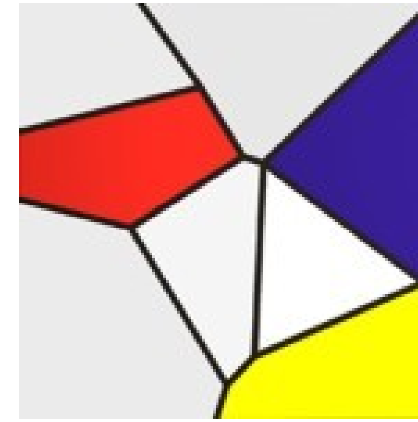
INF555



Fundamentals of 3D

Lecture 5:
Clustering k means
Voronoi diagrams
(+Manipulating images)

Frank Nielsen
nielsen@lix.polytechnique.fr

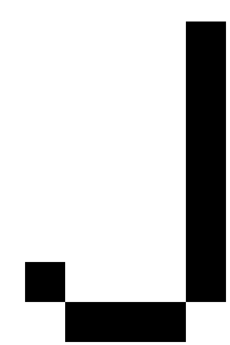
Manipulating PBM/PPM/PGM images in Java

Monochrome bitmap pixels PBM (P1)

Magic code

```
....X.
....X.
....X.
....X.
....X.
....X.
X...X.
.XXX..
......
......
```

```
P1
# This is an example bit map file j.pbm
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```



http://en.wikipedia.org/wiki/Netpbm_format

Manipulating PBM/PPM/PGM images in Java

Portable Grey Map (PGM): P2

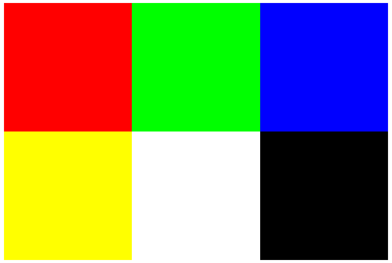
Maximum value (usually 255)

```
P2
# feep.pgm from NetPBM man page on PGM
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Manipulating PBM/PPM/PGM images in Java

PPM ASCII image file: P3

```
P3
#the P3 means colors are in ascii, then 3 columns # and 2 rows, then
255 for max color, then RGB triplets
3 2
255
255 0 0
0 255 0
0 0 255
255 255 0
255 255 255
0 0 0
```



PPM binary image file: P6

```
P6
#any comment string
3 2
255
!@#$%^&*()_+|{}:"<
```

Manipulating portable pixmaps in Java

Beware: size of « raw » images are large compared to:

- Lossless compression PNG format,
- Lossy compression JPEG

www.enseignement.polytechnique.fr/profs/informatique/Philippe.Chassignet/PGM/pgm_java.html

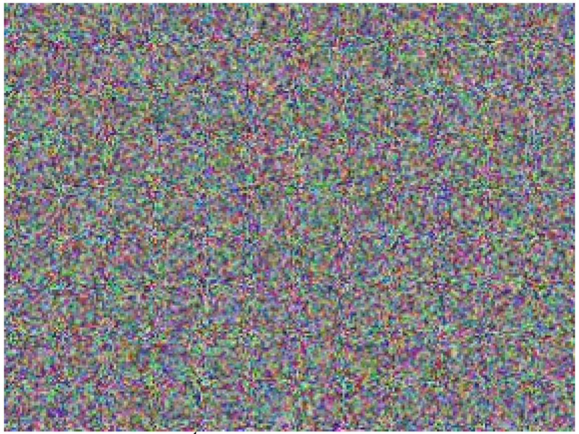
Support screen snapshots

http://en.wikipedia.org/wiki/Comparison_of_image_viewers

```
public void read(String fileName){String line;StringTokenizer st;int i;
    try {
        DataInputStream in = new DataInputStream(new BufferedInputStream
(new FileInputStream(fileName)));
        in.readLine();
        do {  line = in.readLine(); } while (line.charAt(0) == '#');

        st = new StringTokenizer(line);
        width = Integer.parseInt(st.nextToken());
        height = Integer.parseInt(st.nextToken());
            r = new int[height][width];
          g = new int[height][width];
          b = new int[height][width];
        line = in.readLine();
        st = new StringTokenizer(line);
        depth = Integer.parseInt(st.nextToken());

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                    r[y][x] = in.readUnsignedByte();
                    g[y][x] = in.readUnsignedByte();
                    b[y][x] = in.readUnsignedByte();
                }
            }
        in.close();
    } catch(IOException e) {}
  }
```

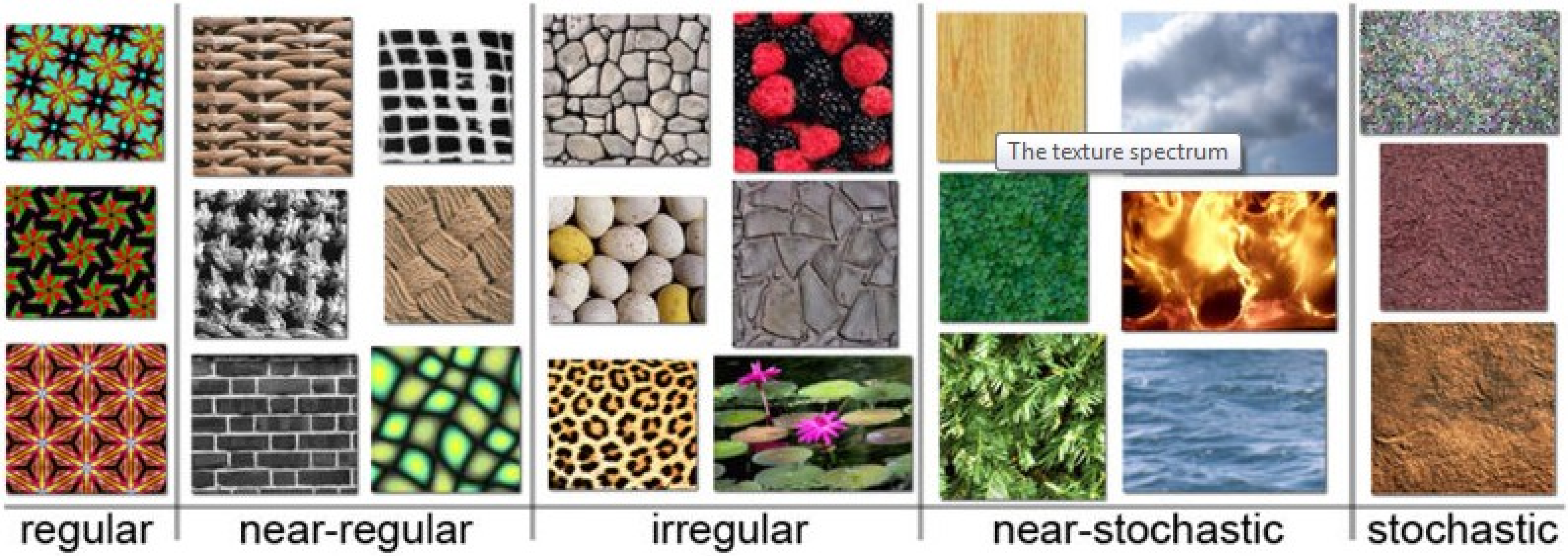
```java
public void write(String filename)
  {
     String line;
     StringTokenizer st;
     int i;
     try {
        DataOutputStream out =new DataOutputStream(
     new BufferedOutputStream(new FileOutputStream(filename)));
        out.writeBytes("P6\n");
        out.writeBytes("# INF555 Ecole Polytechnique\n");
        out.writeBytes(width+" "+height+"\n255\n");

              for (int y = 0; y < height; y++) {
                   for (int x = 0; x < width; x++) {
                        out.writeByte((byte)r[y][x]);
                        out.writeByte((byte)g[y][x]);
                        out.writeByte((byte)b[y][x]);
                   }
              }
        out.close();
     } catch(IOException e) {}
  }
```

```
class DemoPPM
{
public static void main(String [] arg)
{
PPM ppm=new PPM();

ppm.read("polytechnique.ppm");
ppm.write("copy.ppm");

PPM ppm2=new PPM(ppm.width,ppm.height);

for(int i=0;i<ppm2.height;i++)
	for(int j=0;j<ppm2.width;j++)
	{
	ppm2.r[i][j]=(int)(Math.random()*255.0);
	ppm2.g[i][j]=(int)(Math.random()*255.0);
	ppm2.b[i][j]=(int)(Math.random()*255.0);
	}

ppm2.write("random.ppm");
}
}
```

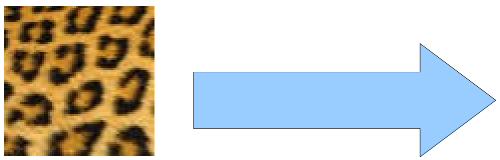


Randomly colored bitmap (PPM)

Stochastic texture synthesis



Texture Synthesis

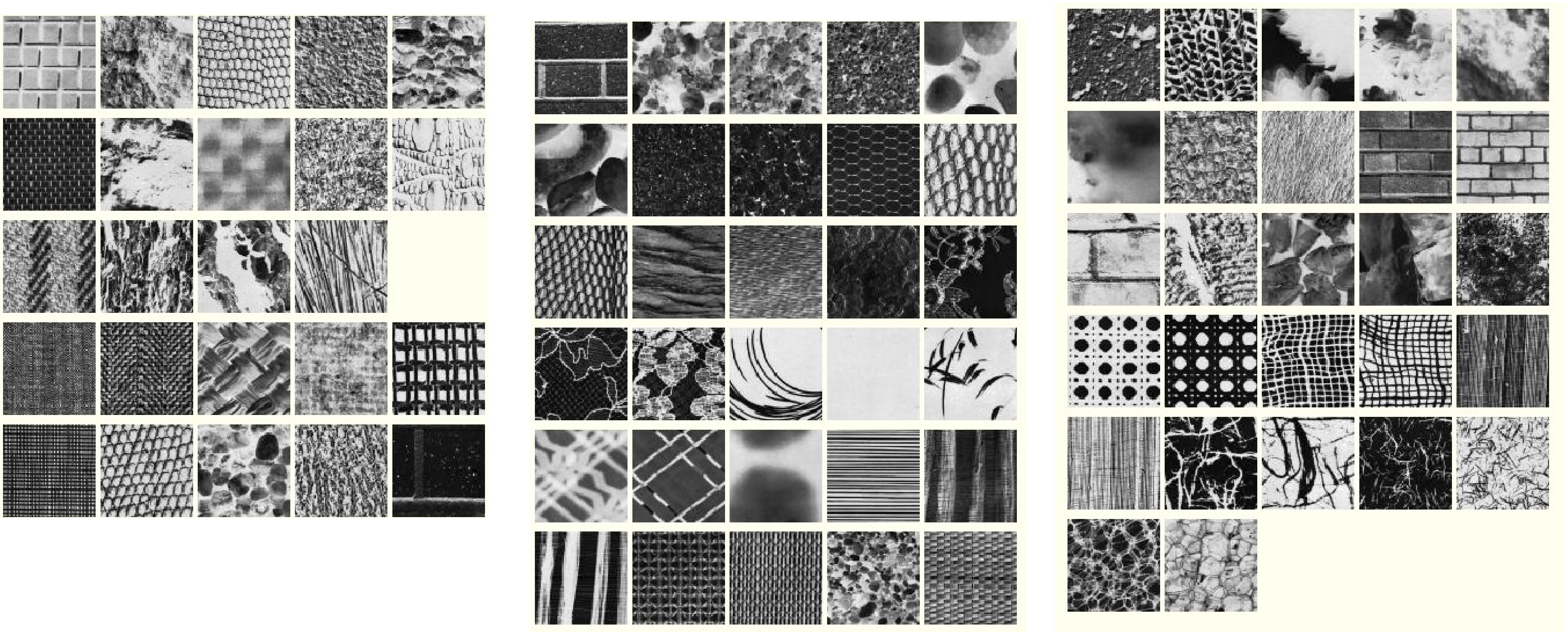


Source
(=exemplar)

