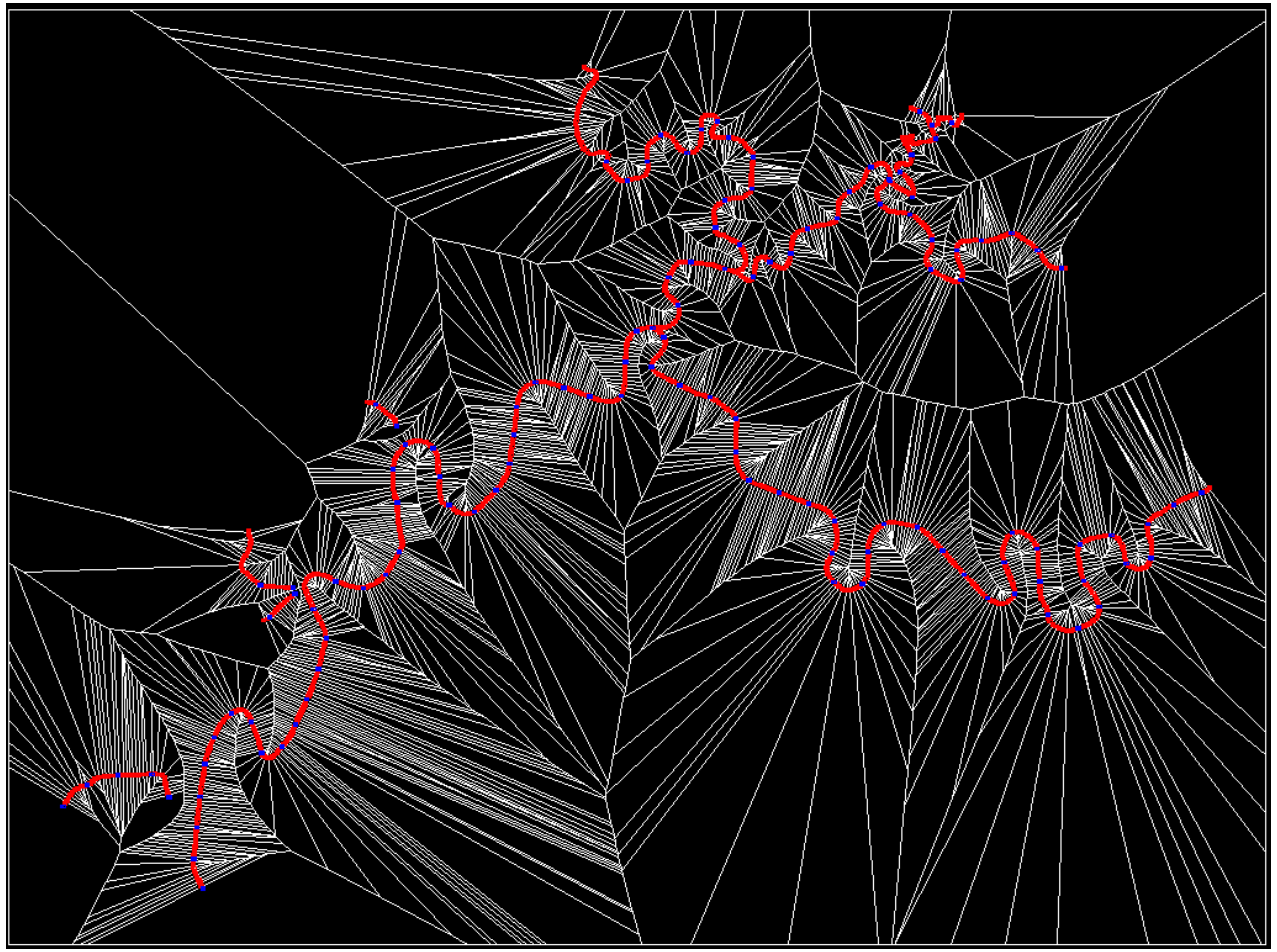
- The River-Mile-Coordinate system:
  - Popularly used in large waterway systems
  - Coordinates of a point in the plane = (l, q) where
    l = measured along a rivers center line,
    q = distance from point $(l, 0)$ perpendicular to the tangent in $(l, 0)$
  - Property: coords reflect how much time it takes to get there along the river
- Task:
  given a point $(x,y) \longrightarrow$
  which coordinate does (l, q) have?



Decomposition of the center line into a finely resolved polygon course

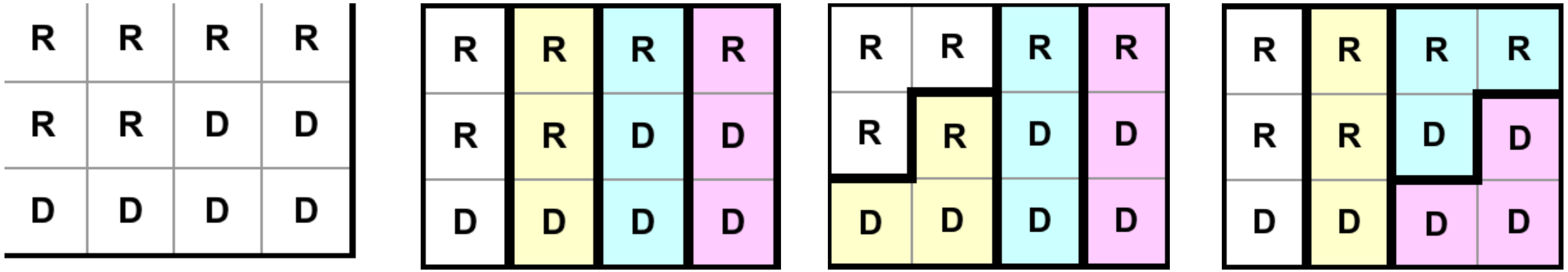


Voronoi diagram for this



Redistricting (Partitioning a Country into Electoral Districts)

- The fairness principle says: "one man, one vote"
  - Simple … or is it?
- A simple example:



Distribution of voters

→ 2x R, 2x D representatives

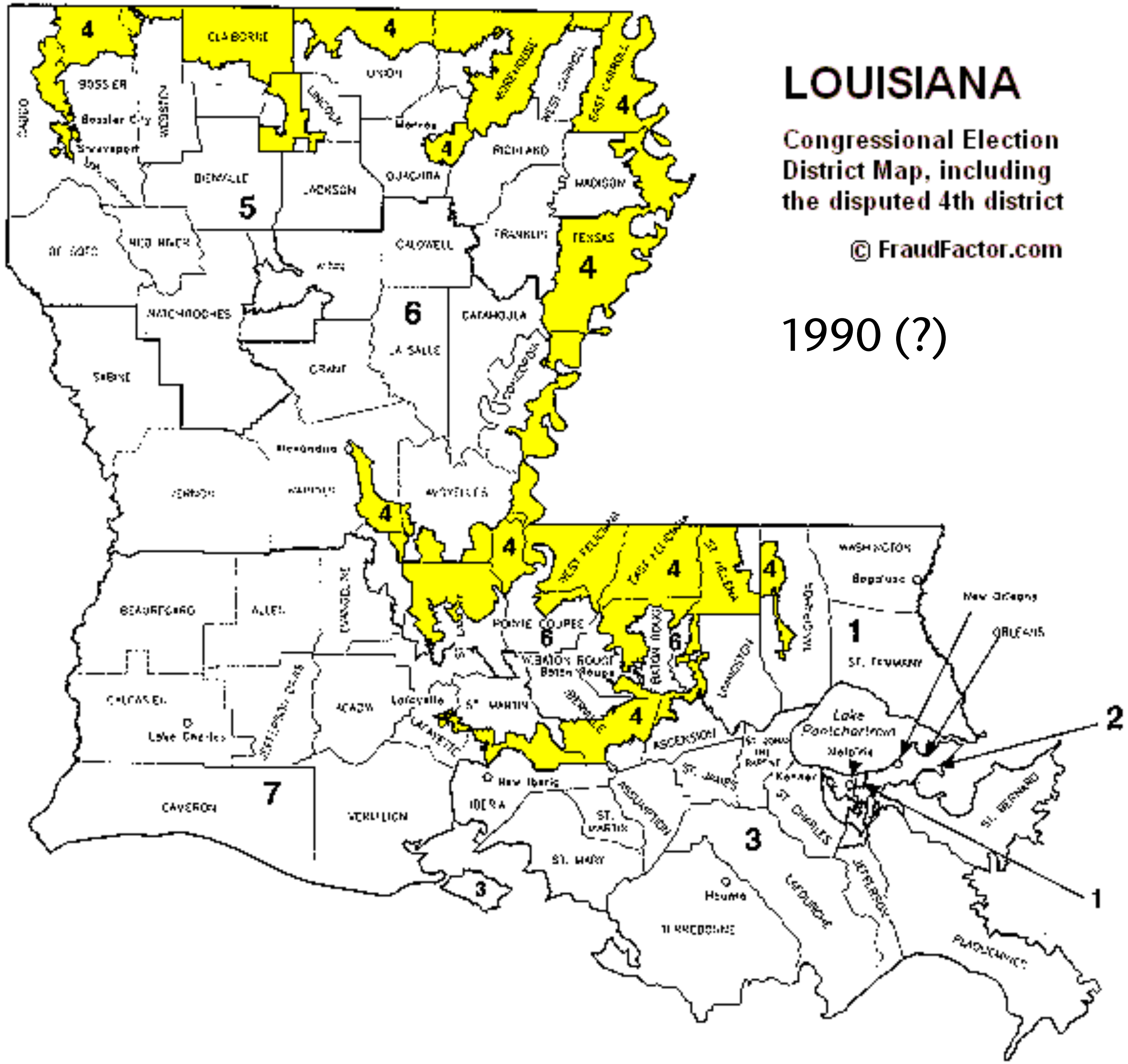
→ 1x R, 3x D representatives

→ 3x R, 1x D representatives

- Bylaws for redistricting in the US:
  - Same number of voters per district
  - Each district must be contiguous
  - Districts should be "compact" (but not precisely defined)

Bad Example

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



1990 (?)

