Fundamentals of 3D

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

Frank Nielsen
nielsen@lix.polytechnique.fr
Manipulating PBM/PPM/PGM images in Java

Monochrome bitmap pixels PBM (P1)

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 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 3 0 0 7 7 7 7 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 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
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

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) {}
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");
    }
}
Stochastic 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
``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

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

Source Image $I_s$

Target Image $I_t$

$\text{SSD}(x_s, y_s; x_t, y_t) = \sum_{l=-s}^{s} \sum_{c=-s}^{s} \text{LShape}(l, c) \left( I_s[x_s + c, y_s + l] - I_t[x_t + c, y_t + l] \right)^2$

$\left( x_s, y_s \right) = \arg\min_{(x,y) \in I_s} \text{SSD}(x, y; x_t, y_t).$
Stochastic texture synthesis

\textsc{Texturesynthesis}(I_s, I_t)
1. \(< I_s \text{ is the input texture sample } >\)
2. \(< \text{Create a large texture } I_t >\)
3. \(\text{Initialize a random color image } I_t\)
4. \(< \text{Synthesize pixels following the horizontal scanline order } >\)
5. \(\text{for } y \leftarrow 1 \text{ to } h_t\)
6. \(\text{do for } x \leftarrow 1 \text{ to } w_t\)
7. \(\text{do } (x_s, y_s) = \text{BestLSHAPEMatch}(I_s, x, y)\)
8. \(I_t[x, y] = I_s[x_s, y_s]\)
Linearization of neighborhood

\[
\begin{align*}
2s + 1 &= 5 \\
\begin{array}{cccc}
  c_{-2,-2} & c_{-1,-2} & c_{0,-2} & c_{1,-2} & c_{2,-2} \\
  c_{-2,-1} & c_{-1,-1} & c_{0,-1} & c_{1,-1} & c_{2,-1} \\
  c_{-2,0} & c_{-1,0} & & & \\
\end{array}
\end{align*}
\]

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

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

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

\[
\text{SSD}(x_s, y_s; x_t, y_t) = \|\mathbf{n}(x_s, y_s) - \mathbf{n}(x_t, y_t)\|^2.
\]
Impact of window size

Neighborhood of size 5, 11, 15, 23
Geometry synthesis

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

\texttt{kMeans}(\mathcal{P}, \epsilon) 
1. \texttt{\langle Cluster points of } \mathcal{P} \texttt{ using kMeans } \rangle 
2. \texttt{\langle } \epsilon \texttt{: threshold criterion to decide whether to stop or not } \rangle 
3. \texttt{Initialize centroids } \mathcal{C} 
4. \texttt{while Total centroid displacements is less than threshold } \epsilon 
5. \texttt{do } \texttt{\langle Allocate points to clusters (hard membership) } \rangle 
6. \texttt{for } i \leftarrow 1 \texttt{ to } n 
7. \texttt{do } C(p_i) = \text{argmin}_{j=1}^{k} \|p_i - c_j\| 
8. \texttt{for } i \leftarrow 1 \texttt{ to } k 
9. \texttt{do } \texttt{\langle Update centroids to the center of mass of clusters } \rangle 
10. \mathcal{C}(c_i) = \{p \in \mathcal{P} \mid C(p) = i\} 
11. \textit{c}_i = \text{CenterOfMass}(\mathcal{C}(c_i))

Centroid initialization:
• Forgy = Choose random seeds
• Draw seeds according to distance distribution:
  Careful seeding kmeans++
K-means Clustering: Color quantization

\[ \mathcal{P} = \{p_1, ..., p_n\} \]

\[ \mathcal{C} = \{c_1, ..., c_k\} \]

\[ \text{MSE}(\mathcal{P}, \mathcal{C}) = \sum_{i=1}^{k} \sum_{j=1}^{n} w(j, i) \|p_j - c_i\|^2 \]

Hard/soft clustering

\[ w(j, i) \geq 0, \quad \sum_{i=1}^{k} w(j, i) = 1 \]