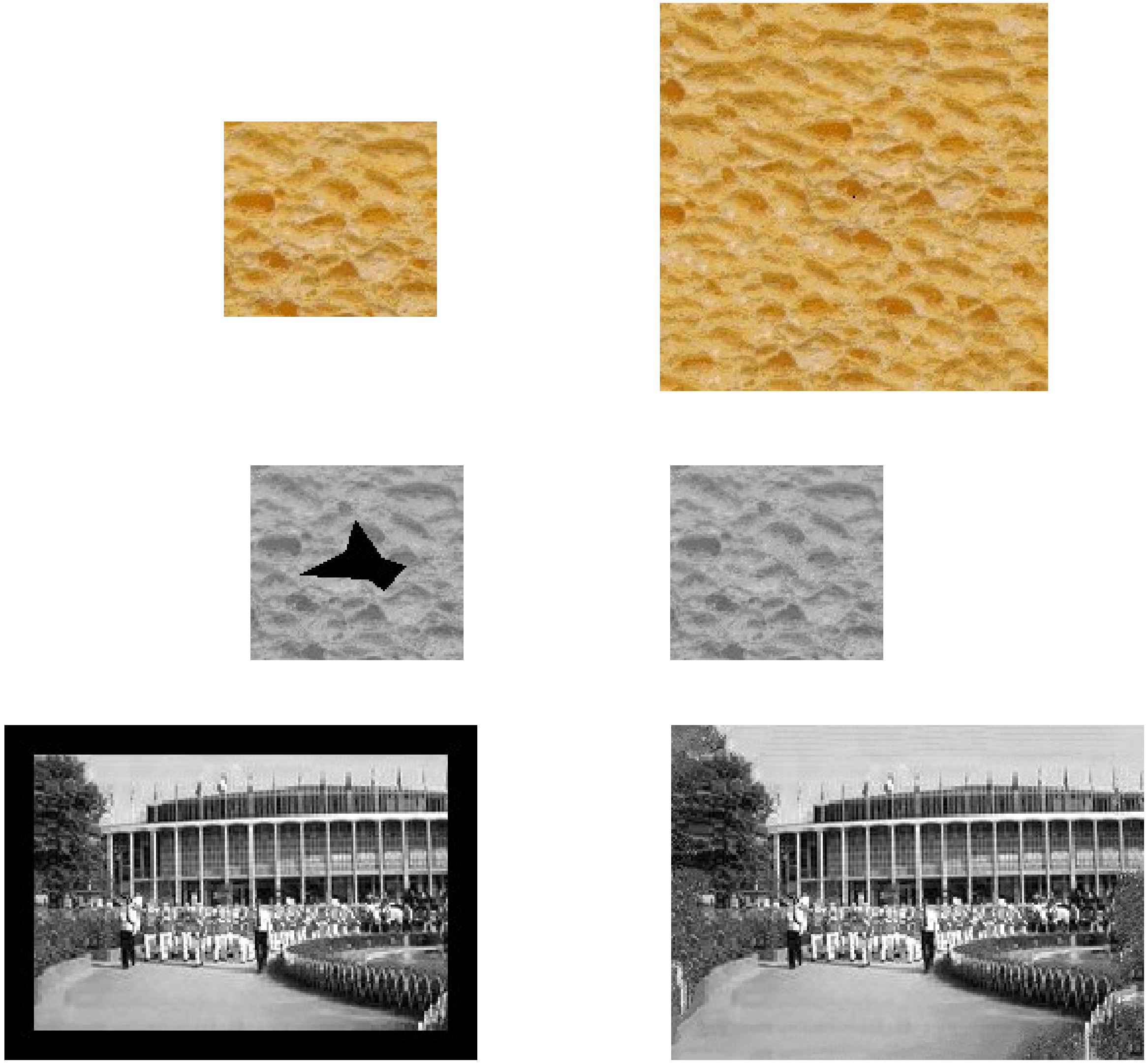
Target

http://en.wikipedia.org/wiki/Texture_synthesis

Broadatz texture catalog



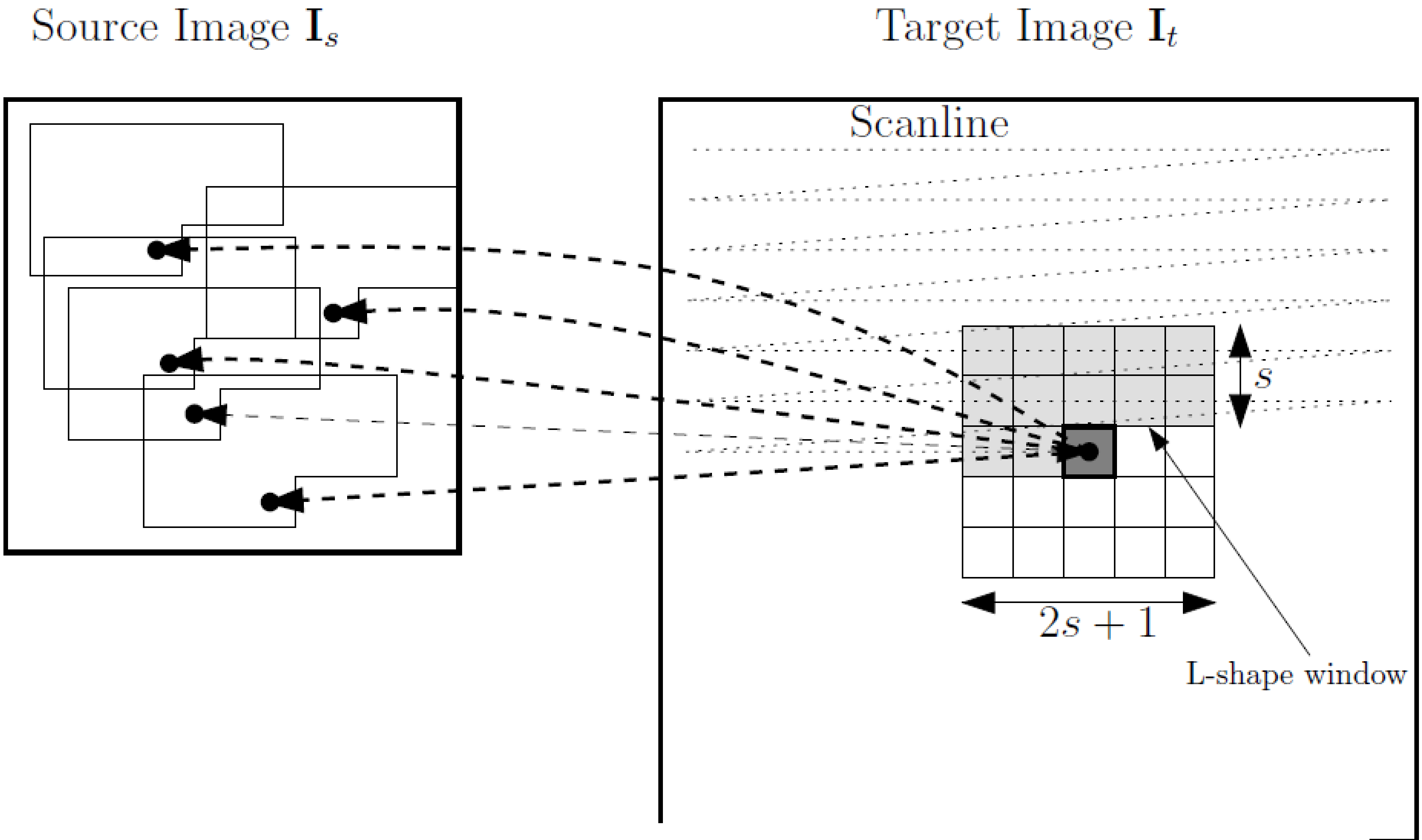
http://www.ux.uis.no/~tranden/brodatz.html
http://sipi.usc.edu/database/database.cgi?volume=textures



http://graphics.cs.cmu.edu/people/efros/research/EfrosLeung.html

``Texture Synthesis by Non-parametric Sampling''
Alexei A. Efros and Thomas K. Leung
IEEE International Conference on Computer Vision (ICCV'99), Corfu, Greece, September 1999

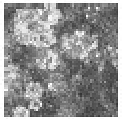
Stochastic texture synthesis

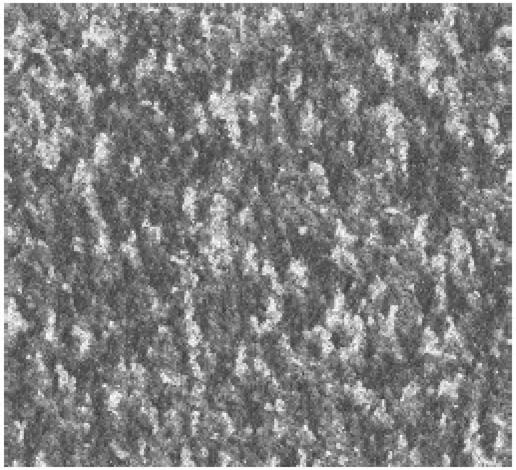


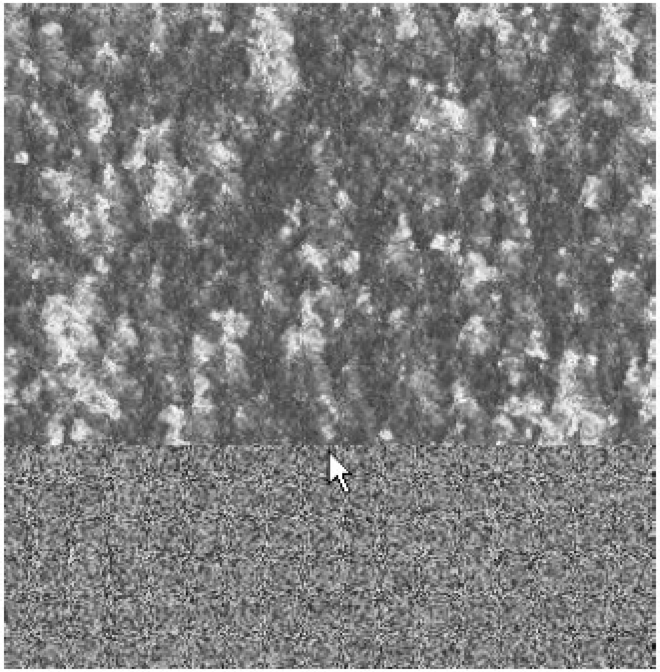
$$\mathrm{SSD}(x_s, y_s; x_t, y_t) = \sum_{l=-s}^{s} \sum_{c=-s}^{s} \mathrm{LShape}(l,c) \left(\mathbf{I}_s[x_s + c, y_s + l] - \mathbf{I}_t[x_t + c, y_t + l]\right)^2$$

$$(x_s, y_s) = \mathrm{argmin}_{(x,y)\in \mathbf{I}_s} \mathrm{SSD}(x, y; x_t, y_t).$$

Stochastic texture synthesis







```
TextureSynthesis(I_s, I_t)
◁ I_s is the input texture sample ▷
◁ Create a large texture I_t ▷
Initialize a random color image I_t
◁ Synthesize pixels following the horizontal scanline order ▷
for y ← 1 to h_t
    do for x ← 1 to w_t
            do (x_s, y_s) = BestLShapeMatch(I_s, x, y)
                I_t[x, y] = I_s[x_s, y_s]
```

Linearization of neighborhood

$2s + 1 = 5$

$c_{-2,-2}$	$c_{-1,-2}$	$c_{0,-2}$	$c_{1,-2}$	$c_{2,-2}$
$c_{-2,-1}$	$c_{-1,-1}$	$c_{0,-1}$	$c_{1,-1}$	$c_{2,-1}$
$c_{-2,0}$	$c_{-1,0}$			

Linearization $d = 2(s^2 + s)$.

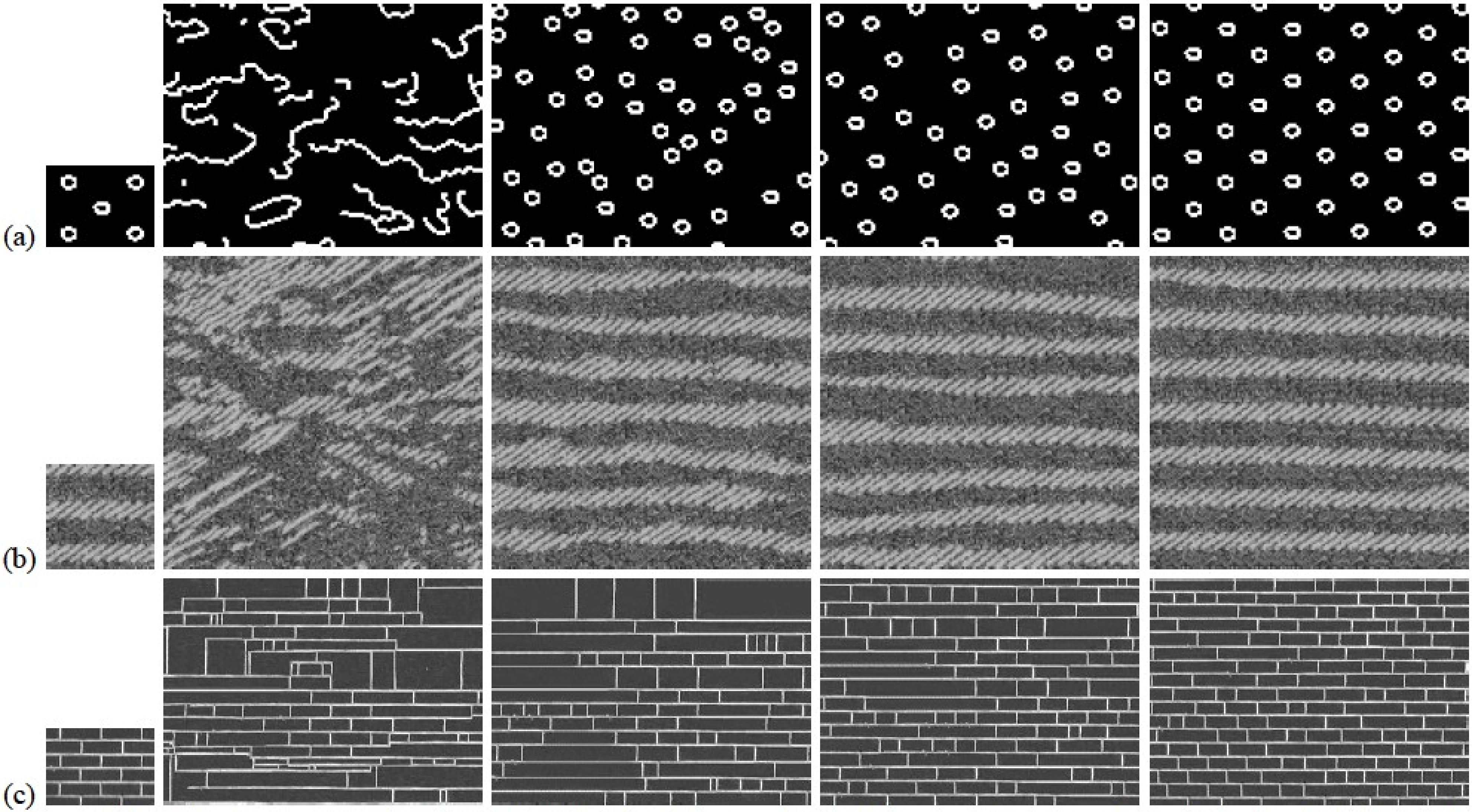
$$\mathbf{n}(x_i, y_j) = \begin{bmatrix} \mathbf{I}_s[x_{i-s}, y_{j-s}] \\ \vdots \\ \mathbf{I}_s[x_{i+s}, y_{j-s}] \\ \mathbf{I}_s[x_{i-s}, y_{j-s+1}] \\ \vdots \\ \mathbf{I}_s[x_{i+s}, y_{j-s+1}] \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{I}_s[x_i - s, y_j] \\ \vdots \\ \mathbf{I}_s[x_i - 1, y_j] \end{bmatrix}.$$

$c_{-2,-2}$	$c_{-1,-2}$	$c_{0,-2}$	$c_{1,-2}$	$c_{2,-2}$	$c_{-2,-1}$	$c_{-1,-1}$	$c_{0,-1}$	$c_{1,-1}$	$c_{2,-1}$	$c_{-2,0}$	$c_{-1,0}$

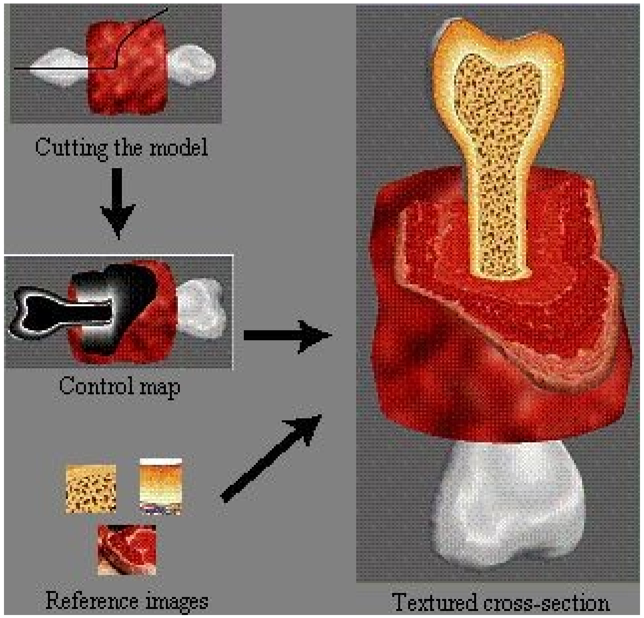
$$\mathrm{SSD}(x_s, y_s; x_t, y_t) = \sum_{l=-s}^{s} \sum_{c=-s}^{s} \mathrm{LShape}(l, c)(\mathbf{I}_s[x_s + c, y_s + l] - \mathbf{I}_t[x_t + c, y_t + l])^2.$$

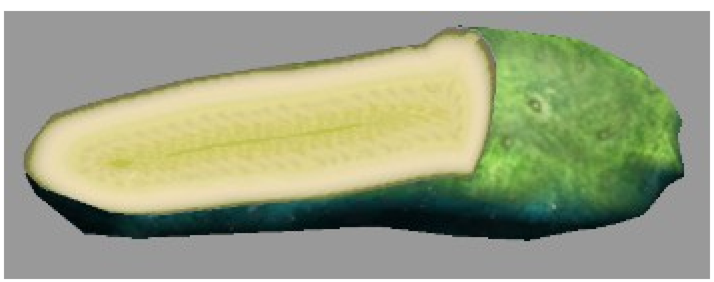
$$\mathrm{SSD}(x_s, y_s; x_t, y_t) = ||\mathbf{n}(x_s, y_s) - \mathbf{n}(x_t, y_t)||^2.$$