- A possible, precise definition of district compactness:
  Let $\mathcal{D} = \{D_1, \ldots, D_k\}$ electoral districts.
  Each district contains a number of voters with locations p_i , i.e.,

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

  Define the compactness of a district as

$$c(D) = \sum_{k,l=i}^{j} d(p_k, p_l)$$

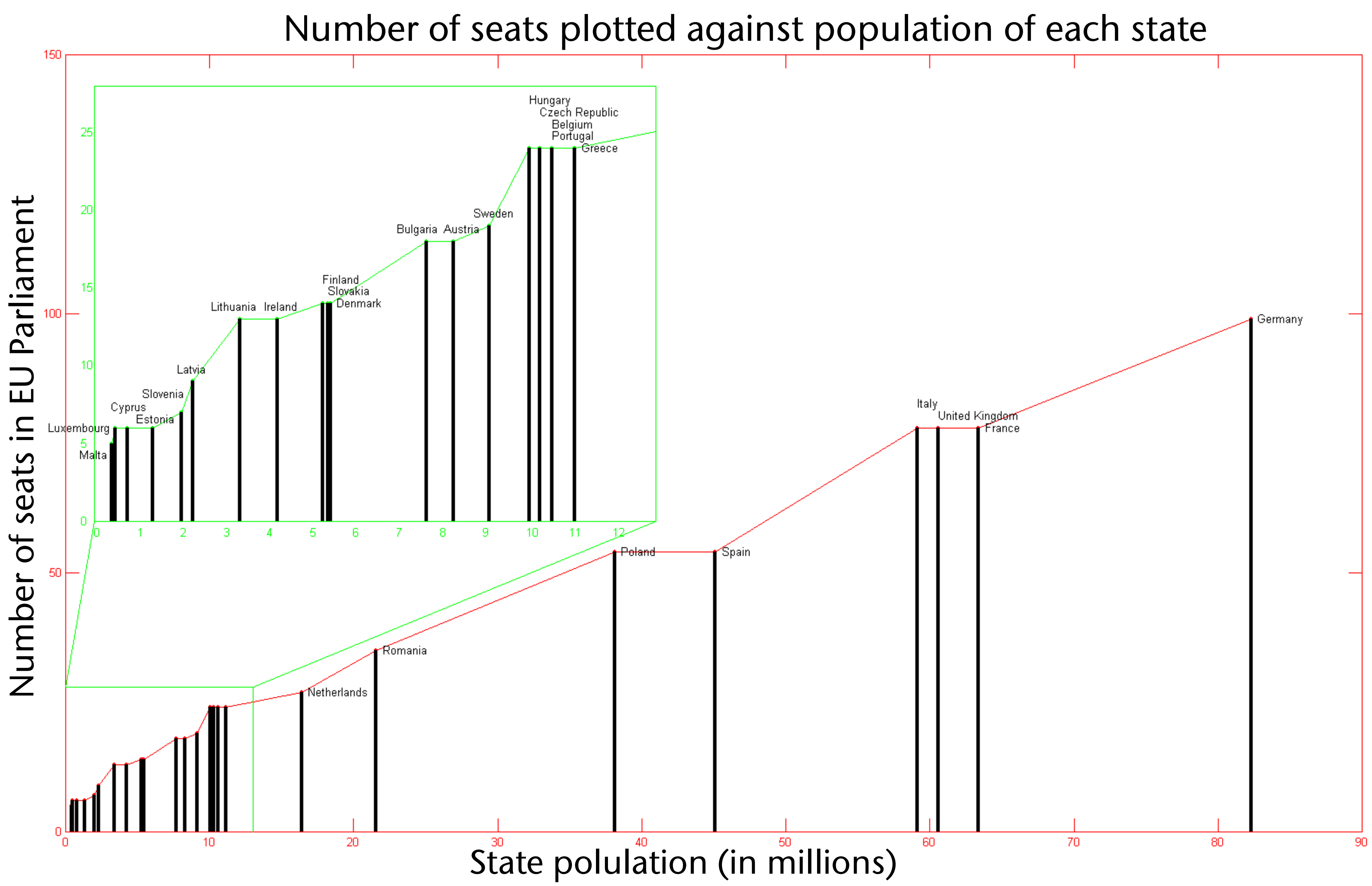
  The total compactness of the redistricting is then

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

- Theorem (w/o proof):
  An optimal partitioning of the country into districts (wrt. the compactness measure just defined) can be derived from the power diagram.
- Redistricting task:
  - For a given set of voters $\{p_i\}$, construct a set of Voronoi generators and appropriate weights such that $\forall i : |D_i| = n$
  - The Voronoi sites can be the polling stations
  - Weights = measure for the population density in the districts (small weight = large density)
- Approach:
  - Start with random sites and weights
  - Iteratively move the sites and change the weights until $c(D)$ reaches the min

Similar Effect in the European Elections

- Votes of people from Malta or Luxembourg have about 10x more weight than those of German voters!



Visibility Sorting Using Voronoi Diagrams

- Reminder: BSPs for Visibility Sorting
- Method:
  - Define a visibility relation on Voronoi regions

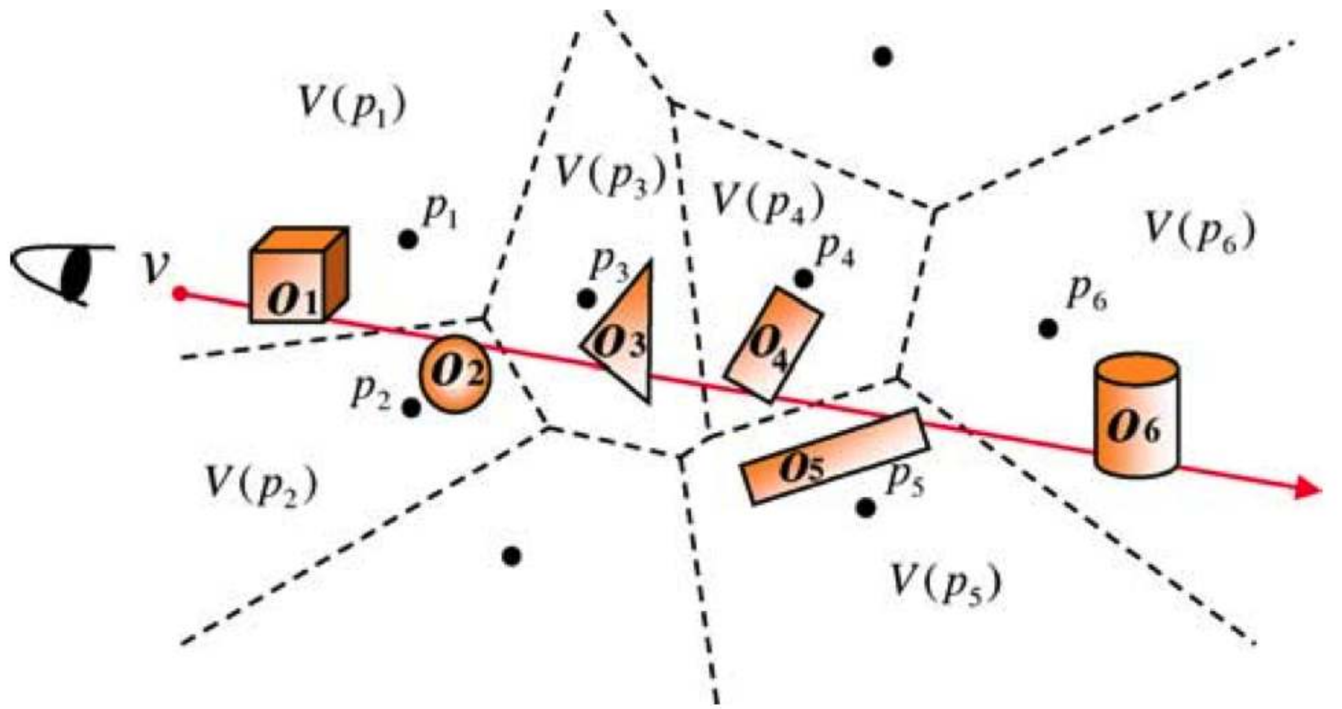
    $$R_1 \prec_v R_2$$

    each point of Voronoi cell R2 is hidden by a point of cell R1 with respect to viewpoint v
  - Now applies:

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

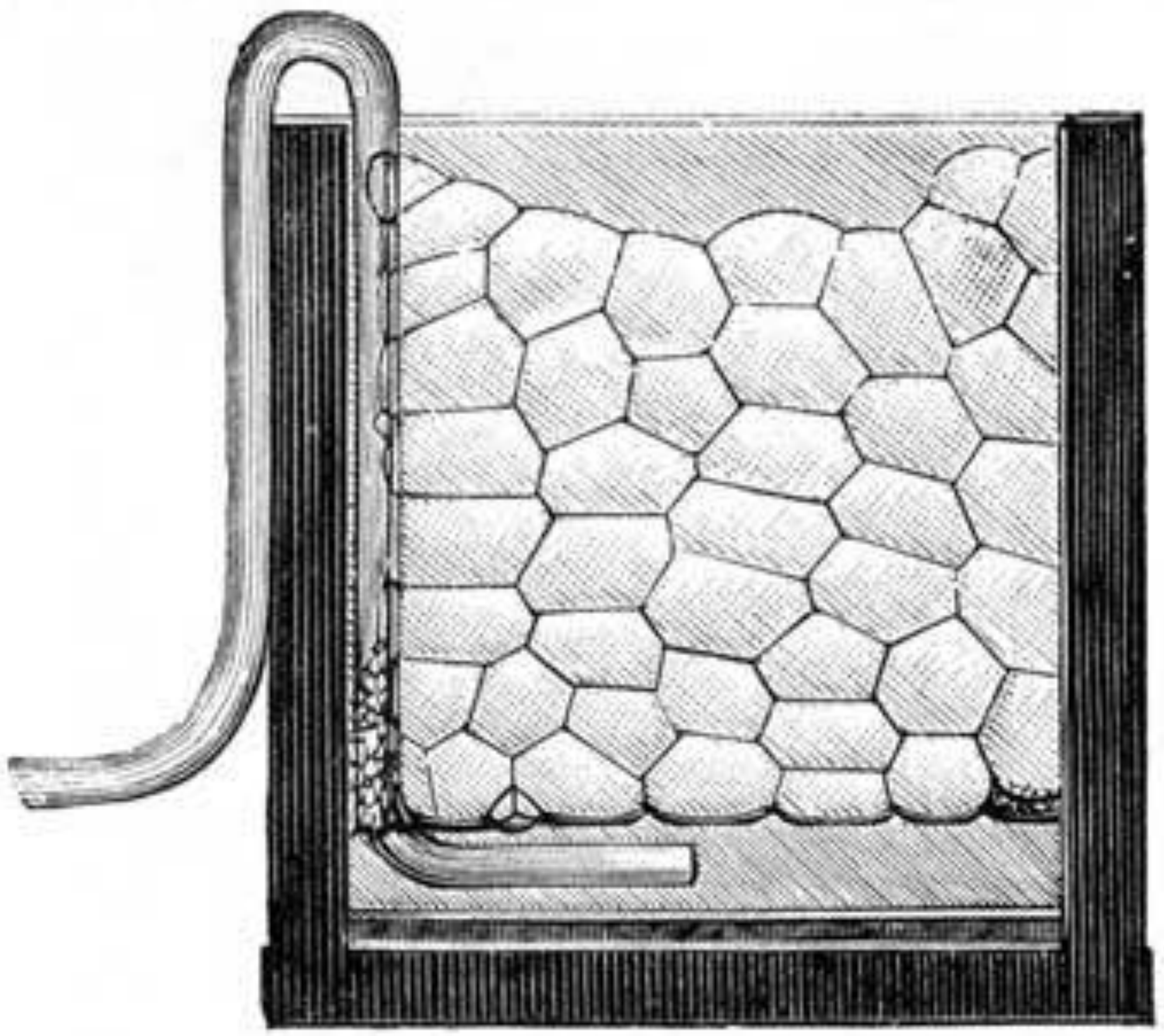
    - Proof: clear because R1 and R2 are completely on different sides of the bisector between R1 and R2.

