Fundamentals of 3D

Lecture 5:
Texture synthesis
Clustering k-means
Voronoi diagrams

Frank Nielsen
12 Octobre 2011
nielsen@lix.polytechnique.fr
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");
    }
}

Nowadays,

Motion JPEG 2000
H.264 (MPEG-4)

Note : In Processing, images are 1D arrays
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

© 2011 Frank Nielsen
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 $I_t$

Scanline

L-shape window

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

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

TEXTURESYNTHESIS($I_s$, $I_t$)
1. $I_s$ is the input texture sample
2. Create a large texture $I_t$
3. Initialize a random color image $I_t$
4. Synthesize pixels following the horizontal scanline order
5. for $y \leftarrow 1$ to $h_t$
6. do for $x \leftarrow 1$ to $w_t$
7. 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

2s + 1 = 5

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

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

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

Neighborhood of size 5, 11, 15, 23
Volumetric illustration

Geometry synthesis

© 2011 Frank Nielsen
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{\textlangle Clusters points of } \mathcal{P} \texttt{ using kMeans \rangle}
2. \texttt{\textlangle } \epsilon \texttt{: threshold criterion to decide whether to stop or not } \texttt{\rangle}
3. \texttt{Initialize centroids } \mathcal{C}
4. \texttt{while Total centroid displacements is less than threshold } \epsilon
5. \texttt{\textlangle Allocate points to clusters (hard membership) \rangle}
6. \texttt{for } i \leftarrow 1 \texttt{ to } n
7. \texttt{\quad do } C(p_i) = \arg\min_{j=1}^{k} ||p_i - c_j||
8. \texttt{for } i \leftarrow 1 \texttt{ to } k
9. \texttt{\quad do } \texttt{\textlangle Update centroids to the center of mass of clusters } \texttt{\rangle}
10. \texttt{\quad } \mathcal{C}(c_i) = \{p \in \mathcal{P} \mid C(p) = i\}
11. \texttt{\quad } 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

\[ \mathcal{P} = \{ \mathbf{p}_1, \ldots, \mathbf{p}_n \} \]

points

\[ \mathcal{C} = \{ \mathbf{c}_1, \ldots, \mathbf{c}_k \} \]

clusters

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

Hard/soft clustering

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

Lloyd k-means celebrated clustering algorithm:

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

© 2011 Frank Nielsen
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: $C \leftarrow kmeans(C, X)$.
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:

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

\[
A_x^2(Z) = A^2(Z)(1 + 4/n - 25/(n^2))
\]

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.

\[ v = c_1 - c_2 \]

\[ x'_i = \frac{\langle x_i, v \rangle}{||v||^2} \]
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:
Centroidal Voronoi diagram

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

\text{SMALL\textsc{EnclosingBall}}(p_1, \ldots, p_n, \epsilon)

1. \text{Compute a (1 + \epsilon)-approximation of the smallest enclosing ball} \triangleright
2. \text{Return the circumcenter of a small enclosing ball} \triangleright
3. \text{c} \leftarrow p_1
4. \text{for } i \leftarrow 1 \text{ to } \left\lceil \frac{1}{\epsilon^2} \right\rceil
5. \quad \text{do} \quad \text{Furthest point is } f_i = p_j \triangleright
6. \quad \quad j = \underset{i=1}{\overset{n}{\text{argmax}}} \|cp_i\|
7. \quad \quad c \leftarrow c + \frac{1}{i+1}cp_j
8. \text{return } c
Designing **predicates**/Geometric axioms

Orient2D

\[
\begin{align*}
\text{CCW} & : & \text{Orient2D}(p, q, r) = \text{sign } \det \begin{bmatrix} 1 & 1 & 1 \\ p & q & r \end{bmatrix} \\
\text{ON} & : & \text{Orient2D}(p, q, r) = \text{sign } \det \begin{bmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{bmatrix} \\
\text{CW} & : & \text{Orient2D}(p_1, \ldots, p_d, p) = \text{sign } \det \begin{bmatrix} p_1^T & 1 \\ p_2^T & 1 \\ \vdots & 1 \\ p_d^T & 1 \\ p^T & 1 \end{bmatrix}
\end{align*}
\]

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

© 2011 Frank Nielsen
Designing predicates/Geometric axioms

\[
\text{InSphere2D}
\]

\[
\begin{align*}
\text{IN} & : s_1 \quad s_2 \quad s_3 \\
\text{ON} & : s_1 \quad s_2 \quad s_3 \\
\text{OUT} & : s_1 \quad s_2 \quad s_3
\end{align*}
\]

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