Impact of window size

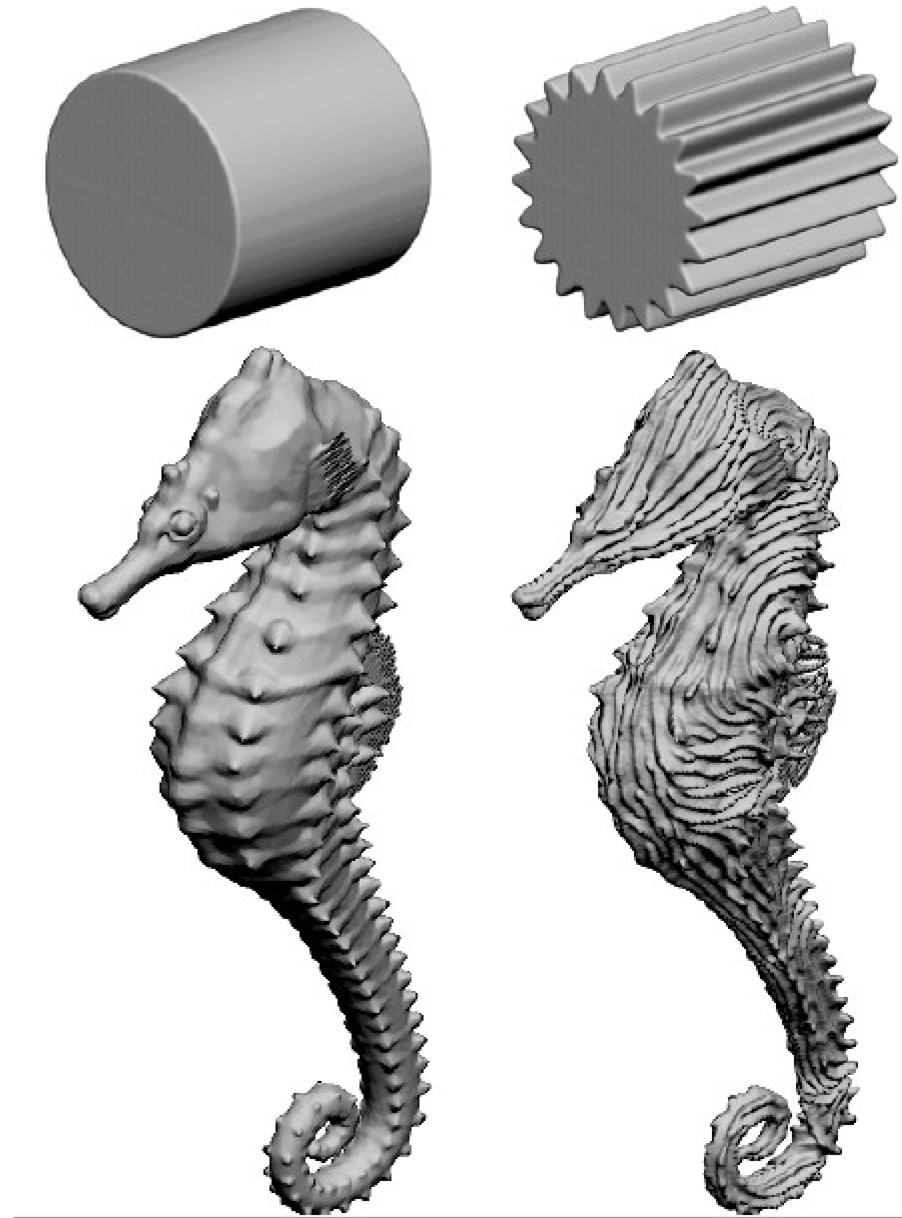


Neighborhood of size 5, 11, 15, 23



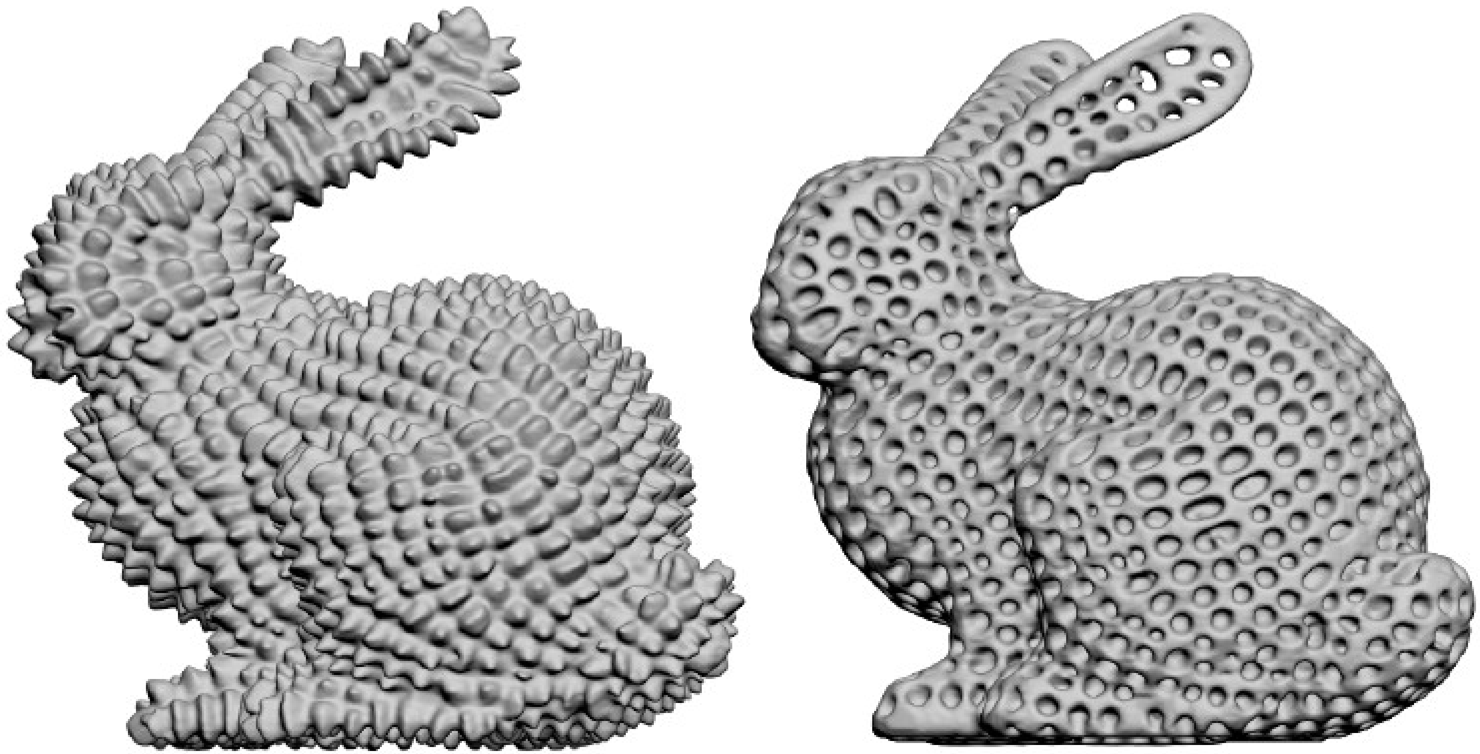


Volumetric illustration

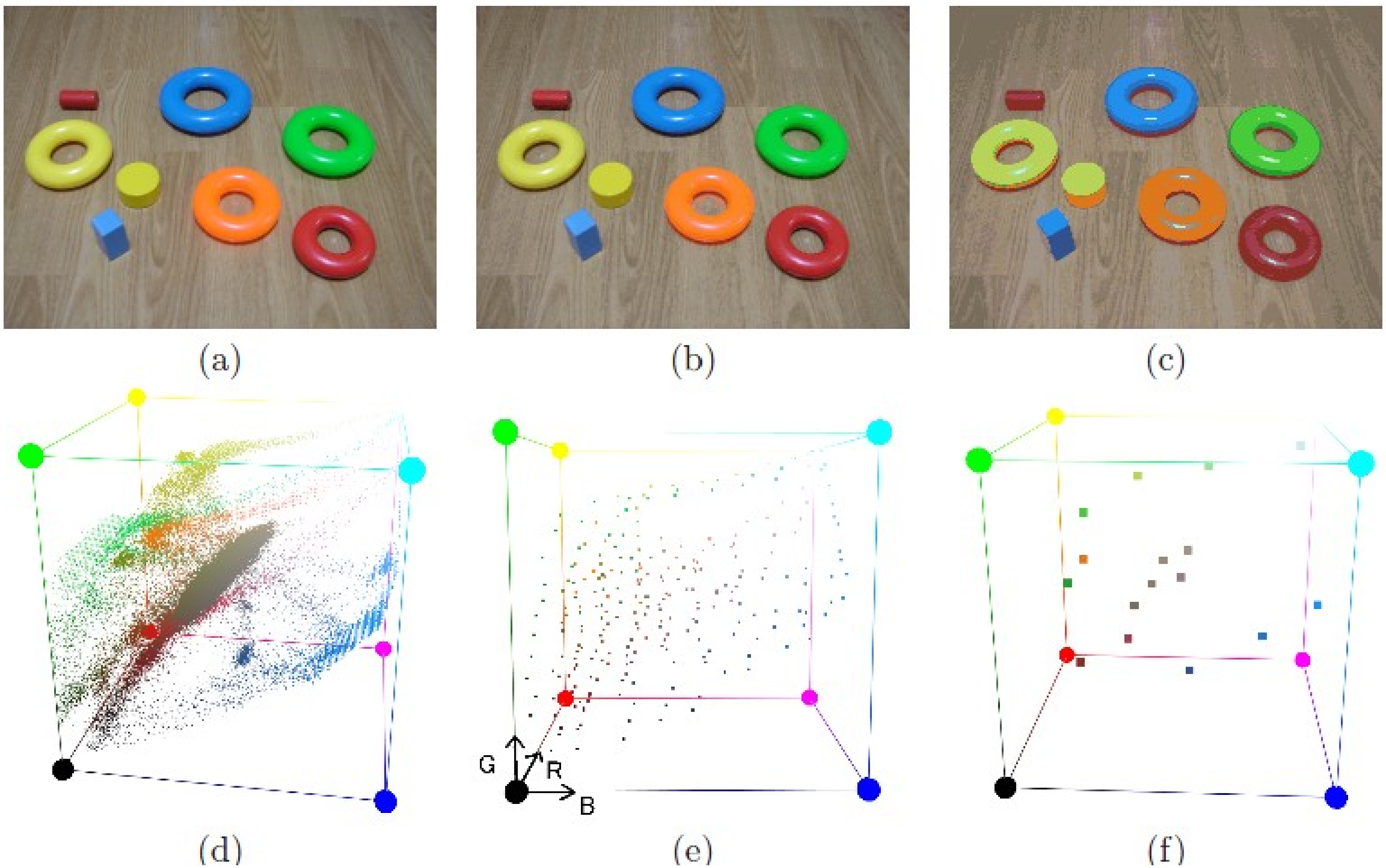


Geometry synthesis

Geometry synthesis:



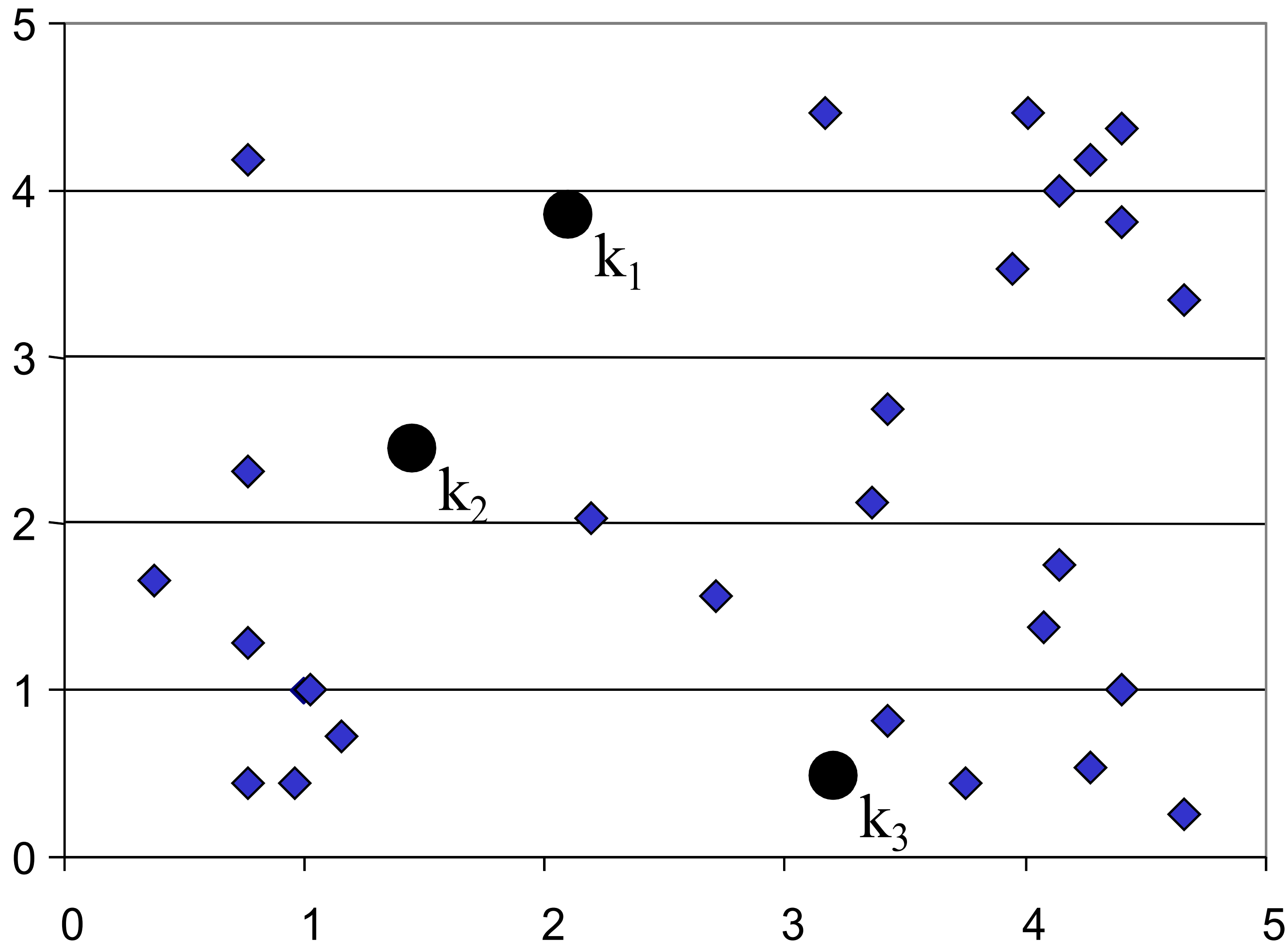
Clustering:: Application:: Color quantization