- Idea:
  - First cluster all polygons into Voronoi cells.
  - At runtime, sort only the Voronoi sites (incrementally).

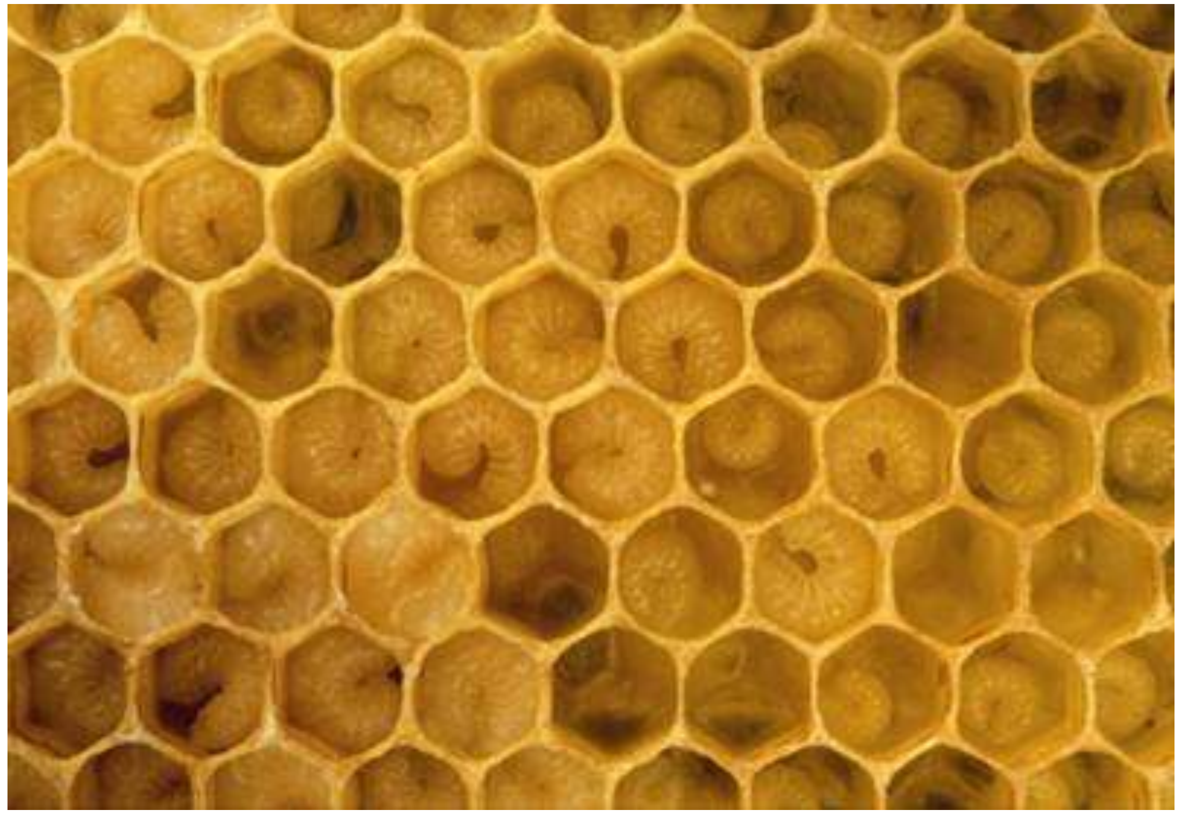


- Approach to Voronoi clustering:
  - Initialize: one cell per polygon with centroid as site.
  - Delete the smallest cell:
    - Recalculate Voronoi diagram locally
    - Assign polygons to the smallest neighboring cell
- Abort if no cell can be resolved without creating a cyclic visibility order in a cell

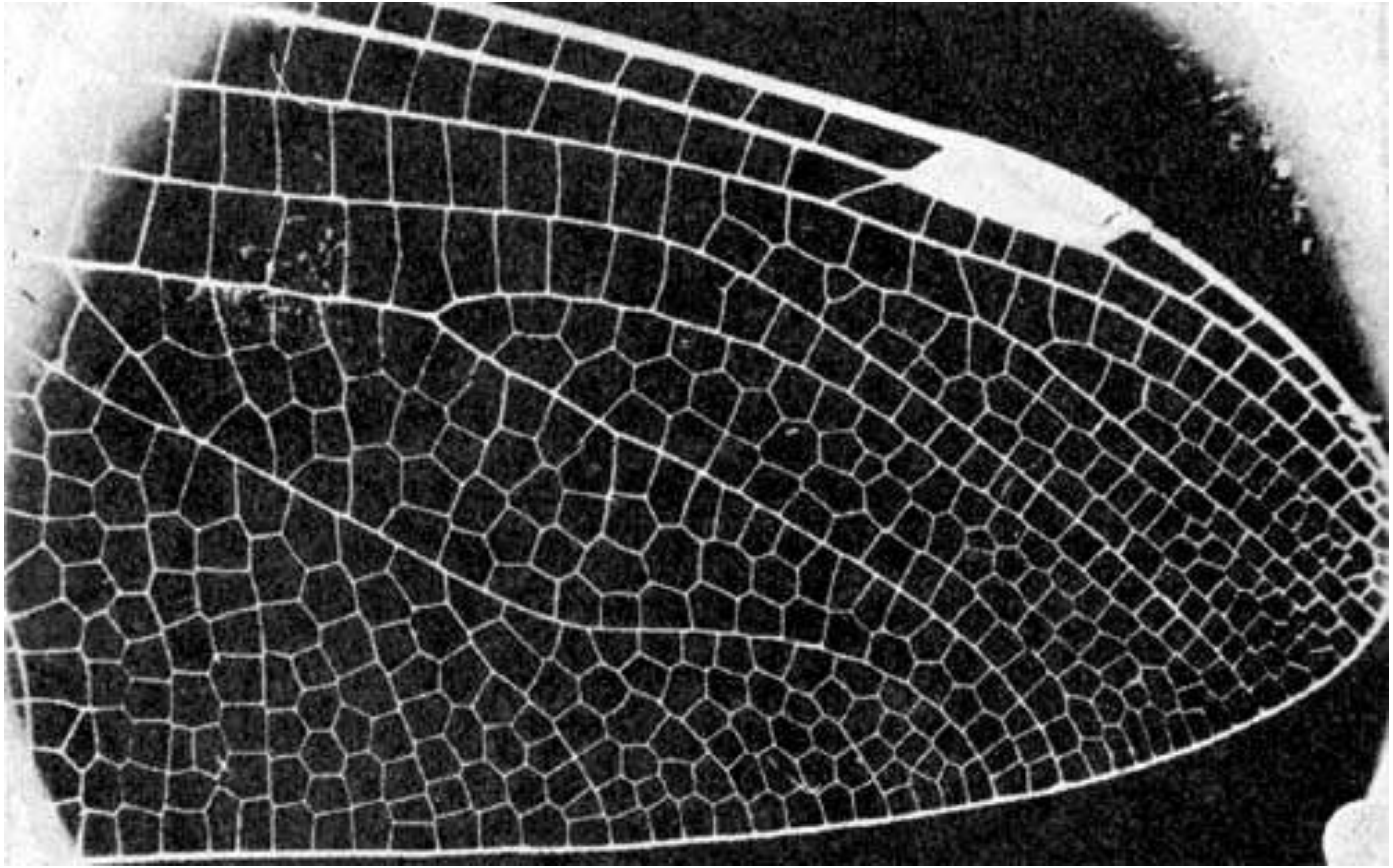
Voronoi Diagrams in Nature



Soap bubbles between two glass plates



Honey comb
(centroidal Voronoi tessellation)