Lloyd k-means celebrated clustering algorithm:

\[ \text{MSE}(\mathcal{P}, \mathcal{C}) = \sum_{i=1}^{n} \min_{j=1}^{k} \|p_i - c_j\|^2 \]
K-means Clustering

- K means **monotonically** 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($X, \alpha$)
1: Let $C$ be the initial set of centers (usually $C \leftarrow \{ \bar{x} \}$).
2: $C \leftarrow \text{kmeans}(C, X)$.
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 $\alpha$).
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:

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

$$A_x^2(Z) = A^2(Z) \left(1 + \frac{4}{n} - \frac{25}{(n^2)} \right)$$

Test for 1D values

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

\[ u = c_1 - c_2 \]

\[ x'_i = \langle x_i, u \rangle / \| u \|^2 \]
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:
**Centroidal Voronoi Diagram**

\[ \text{CentroidalVoronoi}(C, \epsilon) \]

1. \( \triangleright \) Compute \( k \) points evenly distributed on a spatial domain \( \triangleright \)
2. \( \triangleright \epsilon: \) threshold criterion to decide whether to stop or not \( \triangleright \)
3. Initialize centroids \( C \)
4. while Total centroid displacements less than \( \epsilon \)
5. \( \quad \textbf{do} \) Compute Voronoi diagram of \( C \)
6. \( \quad \) Allocate each \( c_i \) to the center of mass of its Voronoi cell

---

(a) \hspace{1cm} (b) \hspace{1cm} Centroidal Voronoi diagram
Stippling with Centroidal Voronoi diagrams

Incorporate a density function

\[ C_i = \frac{\int_A x \rho(x) dA}{\int_A \rho(x) dA} \]

NPR= Non Photorealistic Rendering (NPAR conference)


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

\texttt{SMALLENCLOSINGBALL}(p_1, \ldots, p_n, \epsilon)

1. \hspace{1em} \triangleright Compute a \((1 + \epsilon)\)-approximation of the smallest enclosing ball \triangleright
2. \hspace{1em} \triangleright Return the circumcenter of a small enclosing ball \triangleright
3. \hspace{1em} c \leftarrow p_1
4. \hspace{1em} \textbf{for} \ i \leftarrow 1 \ \textbf{to} \ \lceil \frac{1}{\epsilon^2} \rceil
5. \hspace{2em} \triangleright \text{Furthest point is } f_i = p_j \triangleright
6. \hspace{2em} j = \arg\max_{i=1}^n ||c p_i||
7. \hspace{2em} c \leftarrow c + \frac{1}{i+1} c p_j
8. \hspace{1em} \text{return } c
Designing **predicates**/Geometric axioms

Orient2D

\[
\text{Orient2D}(p, q, r) = \text{sign det} \begin{vmatrix} 1 & 1 & 1 \\ p & q & r \end{vmatrix}
\]

Orient2D(p, q, r) = sign det \[
\begin{vmatrix}
 x_q - x_p & x_r - x_p \\
y_q - y_p & y_r - y_p
\end{vmatrix}
\]

Orient2D(p_1, ..., p_d, p) = sign det

\[
\begin{vmatrix}
p_1^T & 1 \\
p_2^T & 1 \\
\vdots & 1 \\
p_d^T & 1 \\
p^T & 1
\end{vmatrix}
\]

**Determinant**=Signed area of the triangle formed by the 3 points
Designing predicates/Geometric axioms

\[ \text{InSphere2D} \]

\[ \text{InSphere2D}(s_1, \ldots, s_{d+1}, p) = \text{sign} \det \begin{bmatrix} s_1^T & s_1 \cdot s_1 & 1 \\ s_2^T & s_2 \cdot s_2 & 1 \\ \vdots & \vdots & 1 \\ s_{d+1}^T & s_{d+1} \cdot s_{d+1} & 1 \\ p^T & p \cdot p & 1 \end{bmatrix} \]