Vector quantization, codebook: Find centers in point sets

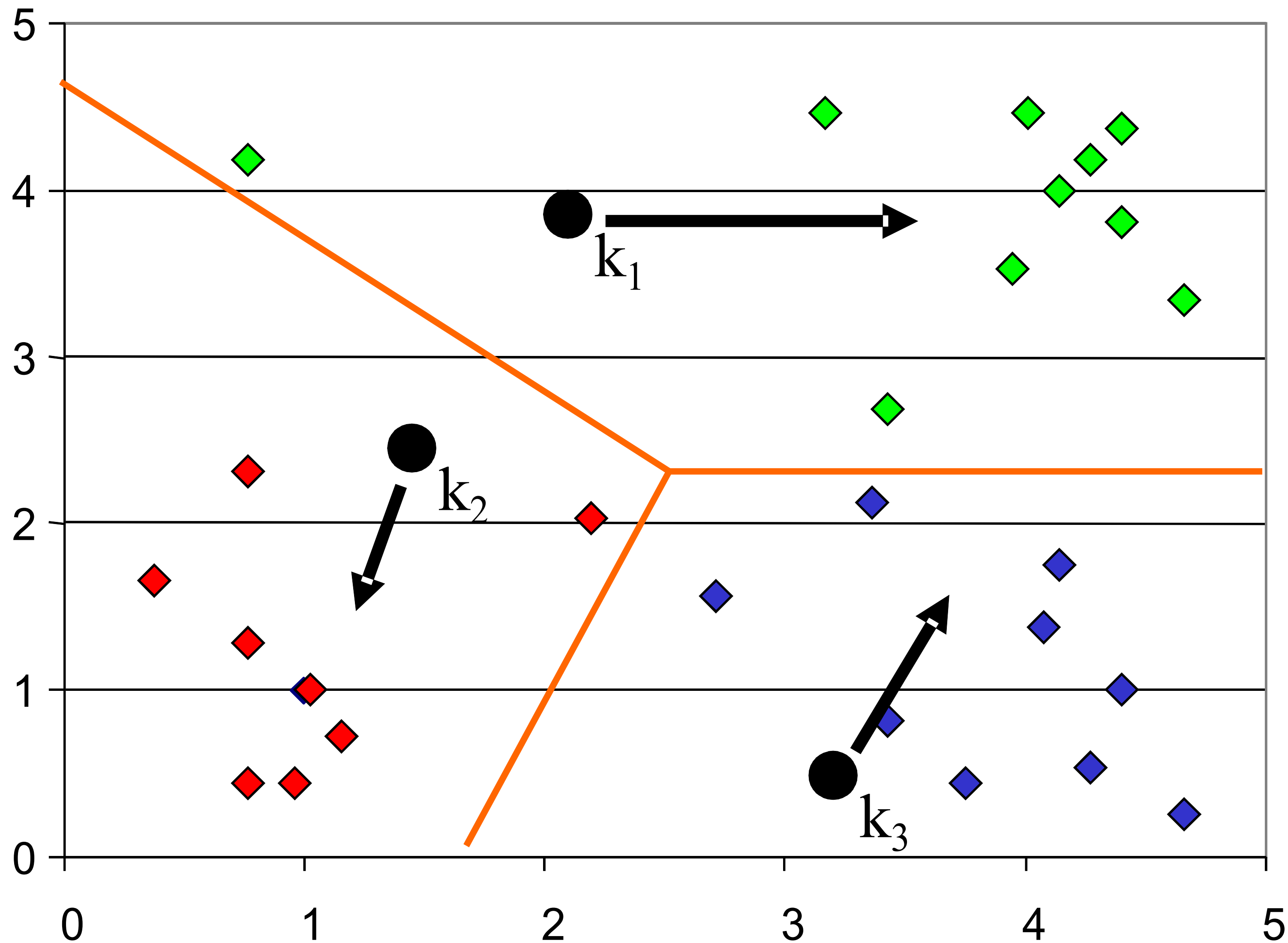
K-means Clustering: Step 1

N – points , 3 centers randomly chosen



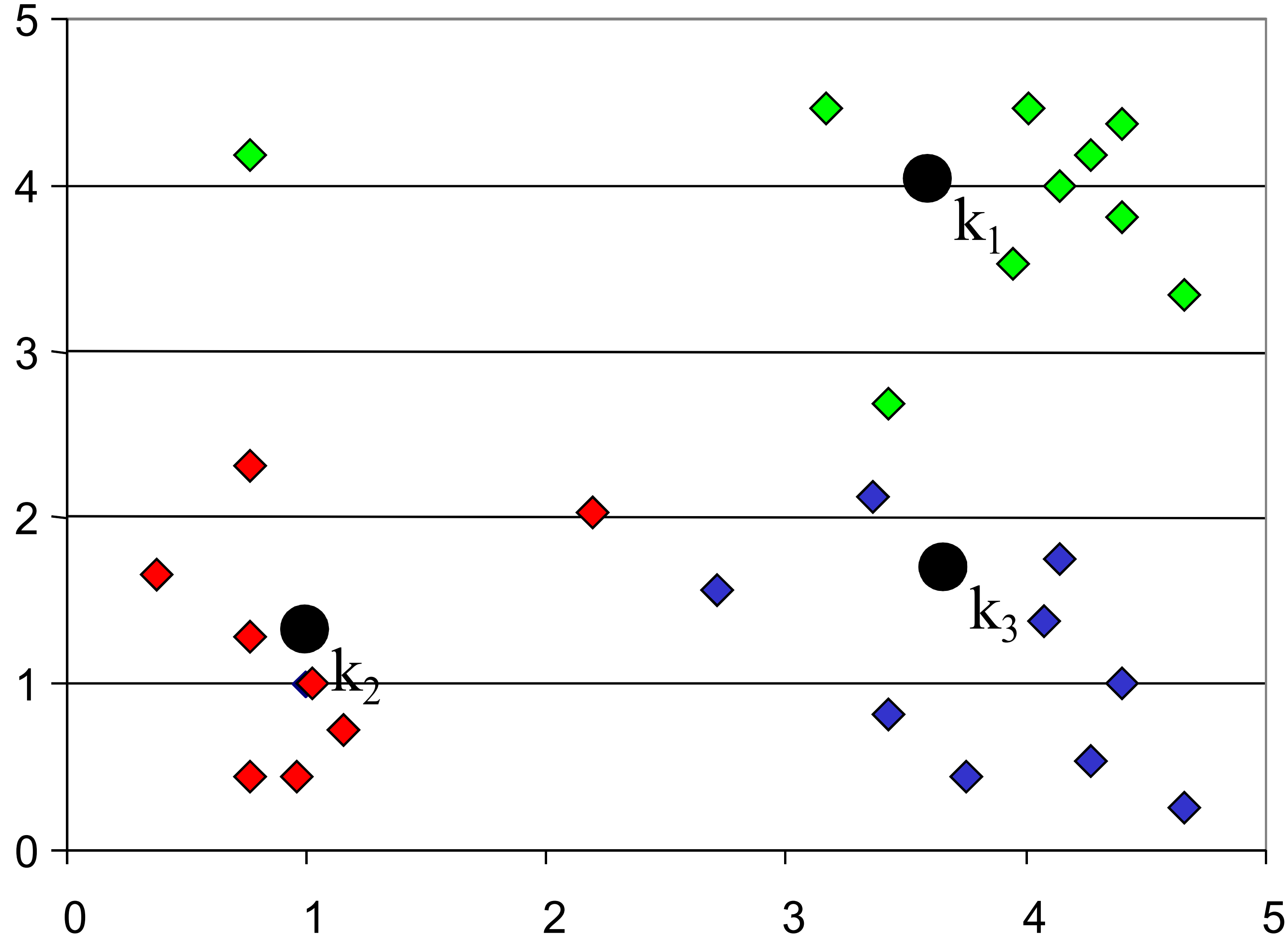
K-means Clustering: Step 2

Notice that the 3 centers divide the space into 3 parts



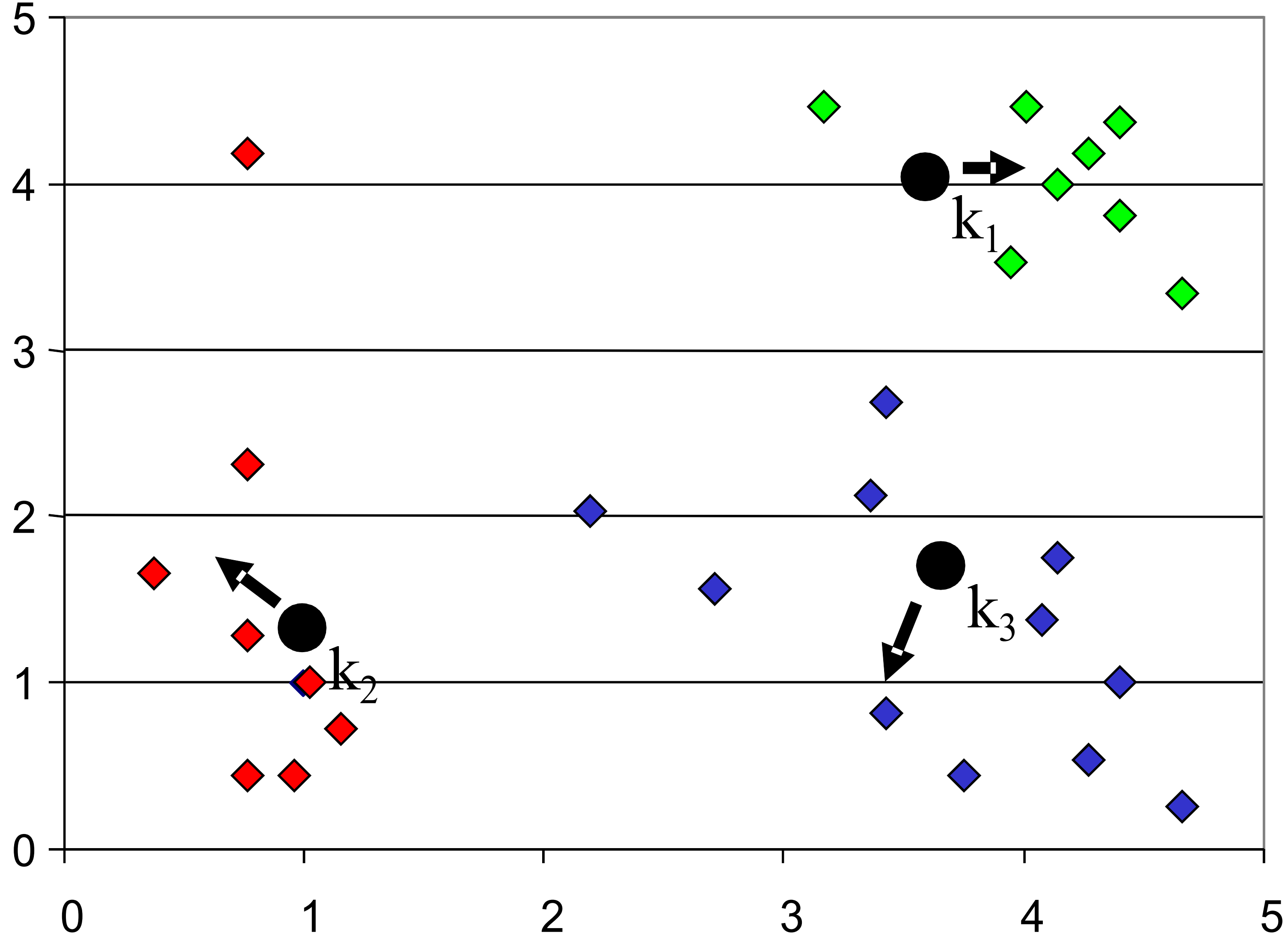
K-means Clustering: Step 3

New centers are calculated according to the instances of each K.



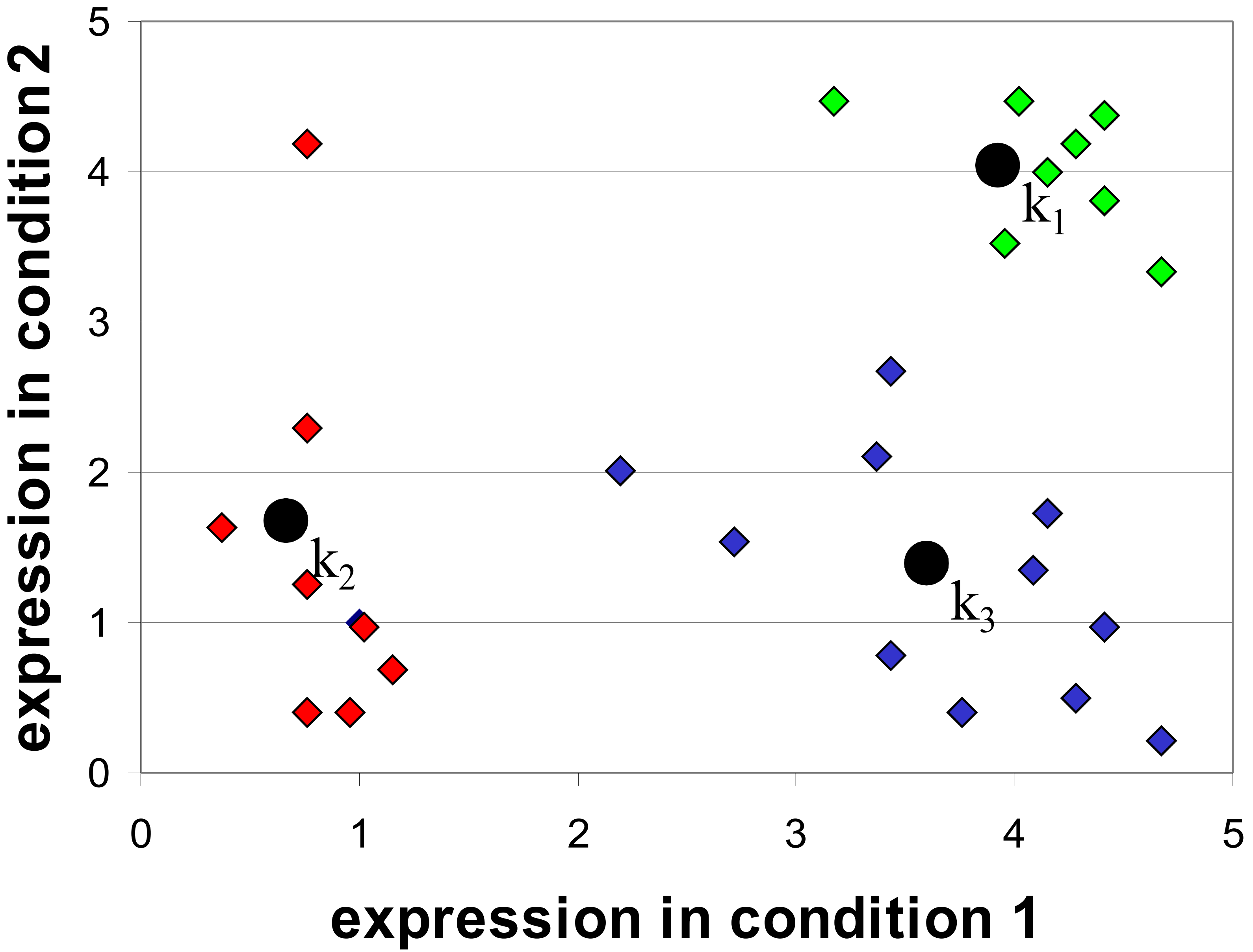
K-means Clustering: Step 4

Classifying each point to the new calculated K.