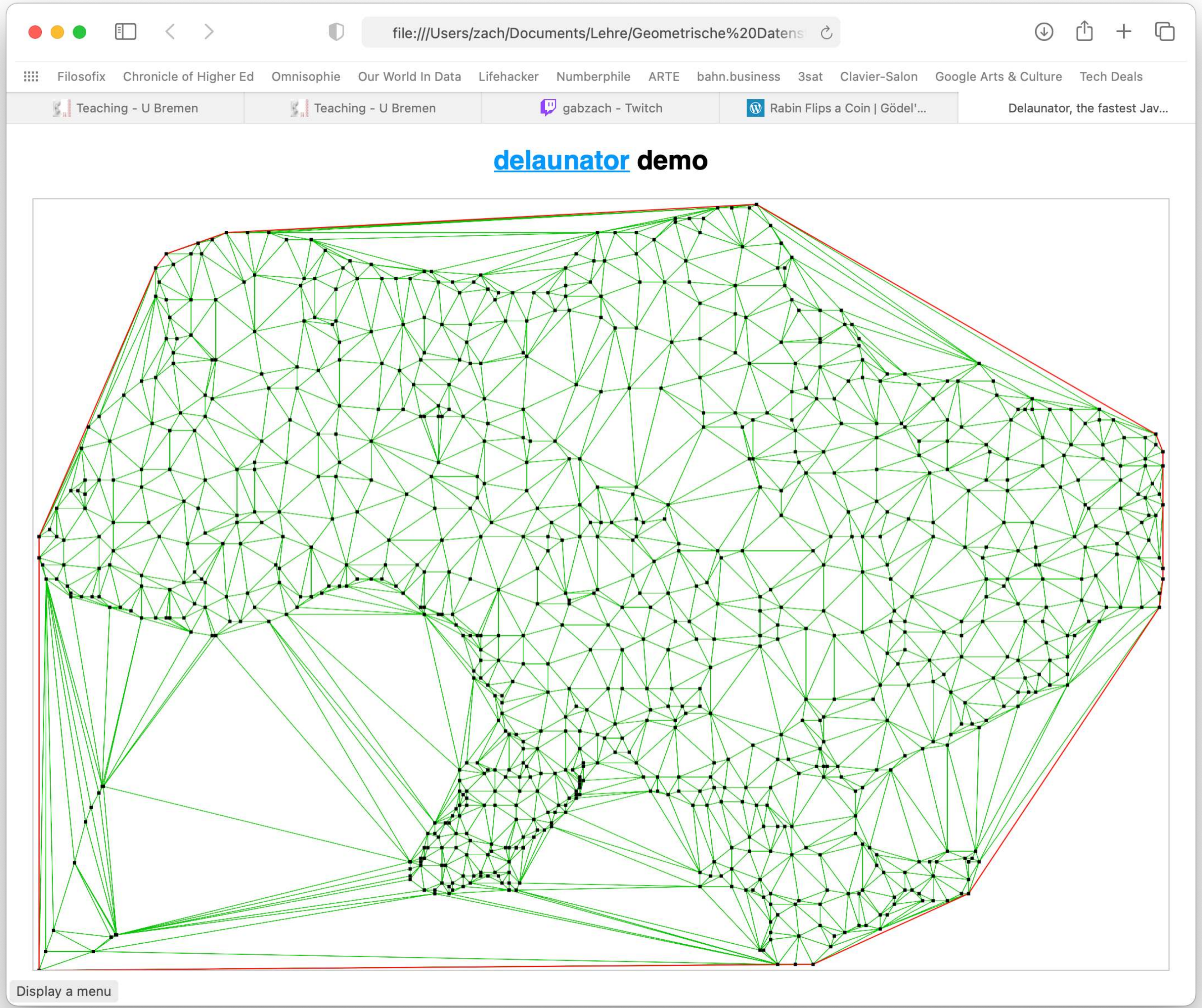


Wings of dragonfly

Voronoi Diagrams in Interactive Art

Scott Snibbe, phaeno, Wolfsburg

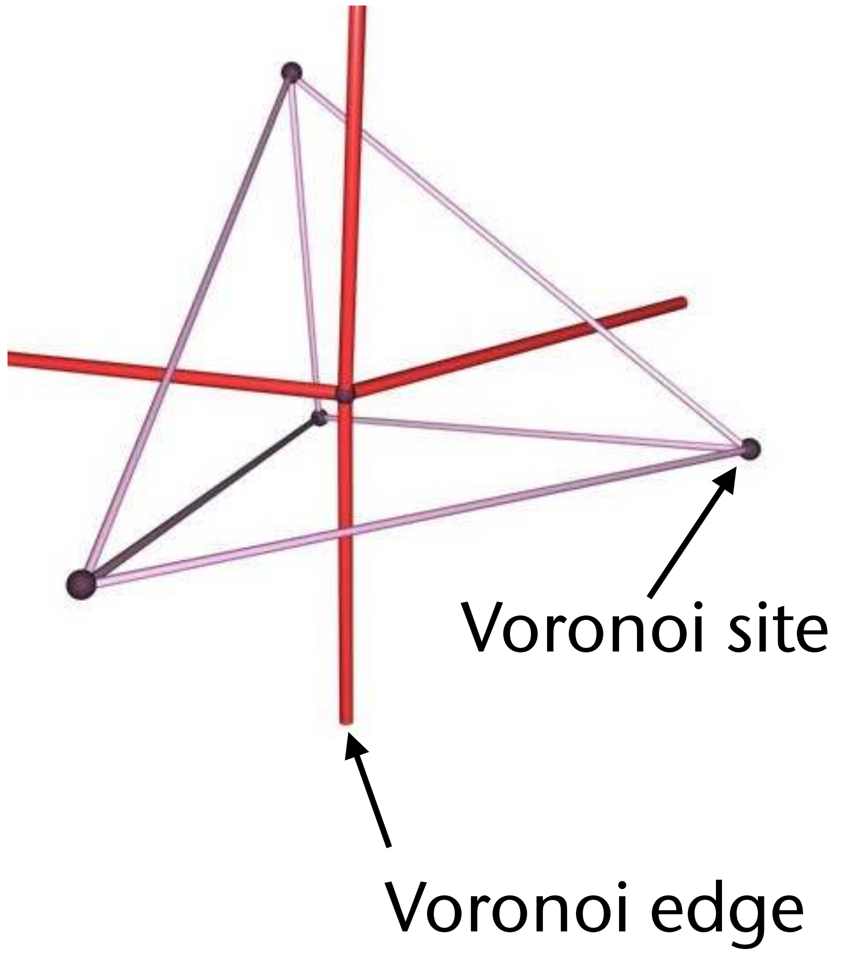
Demo of Delaunay Triangulation in 2D

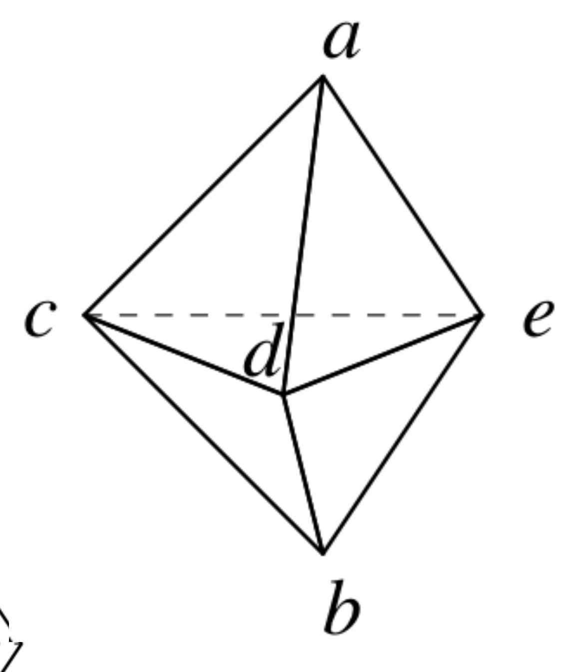


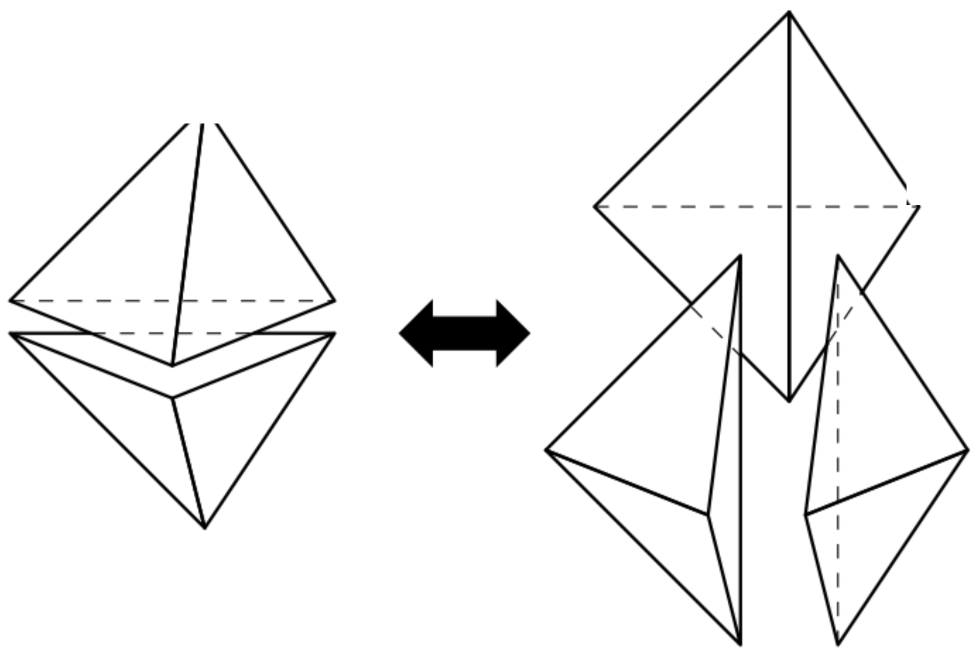
https://github.com/mapbox/delaunator

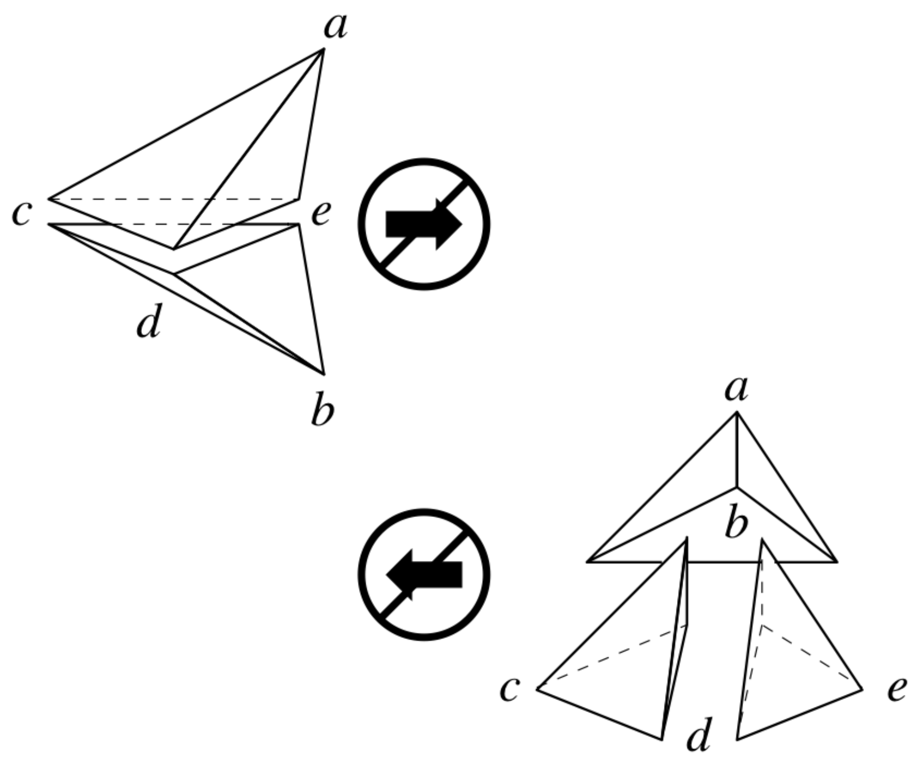
Voronoi / Delaunay in 3D

- Delaunay tetrahedron
- Bisectors = planes
- Edge flip → becomes:
  - Replace 2 tetrahedra by 3 (replace triangle by edge); or
  - Replace 3 tetrahedra by 2

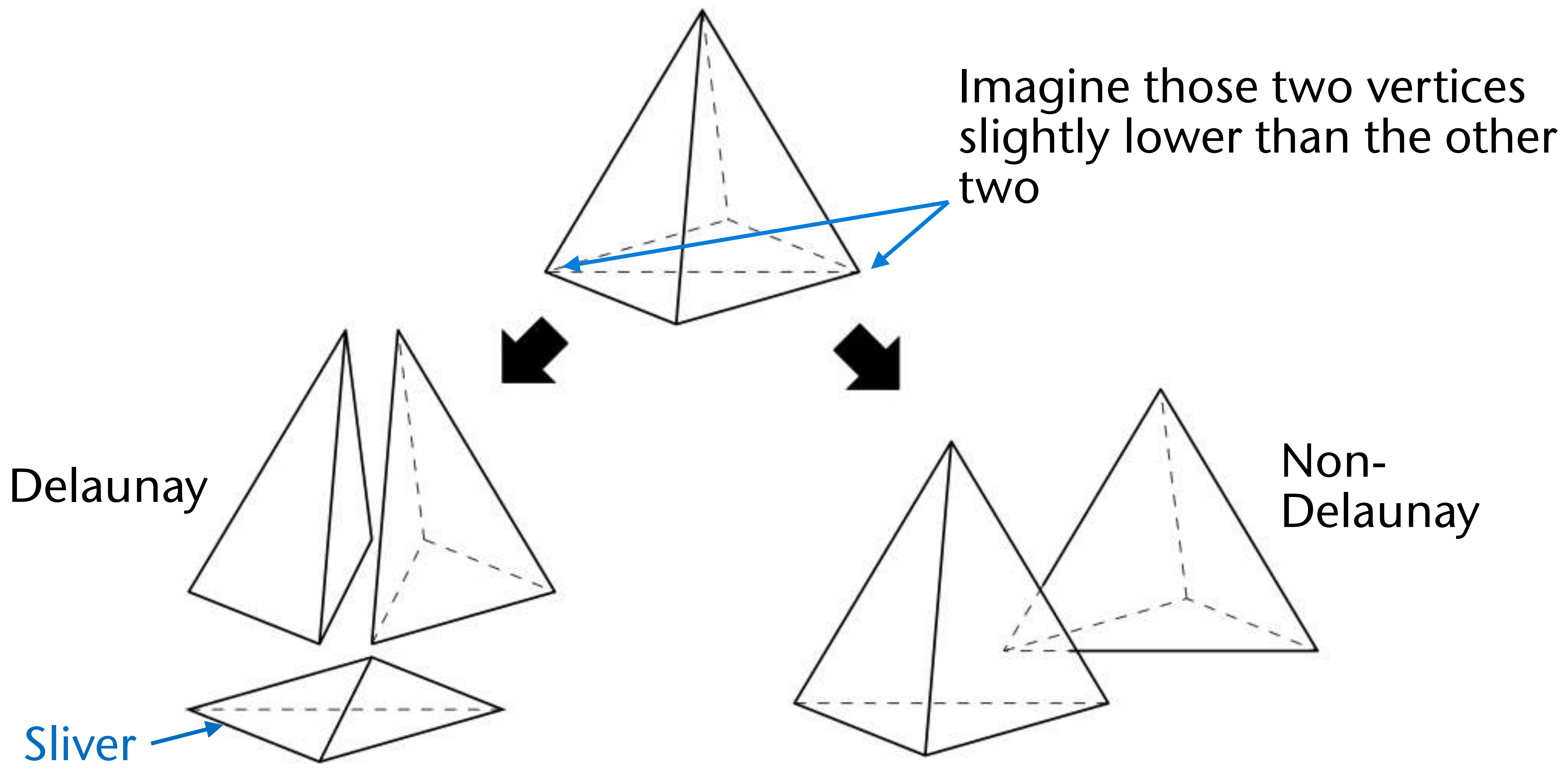






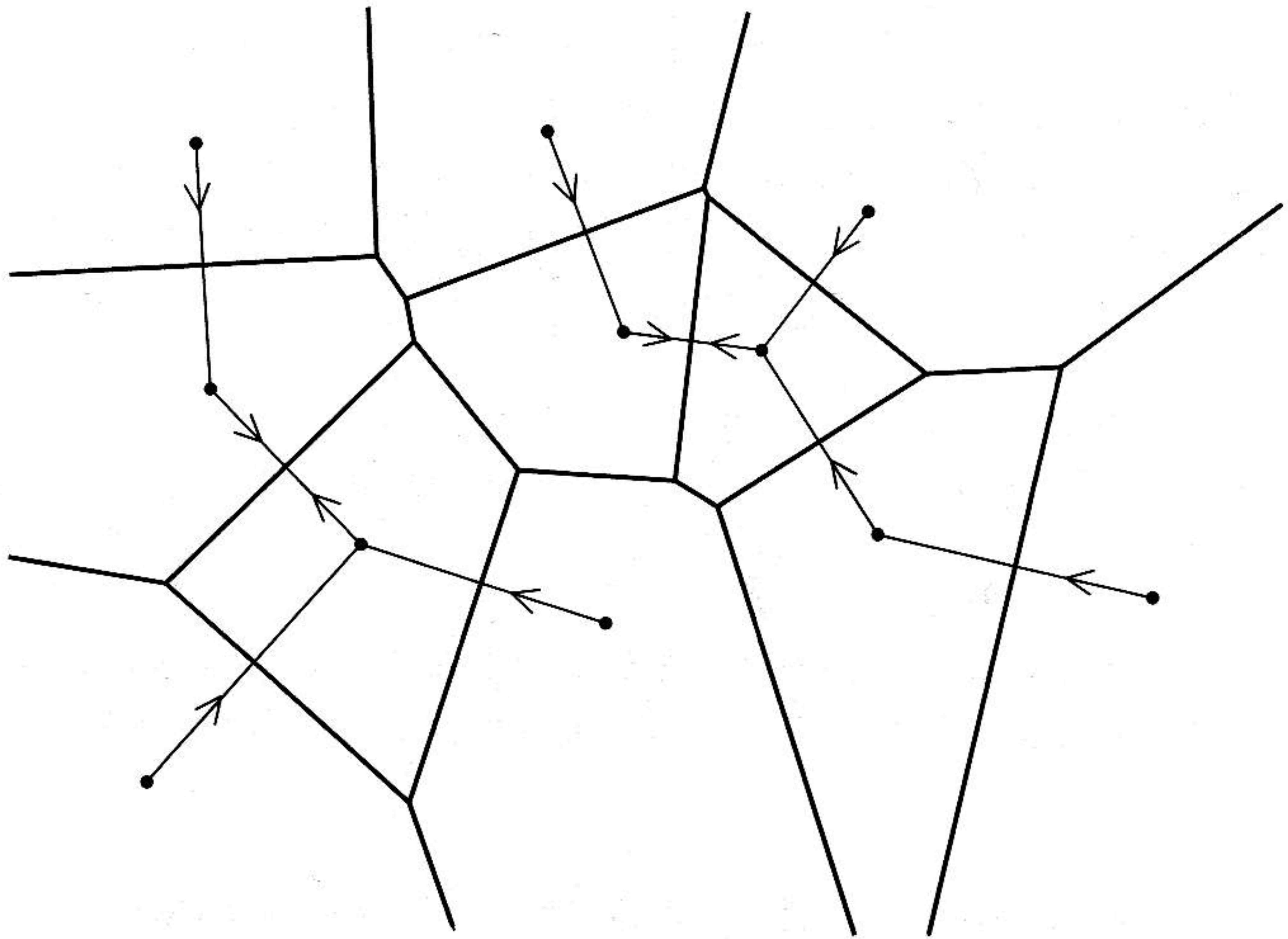


- Slivers in 3D Delaunay triangulations (tetrahedralizations):