K-means Clustering: Step 5

After classifying the points to previous K vector , calculating new one



K-means Clustering

kMeans($\mathcal{P}, \epsilon$)

◁ Cluster points of $\mathcal{P}$ using kMeans ▷
◁ ϵ: threshold criterion to decide whether to stop or not ▷
Initialize centroids $\mathcal{C}$
while Total centroid displacements is less than threshold ϵ
    do ◁ Allocate points to clusters (hard membership) ▷
        for $i \leftarrow 1$ **to** n
            do $C(\mathbf{p}_i) = \mathrm{argmin}_{j=1}^{k} ||\mathbf{p}_i \mathbf{c}_j||$
        for $i \leftarrow 1$ **to** k
            do ◁ Update centroids to the center of mass of clusters ▷
                $\mathcal{C}(\mathbf{c}_i) = \{\mathbf{p} \in \mathcal{P} \mid C(\mathbf{p}) = i\}$
                $\mathbf{c}_i = \mathrm{CenterOfMass}(\mathcal{C}(\mathbf{c}_i))$

Centroid initialization:

- Forgy = Choose random seeds
- Draw seeds according to distance distribution:
  Careful seeding kmeans++

K-means Clustering: Color quantization

$$\mathcal{P} = \{\mathbf{p}_1, ..., \mathbf{p}_n\} \qquad \mathcal{C} = \{\mathbf{c}_1, ..., \mathbf{c}_k\}$$

points

clusters

$$\mathrm{MSE}(\mathcal{P}, \mathcal{C}) = \sum_{i=1}^{k} \sum_{j=1}^{n} w(j,i) ||\mathbf{p}_j - \mathbf{c}_i||^2$$

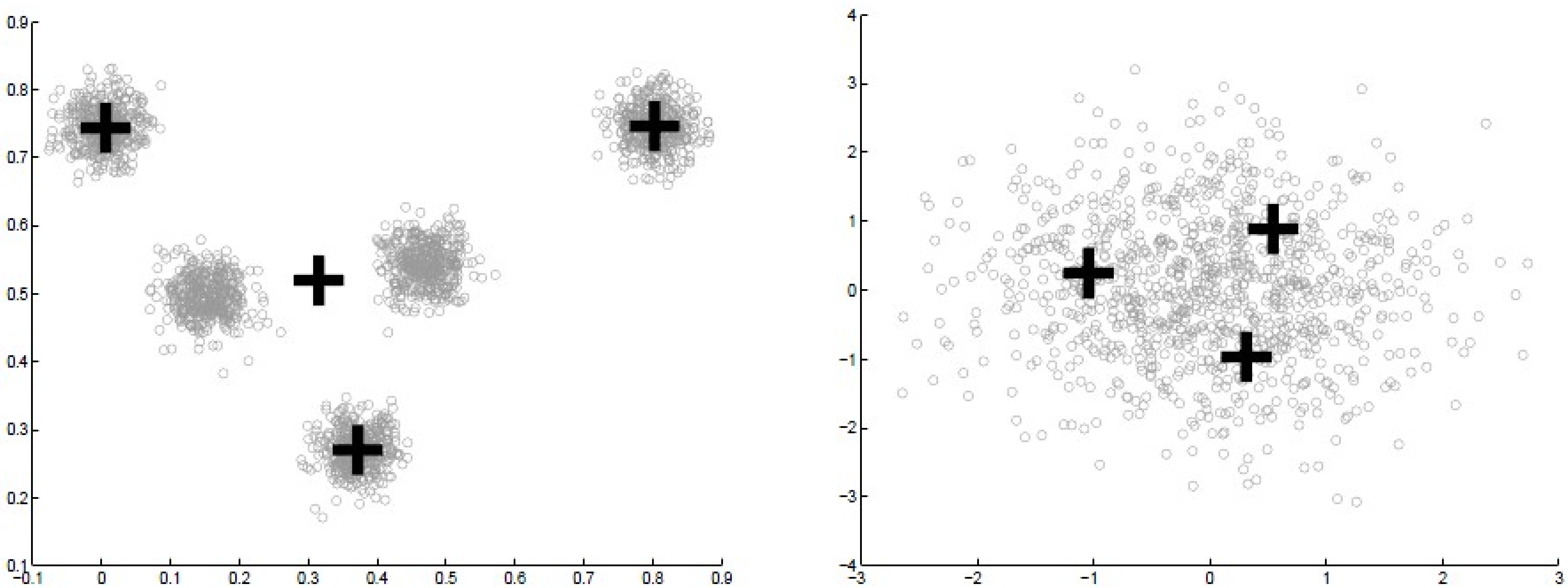
Hard/soft clustering $w(j,i) \geq 0, \qquad \sum_{i=1}^{k} w(j,i) = 1$

Lloyd k-means celebrated clustering algorithm:

$$\mathrm{MSE}(\mathcal{P}, \mathcal{C}) = \sum_{i=1}^{n} \min_{j=1}^{k} ||\mathbf{p}_i - \mathbf{c}_j||^2$$

K-means Clustering

- K means **monotanically** converges to a local minimum
- Learning the k in k-means



Improper seed numbers

Learning the K in G-means Clustering

Algorithm 1 G-means(X, α)

1: Let C be the initial set of centers (usually $C \leftarrow \{\bar{x}\}$).
2: $C \leftarrow kmeans(C, X)$.
3: Let $\{x_i | \text{class}(x_i) = j\}$ be the set of datapoints assigned to center c_j.
4: Use a statistical test to detect if each $\{x_i | \text{class}(x_i) = j\}$ follow a Gaussian distribution (at confidence level α).
5: If the data look Gaussian, keep c_j. Otherwise replace c_j with two centers.
6: Repeat from step 2 until no more centers are added.

Anderson-Darling test
for testing whether reals are from a Gaussian distribution:

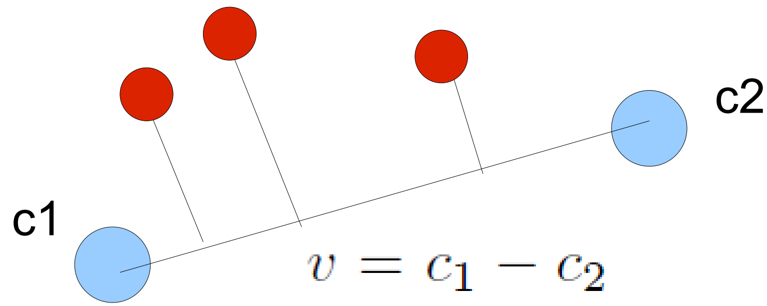
$$A^2(Z) = -\frac{1}{n}\sum_{i=1}^{n}(2i-1)\left[\log(z_i) + \log(1 - z_{n+1-i})\right] - n$$

Test for 1D values

$$A^2_*(Z) = A^2(Z)(1 + 4/n - 25/(n^2))$$

Compare this value with a confidence threshold alpha

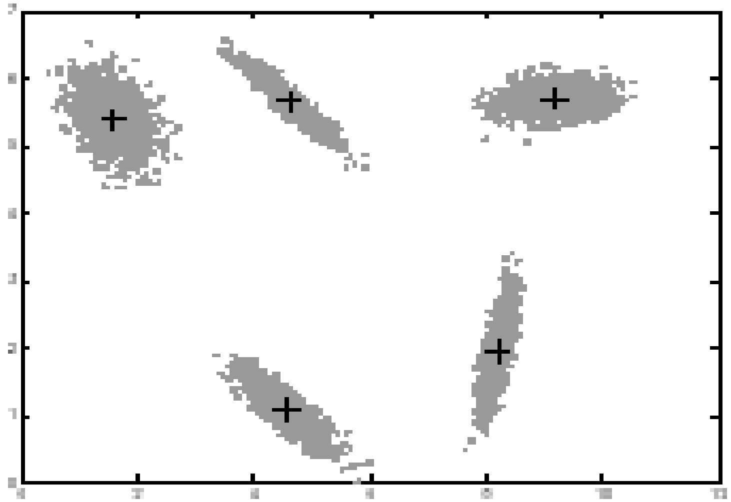
Learning the K in G-means Clustering



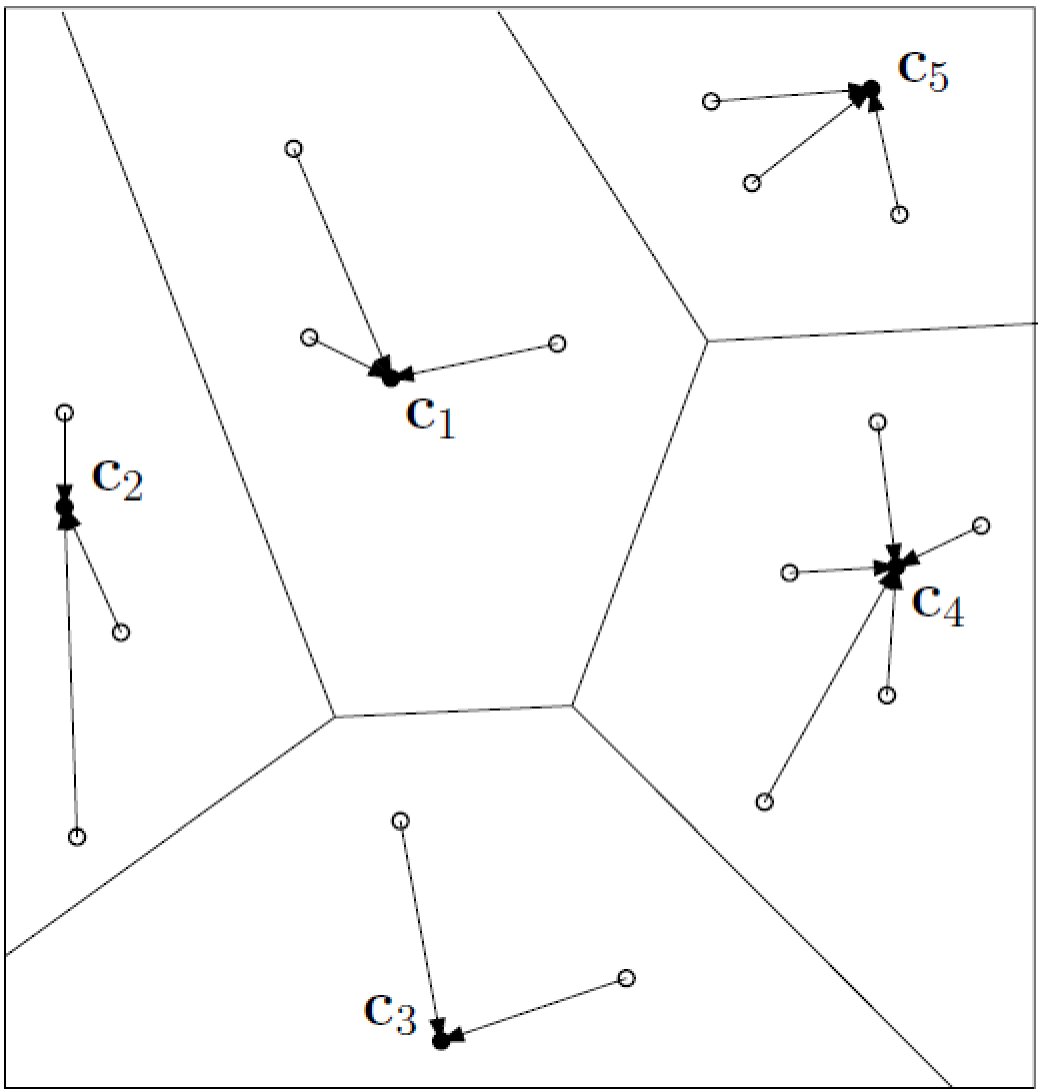
Project (orthogonally) points onto the line linking the two centroids
Sort them
Transform to mean 0 and variance 1.
Perform Anderson-Darling test

$$v = c_1 - c_2$$

$$x'_i = \langle x_i, v \rangle / ||v||^2$$

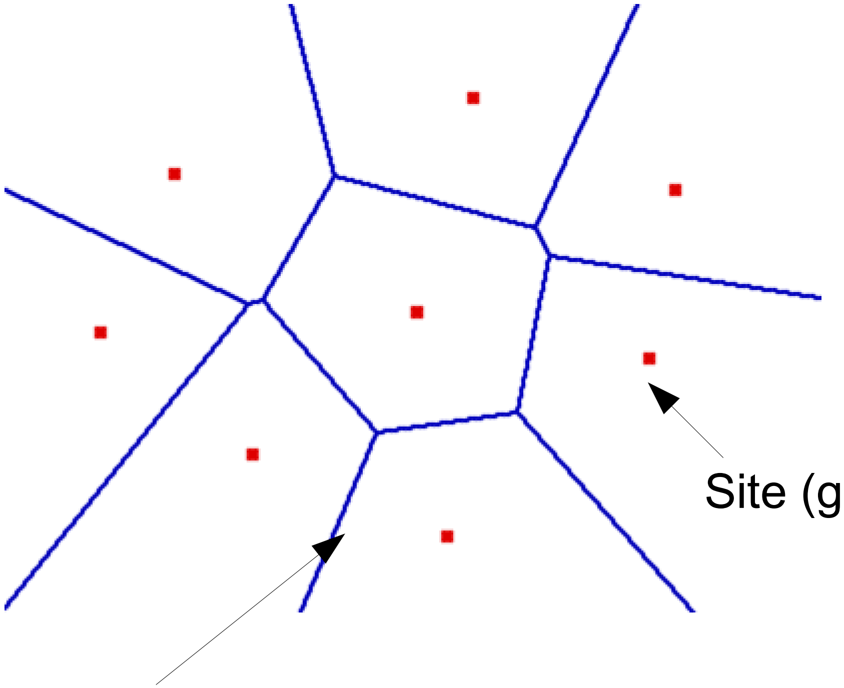


K-means Clustering & Voronoi diagrams



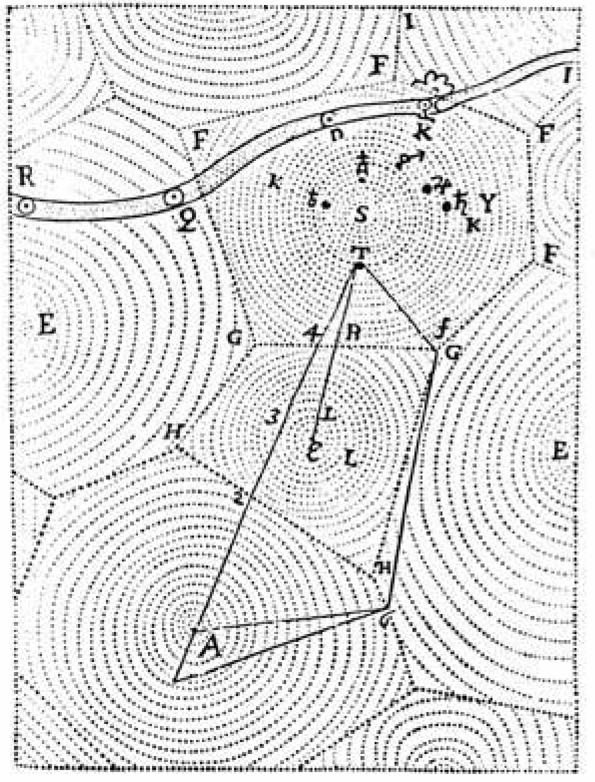
Facility locations

Voronoi diagrams



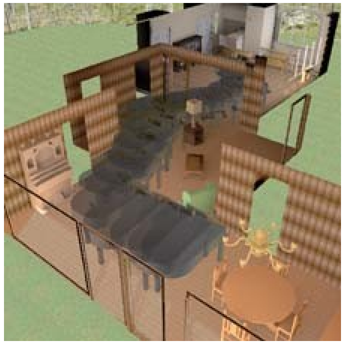
Site (generator)