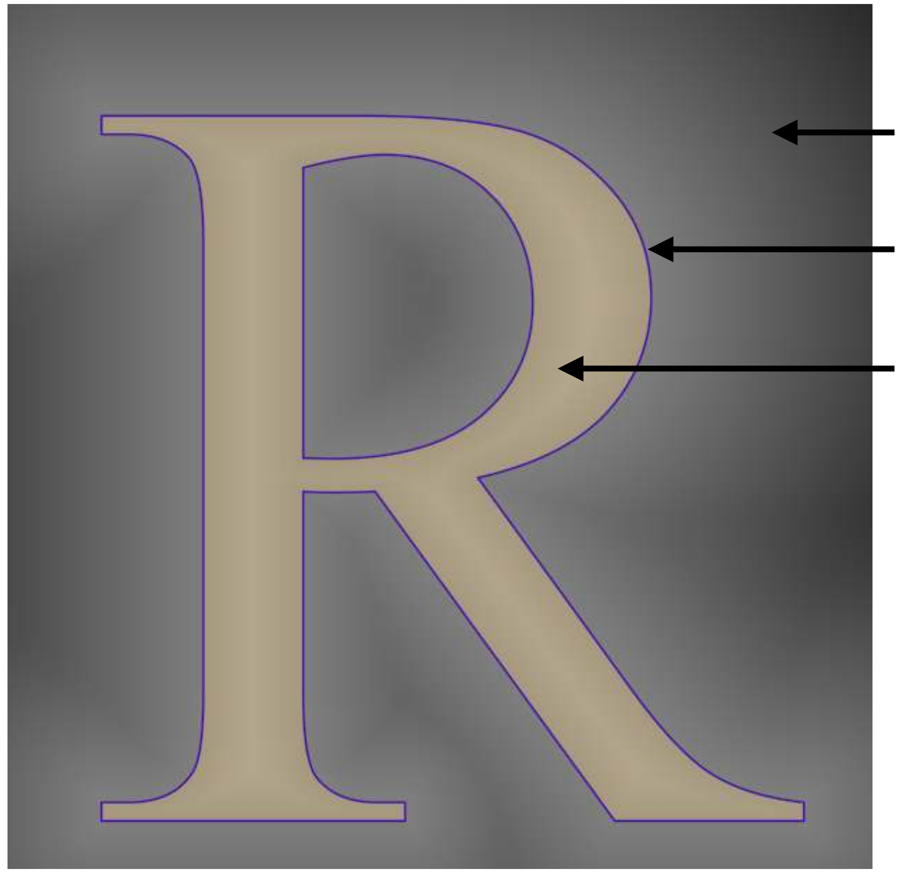
- Sad truth: the max-min-angle property holds only in 2D! ☹

Example for $NNG(S) \subseteq D(S)$



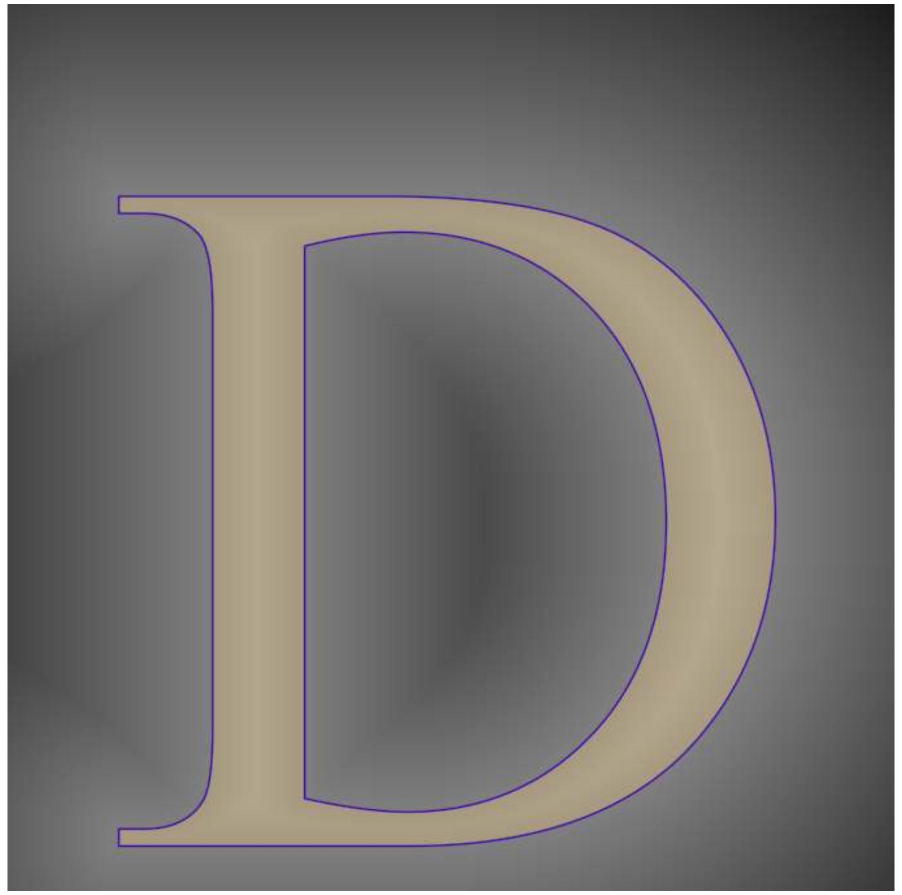
Distance Fields

2D Shape

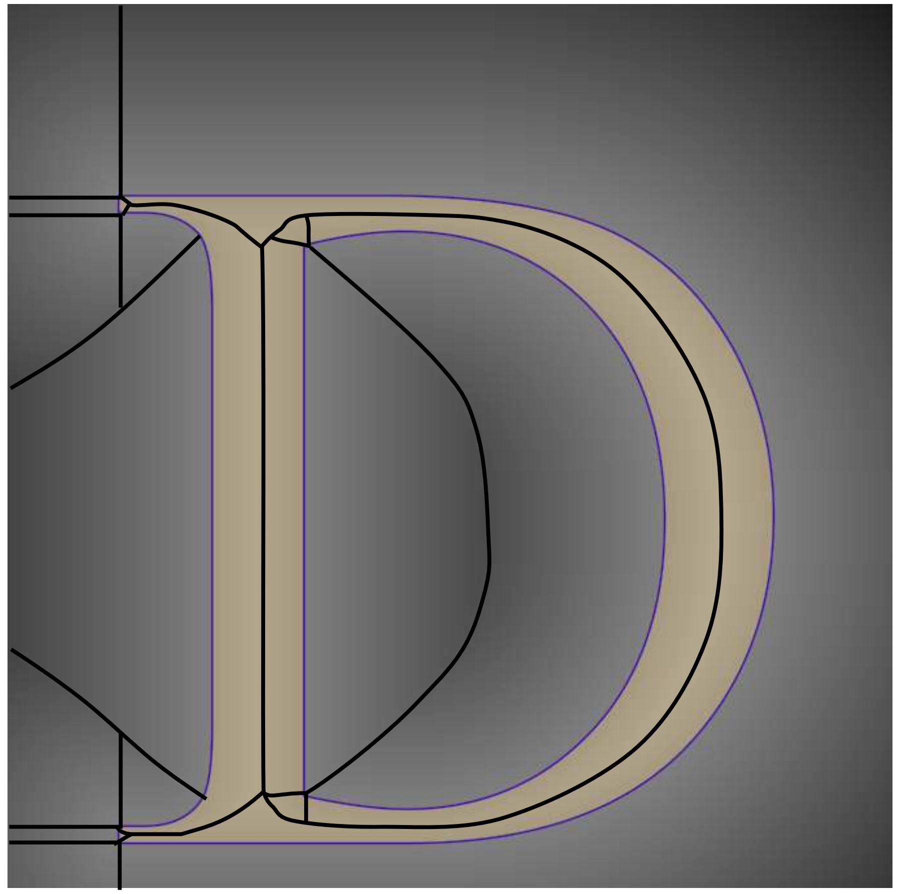


Shape's distance field

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

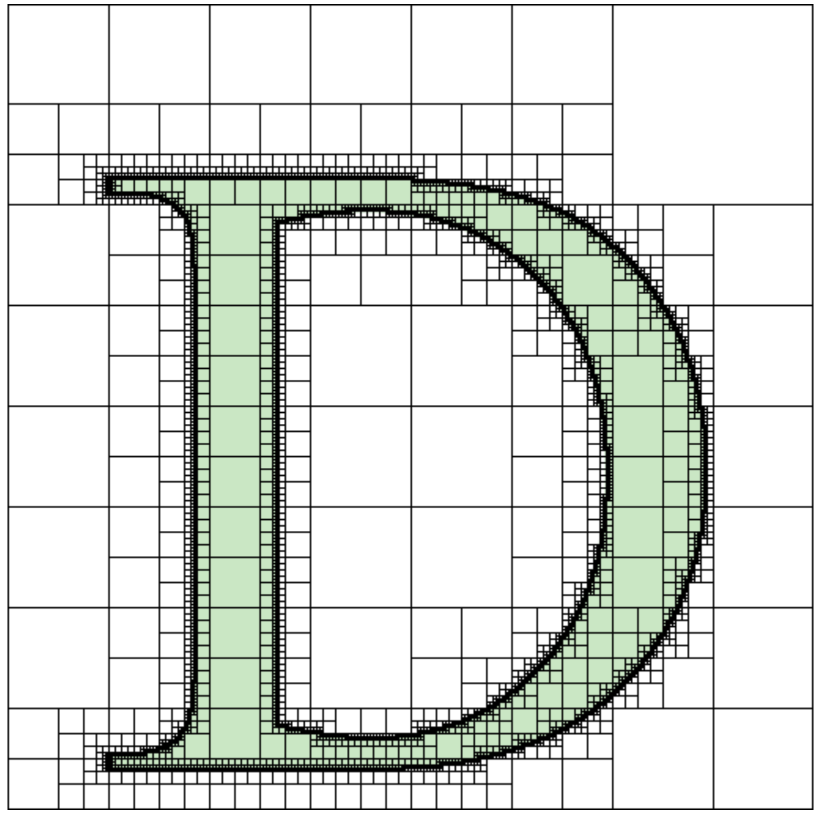


Distance field is C^0 continuous

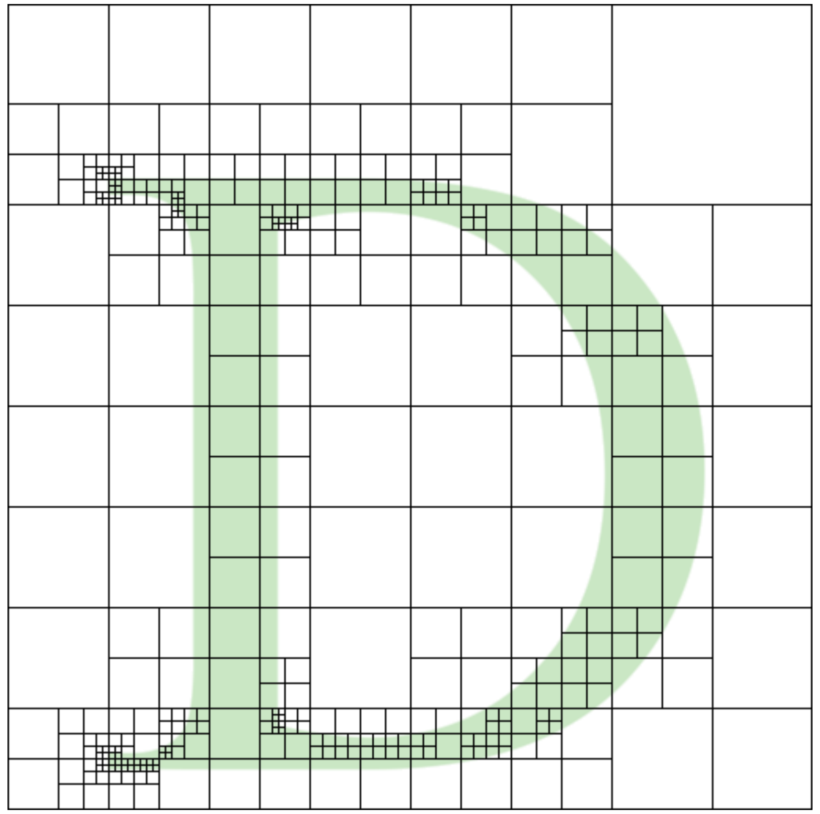


C^1 continuous except at Voronoi boundaries

- 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)