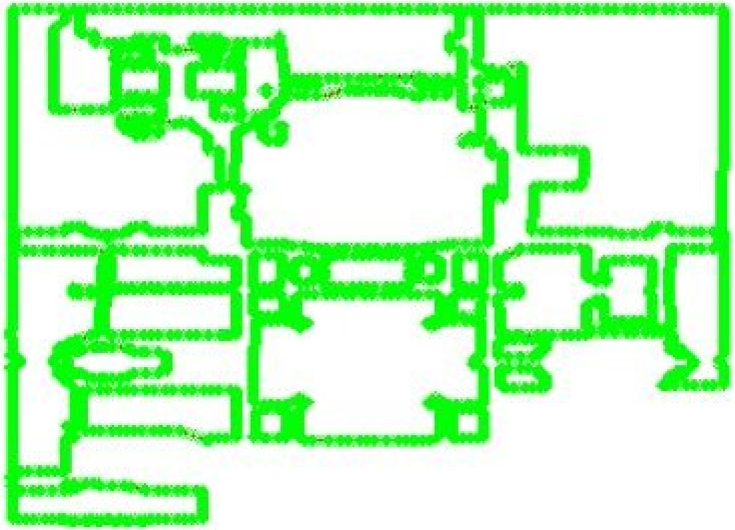
Bisector

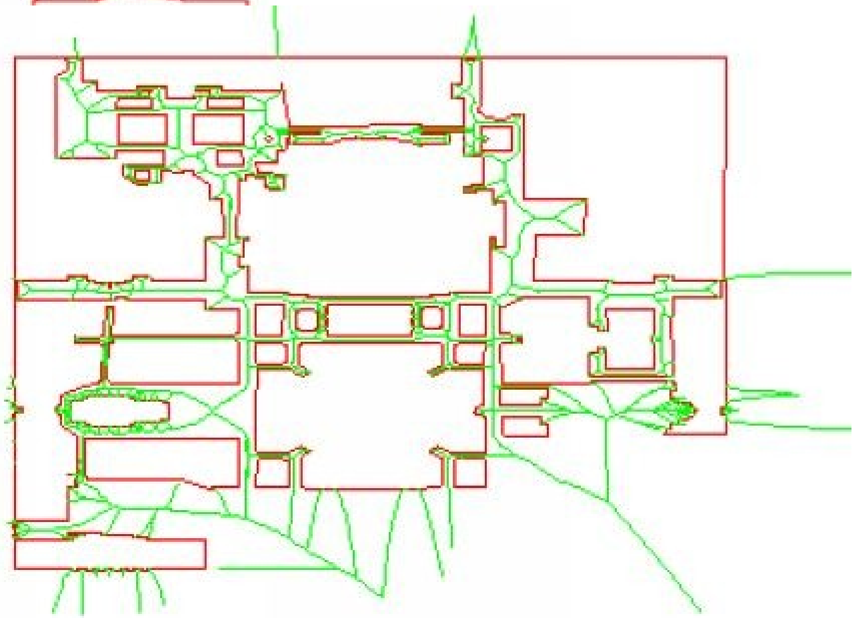


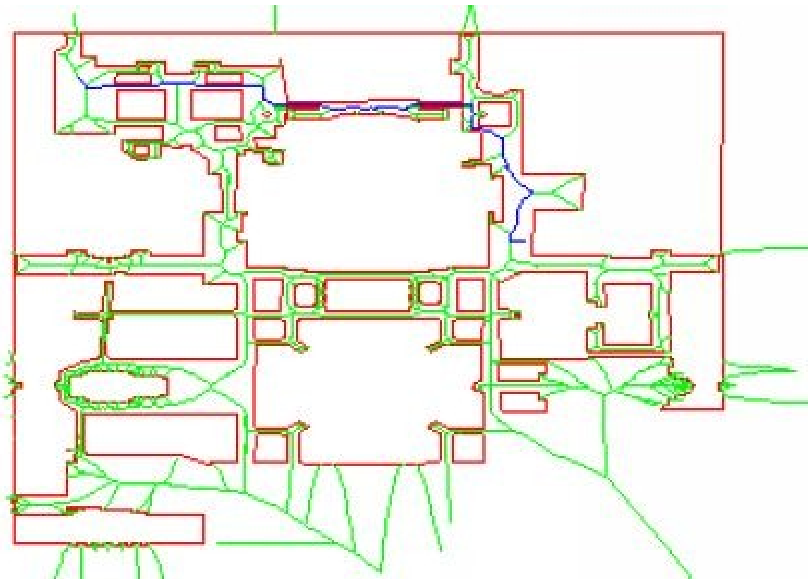
Descartes

Voronoi diagrams:
Piano mover problem
Robotics: path planning

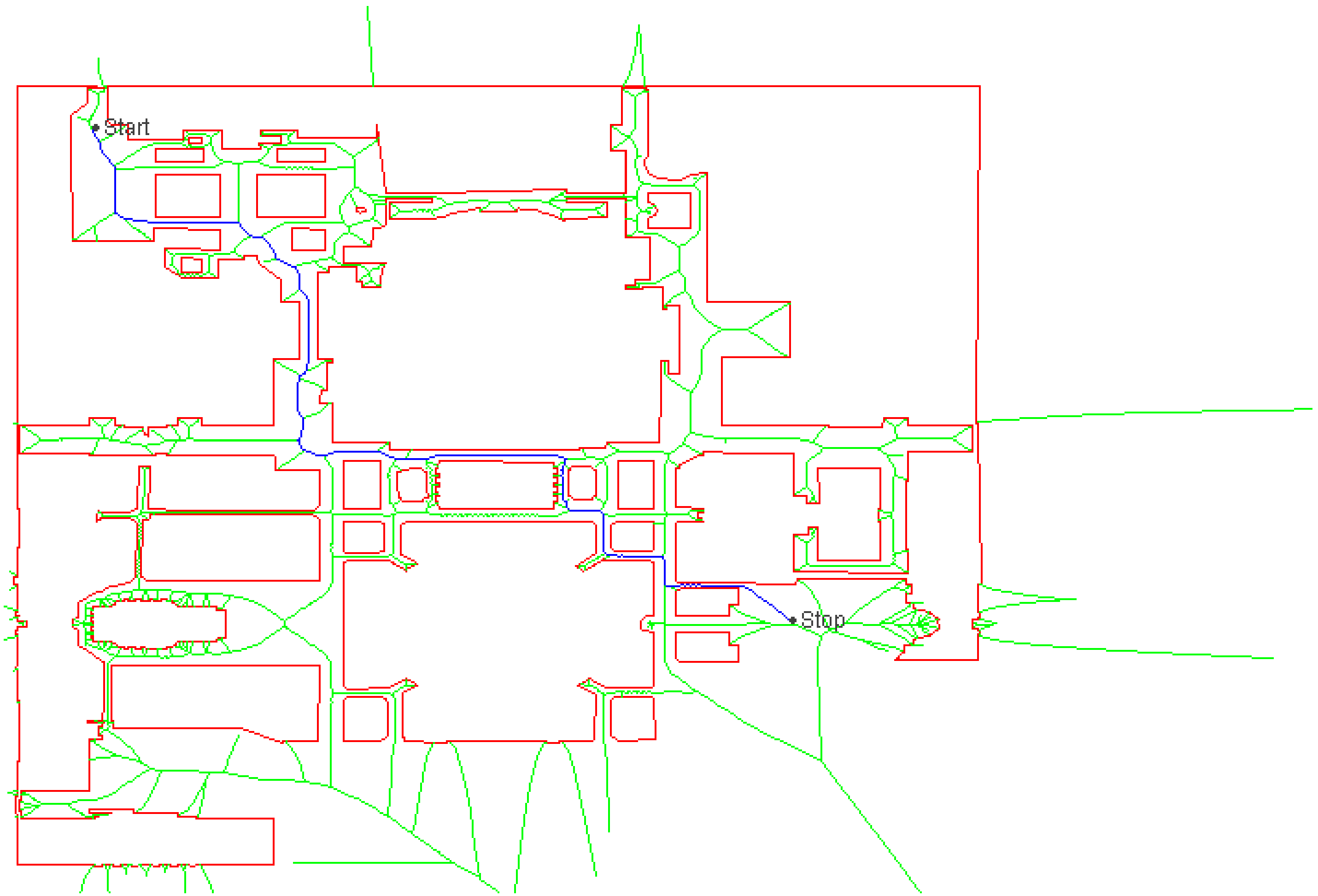








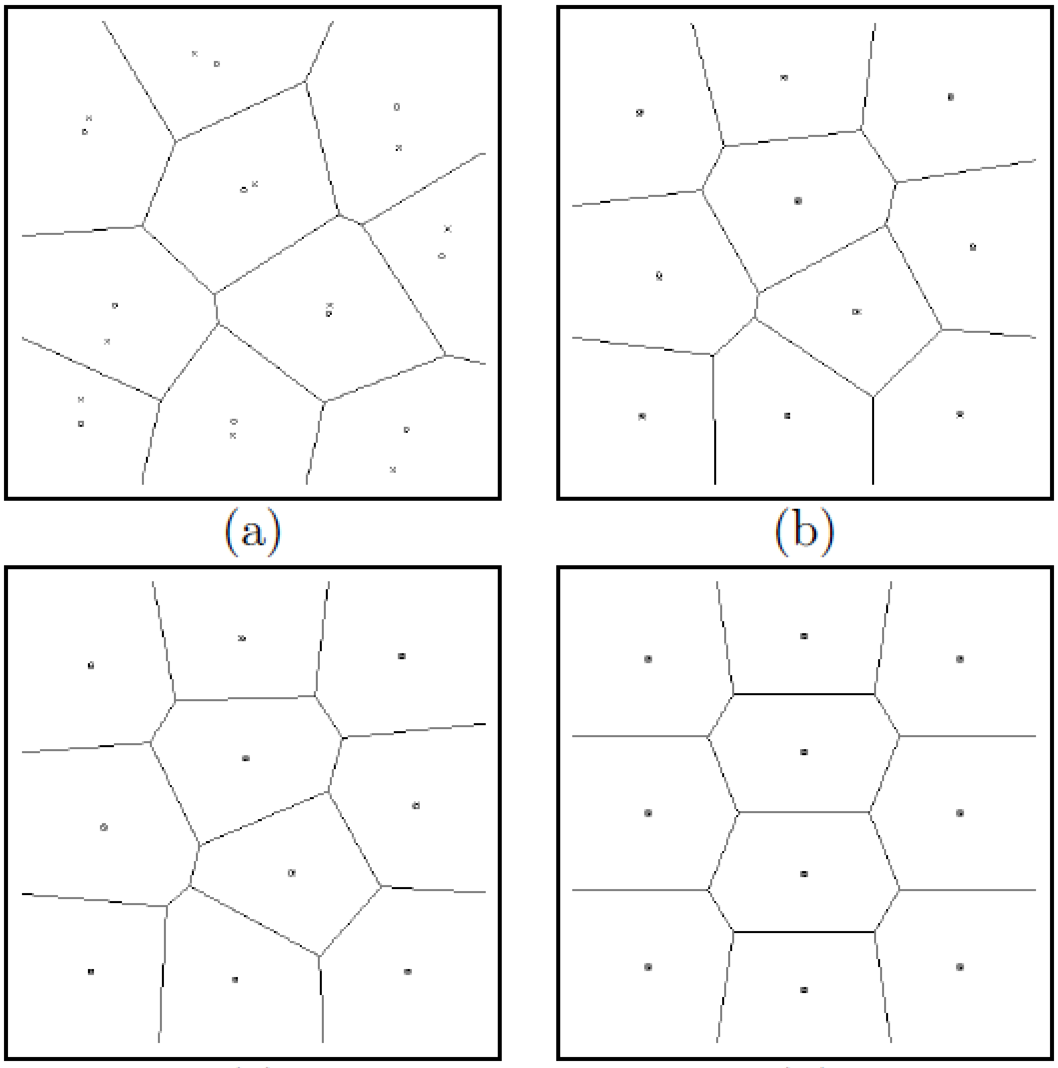
path planning



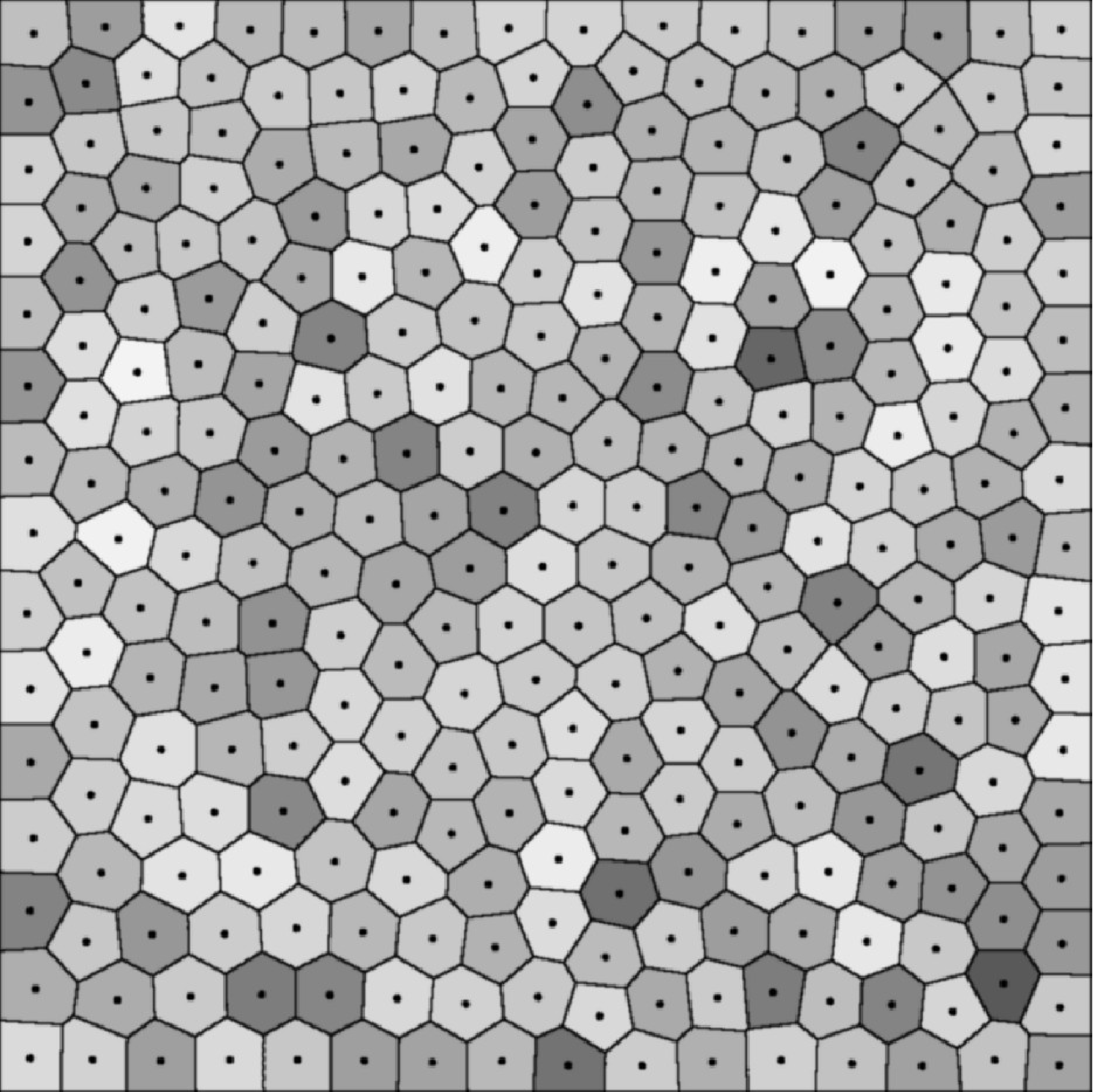
Applet at:
http://www.cs.columbia.edu/~pblaer/projects/path_planner/

CENTROIDALVORONOI($\mathcal{C}, \epsilon$)