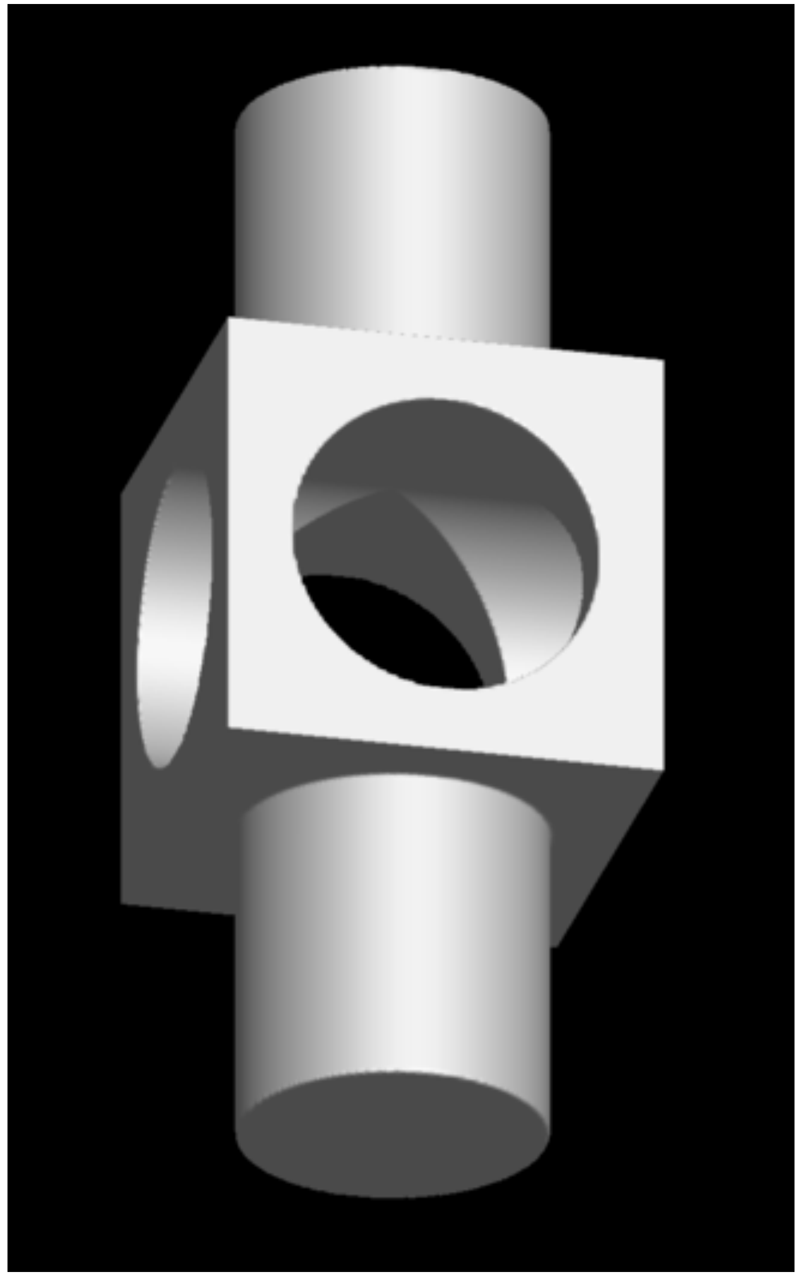


Boundary-limited quadtree

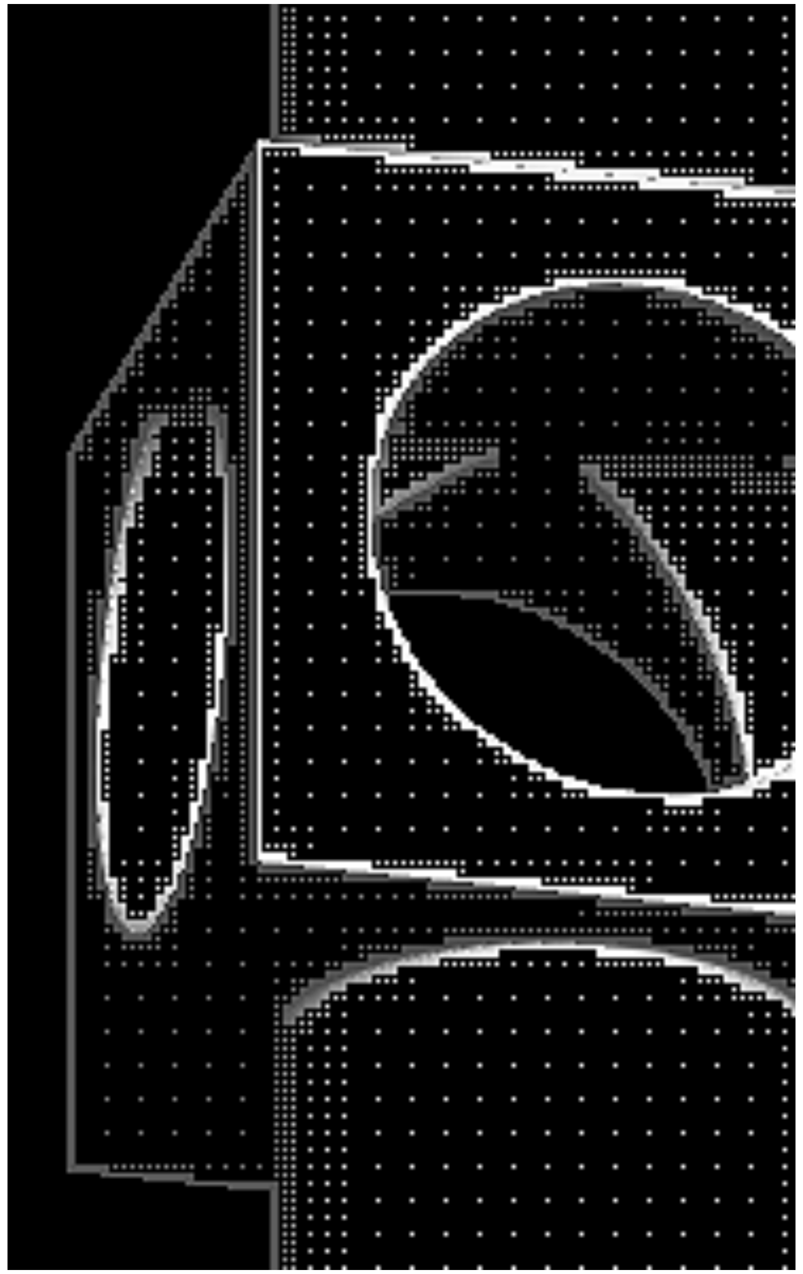


Detail-directed ADF

- Rendering ADF's using adaptive ray-casting:

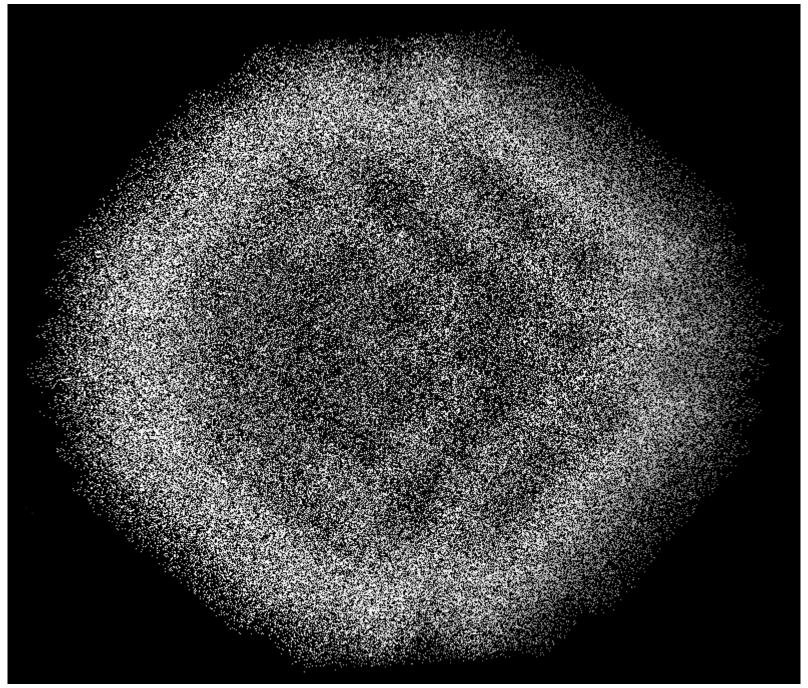


Rendered via adaptively ray casting

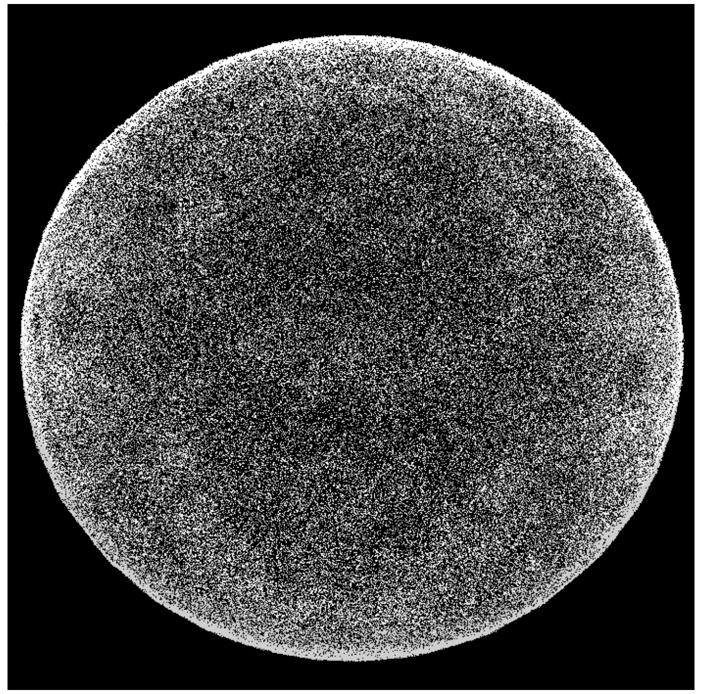


Rays cast to render part of the image on the left

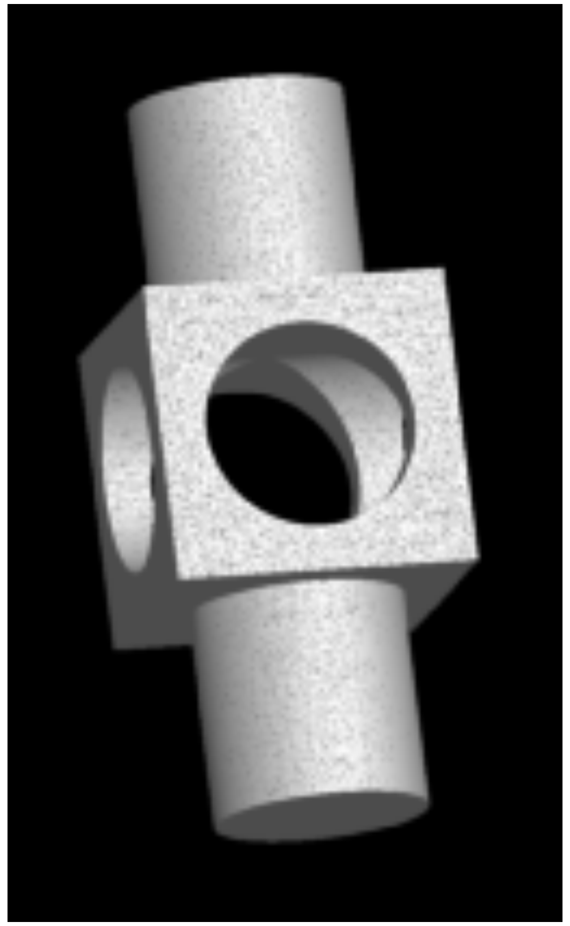
- Point-based rendering of ADF's:
  - Seed each boundary leaf cell with randomly placed points, number of points proportional to cell size
  - Relax the points onto the ADF surface using the distance field and gradient
  - Optionally shade each point using the field's gradient

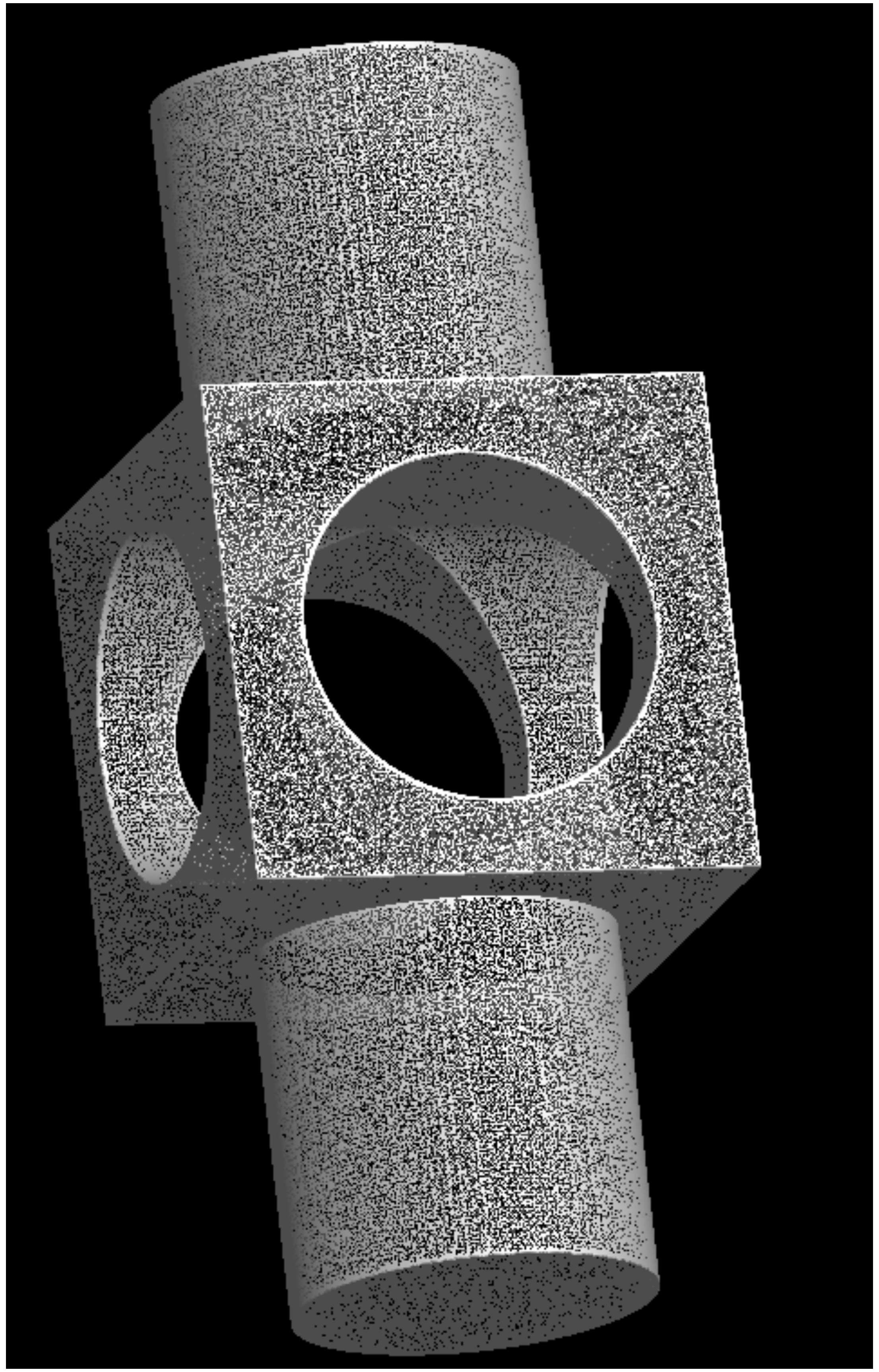


Original points seeded in boundary leaf cells



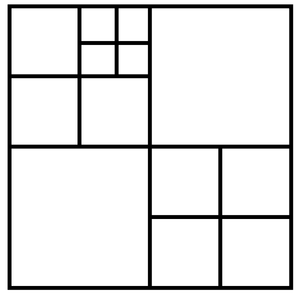
Points after relaxation onto the surface

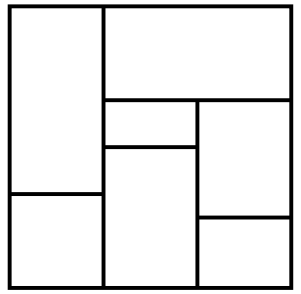


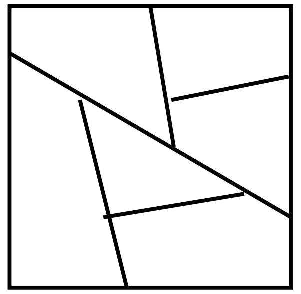


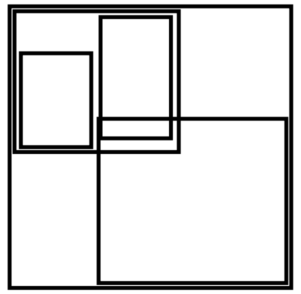
An ADF rendered as points at two different scales

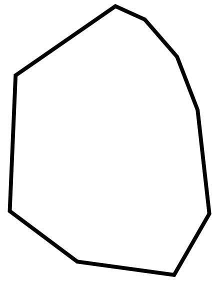
A Continuum of Geometric Data Structures …

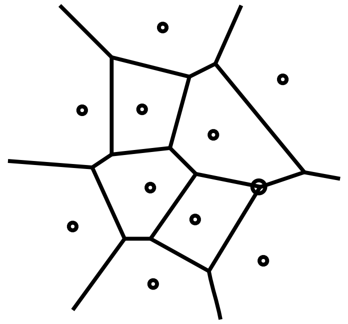

Quadtree

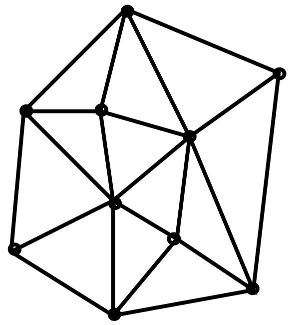

K-d tree


BSP tree


BV hierarchy


Convex hull


Voronoi

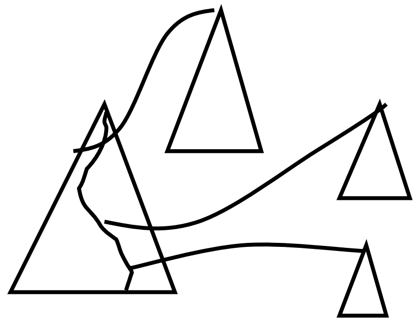

Delaunay

Meshing, terrain visualization, iso-surface generation, Ray casting, distance fields.

Nearest-neighbor search, texture synthesis, shape matching, ray tracing.

Boolean operations, rendering, (Shadows),

Occlusion culling, ray casting, hierarchical coll. detection.


Range tree

Range queries

Thanks Folks!