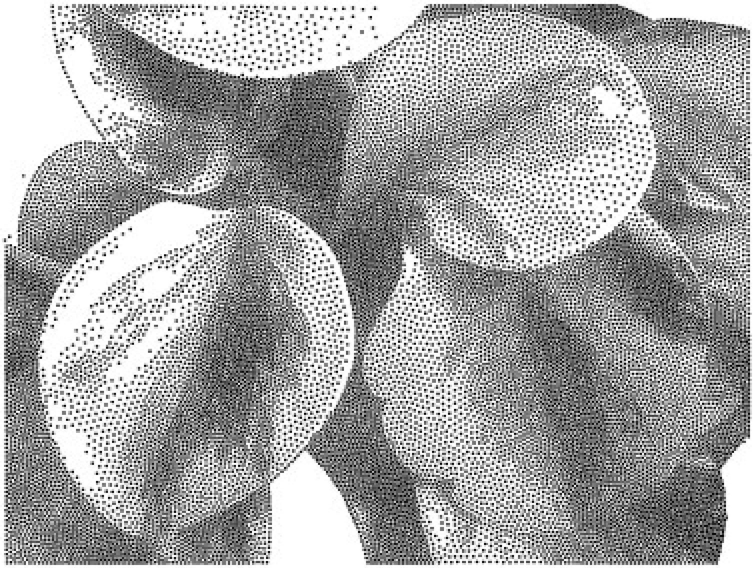
1. ◁ Compute k points evenly distributed on a spatial domain ▷
2. ◁ ϵ: threshold criterion to decide whether to stop or not ▷
3. Initialize centroids $\mathcal{C}$
4. **while** Total centroid displacements less than ϵ
5.   **do** Compute Voronoi diagram of $\mathcal{C}$
6.   Allocate each $\mathbf{c}_i$ to the center of mass of its Voronoi cell



Centroidal Voronoi diagram

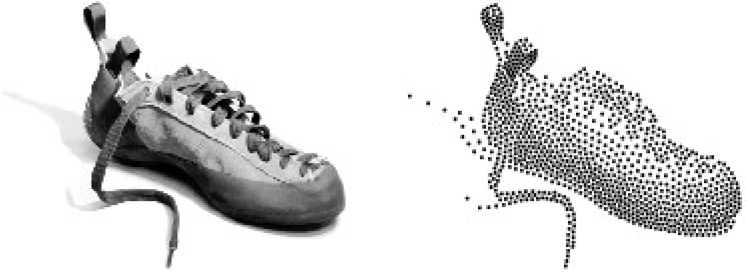


Stippling with Centroidal Voronoi diagrams



Incorporate a density function

$$\mathbf{C}_i = \frac{\int_A \mathbf{x}\rho(\mathbf{x})\,dA}{\int_A \rho(\mathbf{x})\,dA}$$

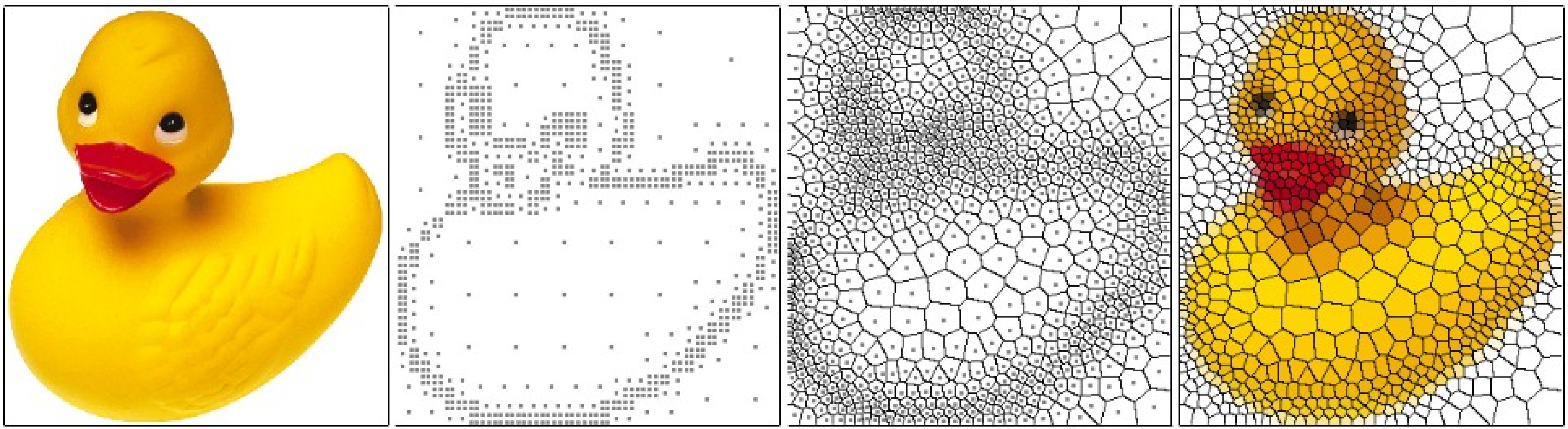


NPAR, 2002

NPR= Non Photorealistic Rendering (NPAR conference)

http://www.mrl.nyu.edu/~ajsecord/npar2002/html/stipples-node3.html

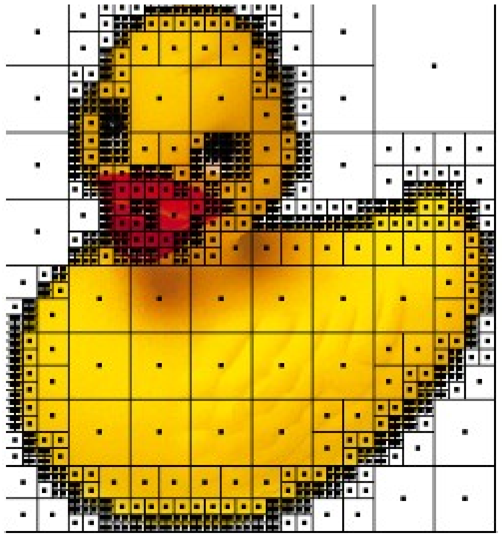
Centroidal Voronoi diagrams:
Adaptive mosaicing effect (2005)



1. Sample the image adaptively, finding a number of seed points. (center of **quad-tree** cells)

2. Compute the centroidal Voronoi diagram of the seeds, using a density map computed from the original image.

3. Paint each Voronoi cell.



$$z = \frac{\int_V x\mu(x)\,dx}{\int_V \mu(x)\,dx}$$

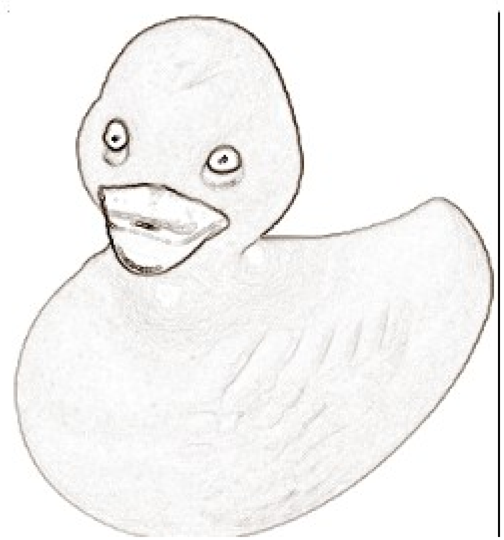
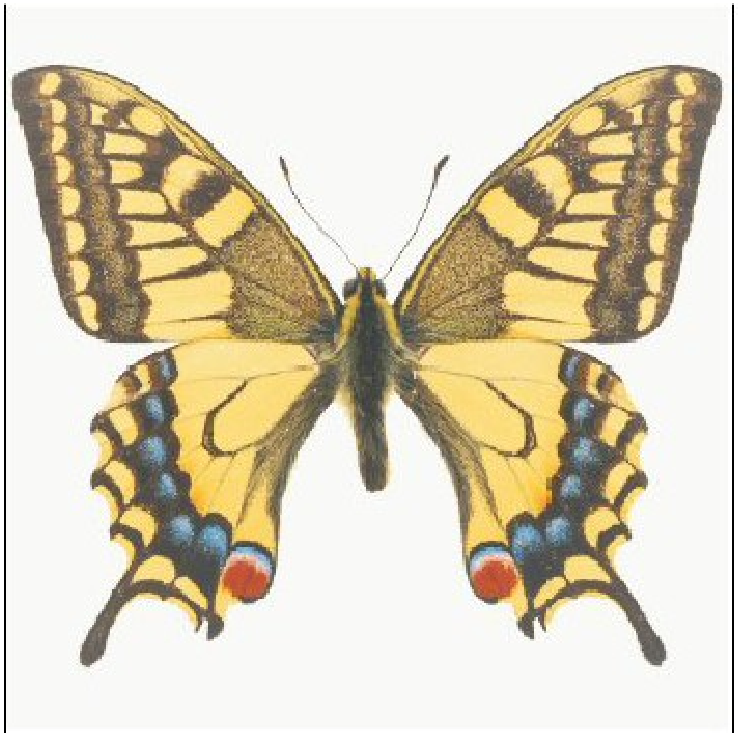
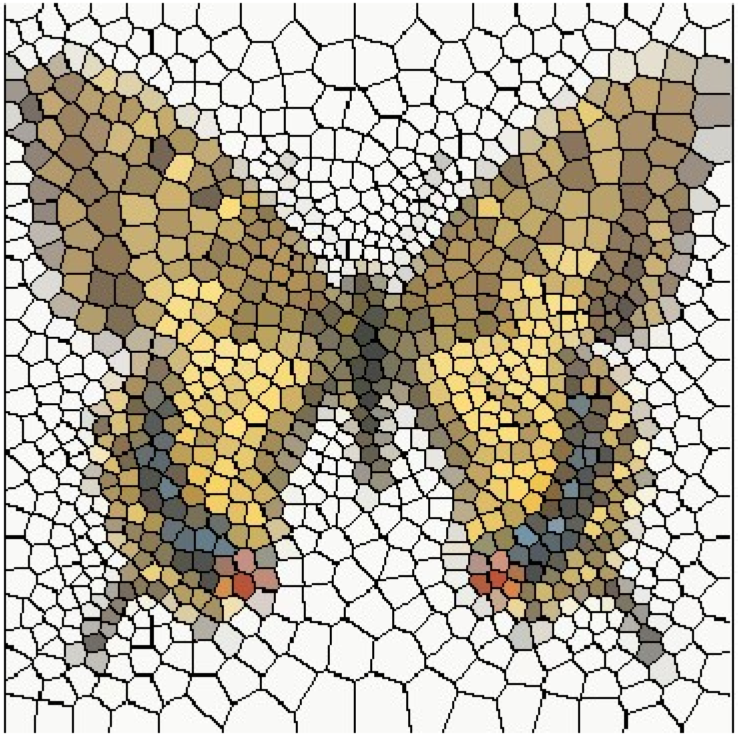
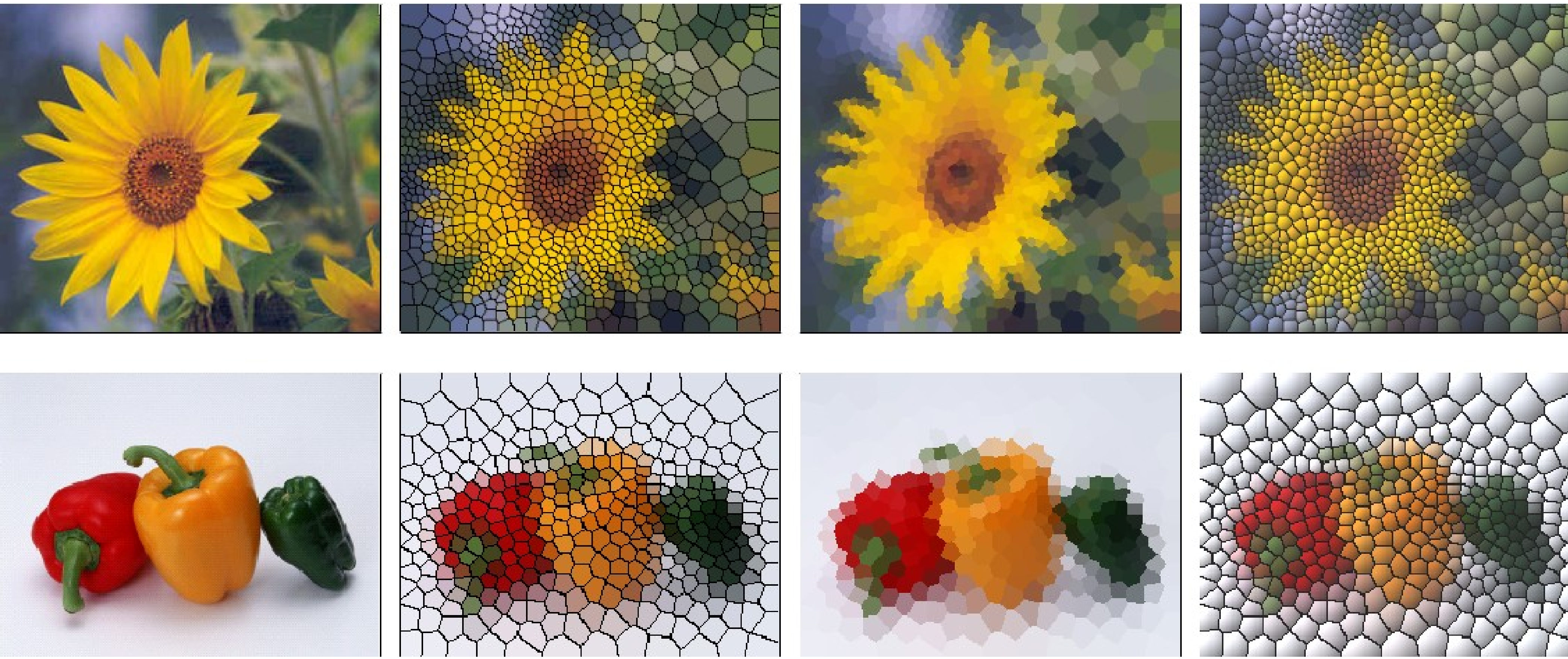


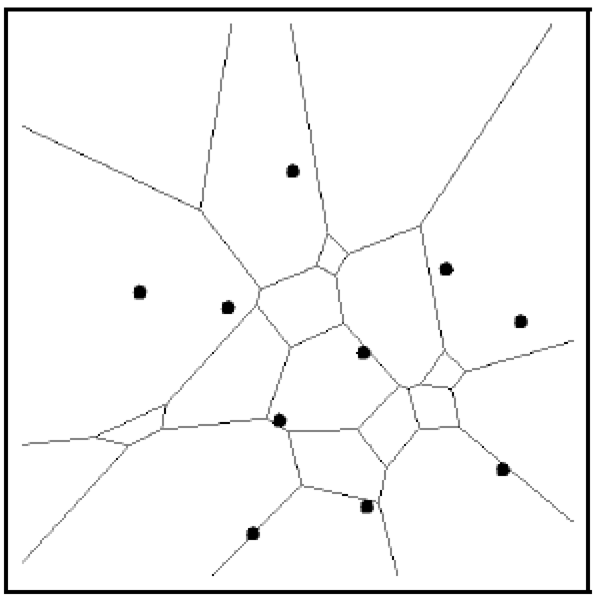
Image gradient as a density function

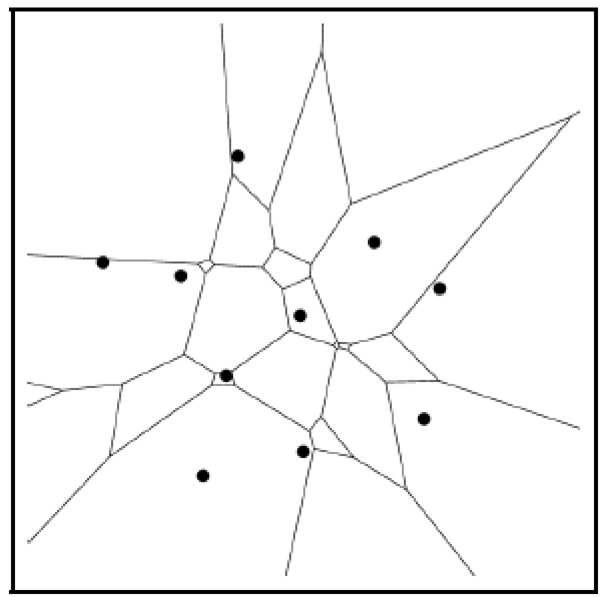
Various coloring effects of Voronoi cells



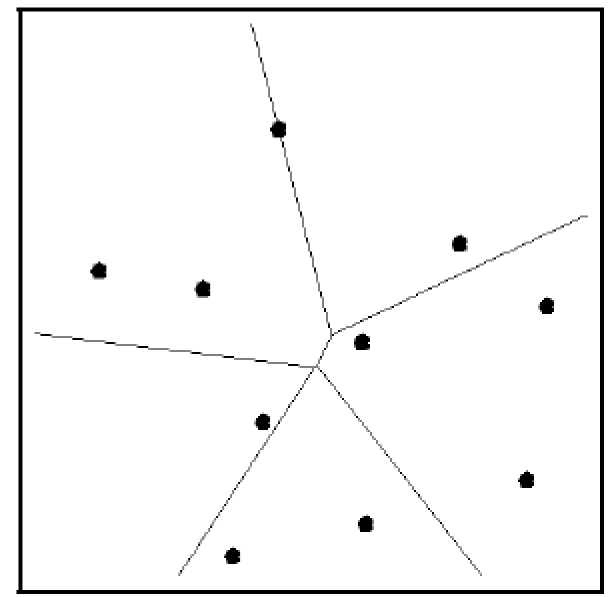
K-order Voronoi diagrams
Affine Voronoi diagrams



Order 2



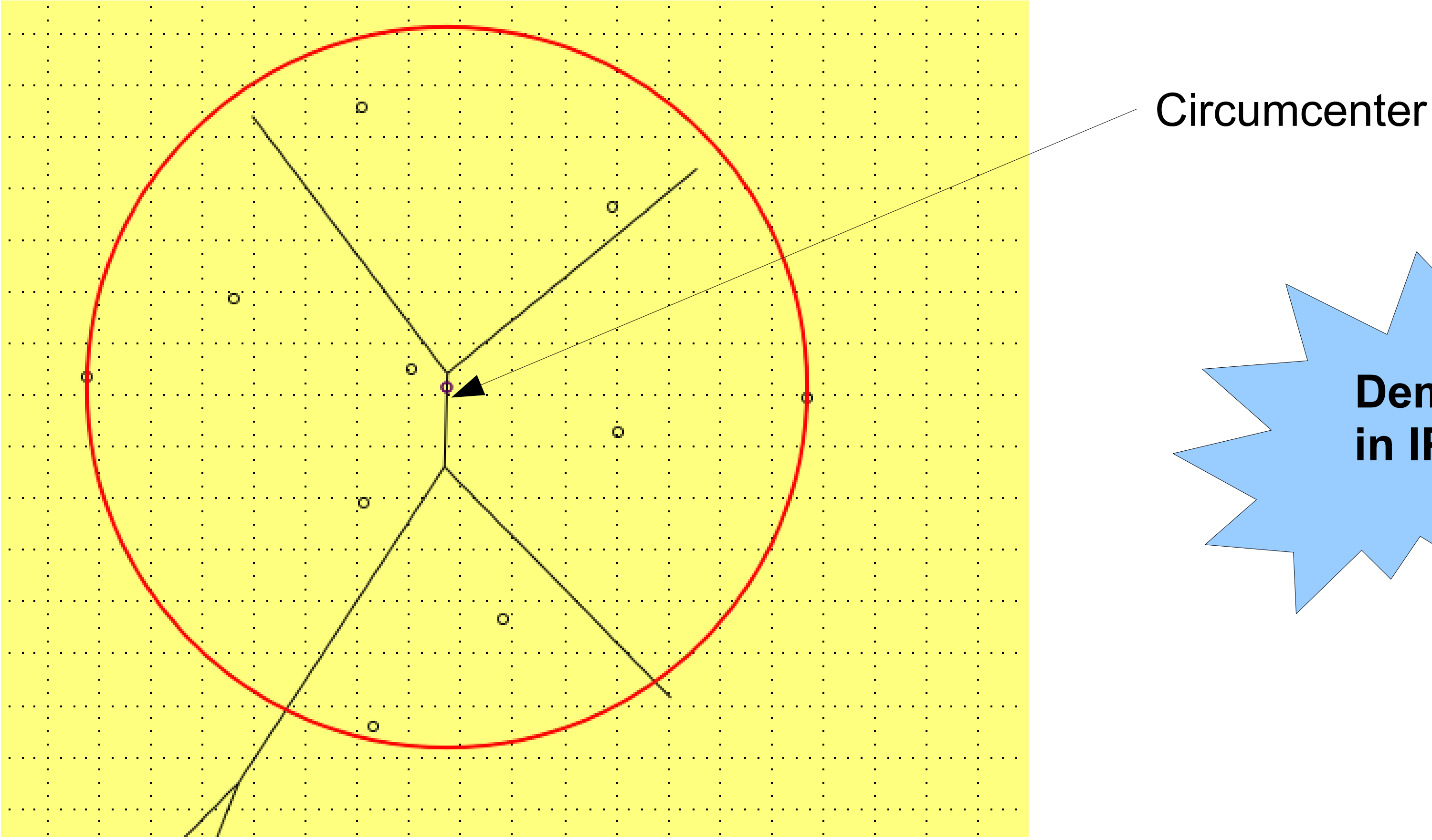
Order 3



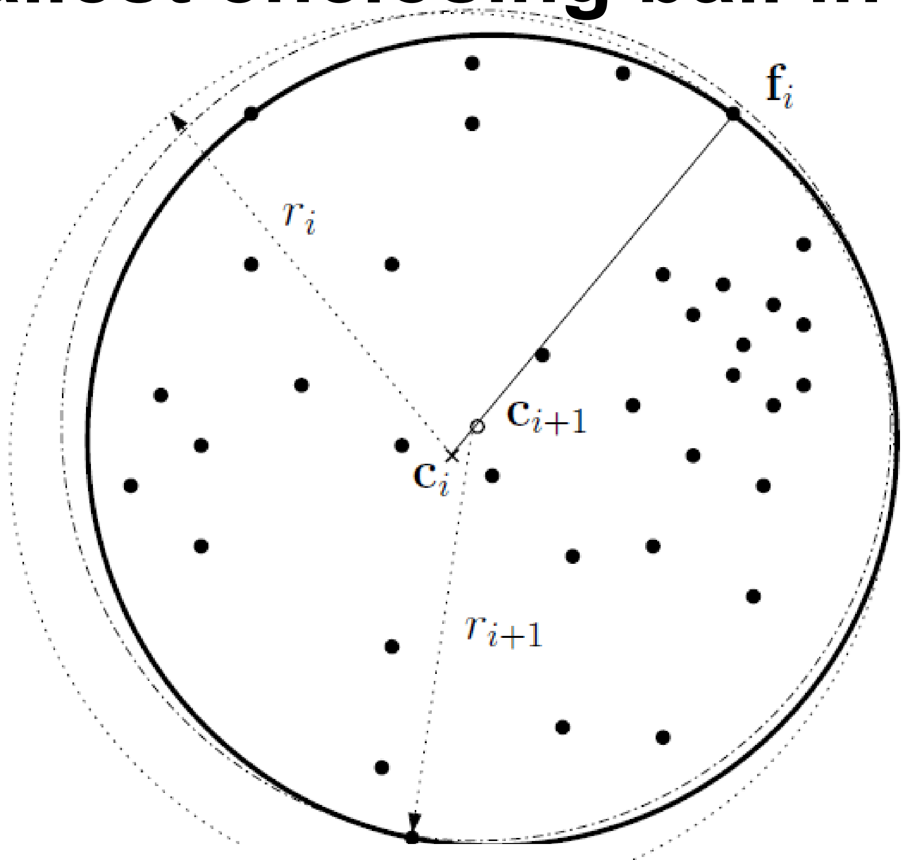
Order n-1
Farthest Voronoi diagram

Furthest Voronoi diagram and smallest radius enclosing ball

The center of the smallest enclosing ball (min max) is necessarily located at the furthest Voronoi diagram



Approximating the smallest enclosing ball in very large dimension

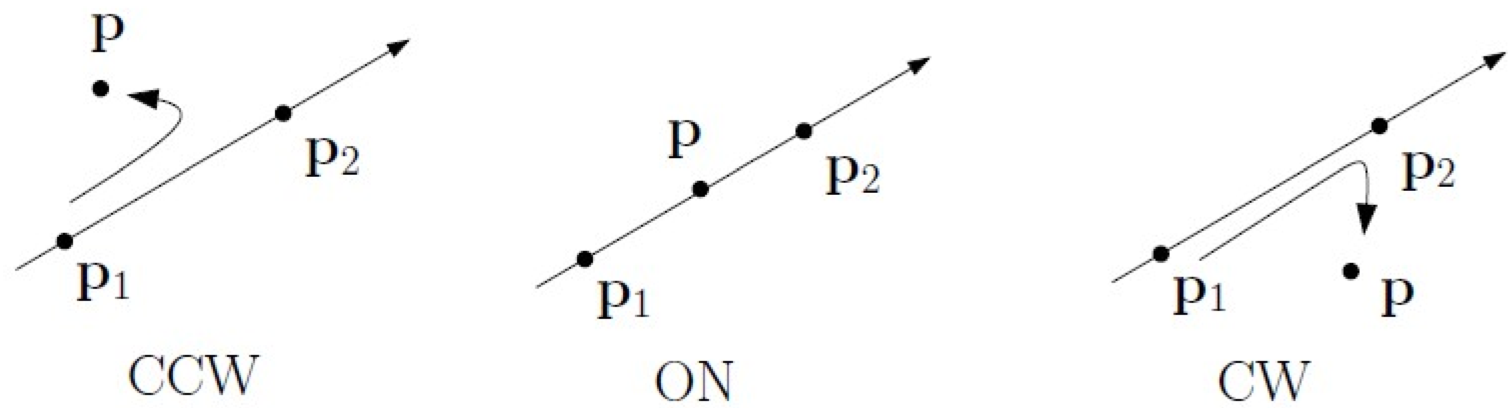


SmallEnclosingBall($\mathbf{p_1}, ..., \mathbf{p_n}, \epsilon$)

◁ Compute a $(1+\epsilon)$-approximation of the smallest enclosing ball ▷
◁ Return the circumcenter of a small enclosing ball ▷
$\mathbf{c} \leftarrow \mathbf{p_1}$
for $i \leftarrow 1$ **to** $\lceil \frac{1}{\epsilon^2} \rceil$
    do ◁ Furthest point is $\mathbf{f}_i = \mathbf{p}_j$ ▷
        $j = \mathrm{argmax}_{i=1}^{n} ||\mathbf{c}\mathbf{p_i}||$
        $\mathbf{c} \leftarrow \mathbf{c} + \frac{1}{i+1}\mathbf{c}\mathbf{p}_j$
return c

Designing predicates/Geometric axioms

Orient2D



$$\text{Orient2D}(\mathbf{p}, \mathbf{q}, \mathbf{r}) = \text{sign}\det \begin{bmatrix} 1 & 1 & 1 \\ \mathbf{p} & \mathbf{q} & \mathbf{r} \end{bmatrix}$$

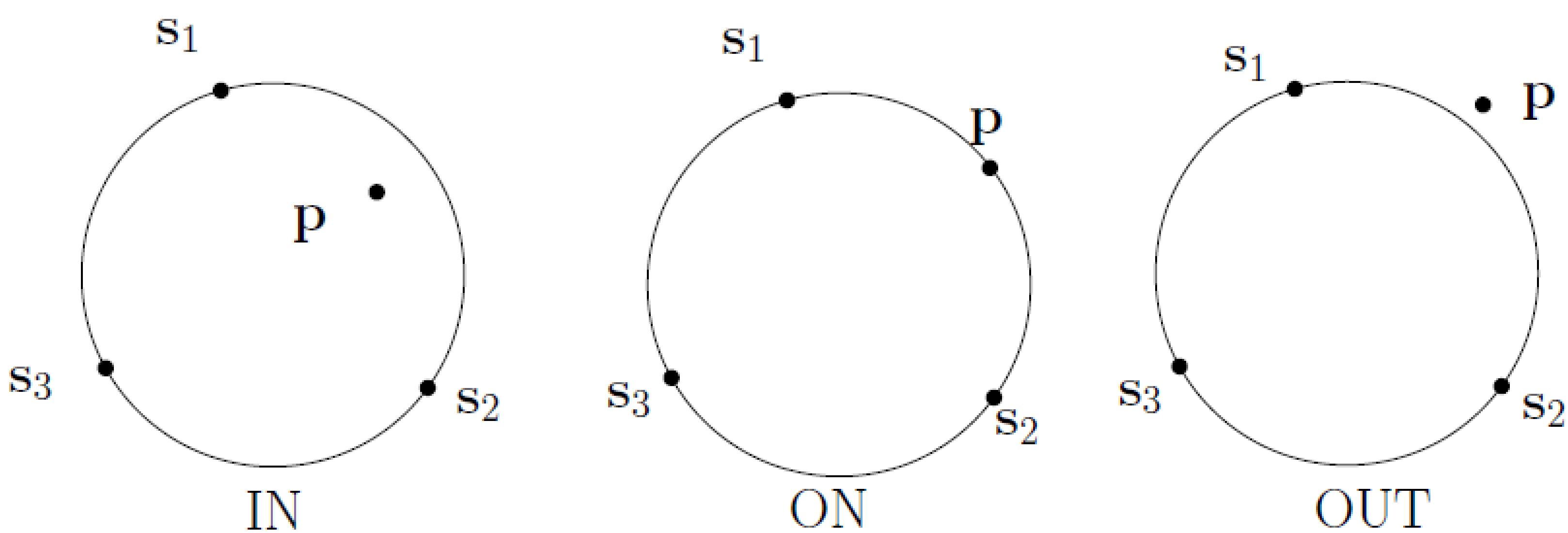
$$\text{Orient2D}(\mathbf{p}, \mathbf{q}, \mathbf{r}) = \text{sign}\det \begin{bmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{bmatrix}$$

$$\text{OrientdD}(\mathbf{p}_1, ..., \mathbf{p}_d, \mathbf{p}) = \text{sign}\det \begin{bmatrix} \mathbf{p}_1^T & 1 \\ \mathbf{p}_2^T & 1 \\ \vdots & 1 \\ \mathbf{p}_d^T & 1 \\ \mathbf{p}^T & 1 \end{bmatrix}$$

Determinant=Signed area of the triangle formed by the 3 points

Designing predicates/Geometric axioms

InSphere2D



$$\text{InSpheredD}(\mathbf{s}_1, ..., \mathbf{s}_{d+1}, \mathbf{p}) = \text{sign}\det \begin{bmatrix} \mathbf{s}_1^T & \mathbf{s}_1 \cdot \mathbf{s}_1 & 1 \\ \mathbf{s}_2^T & \mathbf{s}_2 \cdot \mathbf{s}_2 & 1 \\ \vdots & \vdots & 1 \\ \mathbf{s}_{d+1}^T & \mathbf{s}_{d+1} \cdot \mathbf{s}_{d+1} & 1 \\ \mathbf{p}^T & \mathbf{p} \cdot \mathbf{p} & 1 \end{bmatrix}$$