Fundamentals of 3D

Lecture 1 Follow-up: Connected Components

Lecture 2: Convolutions and Filters
Matrix decompositions

Frank Nielsen
21 Septembre 2011
Labelling connected components (Union-Find)

Input: binary image Foreground/Background pixels
Output: Each connected component
Useful in computer vision...
→ Image segmentation

How many (large) objects?
A simple algorithm

Initially each pixel defines its own region
(labelled by say the pixel index: \(x+y\times\text{width}\))

Scan the image from left to right and top to bottom:

- If current pixel is background AND East pixel is background:
  Merge their regions into one = « connect » them

- If current pixel is background AND South pixel is background:
  Merge their regions into one = « connect » them

Extract regions (floodfilling or single pass algorithm)

How do we merge quickly **disjoint sets**?
Union-Find abstract data-structures

**MakeSet**($x$)
1. $\text{parent}(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

\[ S_1 = \{a, b, c, d\} \quad S_2 = \{e, f, g\} \]

\[ S = S_1 \cup S_2 \]

**RANK=DEPTH**

**Visual Depiction**
class UnionFind {
  int [] rank; int [] parent;

  UnionFind(int n) {
    int k;
    parent=new int[n];
    rank=new int[n];
    for (k = 0; k < n; k++)
      {parent[k]   = k; rank[k] = 0;     }
  }

  int Find(int k) {
    while (parent[k]!=k ) k=parent[k]; return k;}

  int Union(int x, int y) {
    x=Find(x);y=Find(y);
    if ( x == y ) return -1; // Do not perform union of same set
    if (rank[x] > rank[y])
      {parent[y]=x;
       return x;}
    else
      { parent[x]=y;
        if (rank[x]==rank[y]) rank[y]++;return y;}
  }
}

S_1 = \{a, b, c, d\} \quad S_2 = \{e, f, g\}
S = S_1 \cup S_2

// Attach tree(y) to tree(x)
// else
// Attach tree(x) to tree(y)
and if same depth, then increase depth of y by one
Many applications of the union-find data-structure

- Variable name aliases.
- Pixels in a digital photo.
- Computers in a network.
- Web pages on the Internet.
- Transistors in a computer chip.
Yet another example of UF/segmentation: Inbetweening for cell animation

http://www.vuse.vanderbilt.edu/~bobbyb/pubs/sca06.html

© 2011 Frank Nielsen
Minimum Spanning Tree (Kruskal's algorithm)

Implemented using Union-Find data structure

```
function Kruskal(G)
    for each vertex v in G do
        Define an elementary cluster C(v) = {v}.
    Initialize a priority queue Q to contain all edges in G, using the weights as keys.
    Define a tree T = ∅  // T will ultimately contain the edges of the MST
    // n is total number of vertices
    while T has fewer than n-1 edges do
        // edge u,v is the minimum weighted route from/to v
        (u,v) = Q.removeMin()
        // prevent cycles in T. add u,v only if T does not already contain a path between u and v.
        // Note that the cluster contains more than one vertex only if an edge containing a pair of
        // the vertices has been added to the tree.
        Let C(v) be the cluster containing v, and let C(u) be the cluster containing u.
        if C(v) ≠ C(u) then
            Add edge (v,u) to T.
            Merge C(v) and C(u) into one cluster, that is, union C(v) and C(u).
    return tree T
```
Convolutions et filtres

Color image

Grey image

Intensity=0.3red+0.59green+0.11blue
Image Convolution

Roberts Cross (edge) detection

Discrete gradient (maximal response for edge at 45 degrees)

\[
\begin{array}{cc}
+1 & 0 \\
0 & -1 \\
\end{array}
\quad
\begin{array}{cc}
0 & +1 \\
-1 & 0 \\
\end{array}
\]

\( G_x \)
\( G_y \)

The two filters are 90 degrees apart (inner product is zero)

\[
|G| = \sqrt{G_x^2 + G_y^2}
\]

\[
\theta = \arctan\left(\frac{G_y}{G_x}\right) - \frac{3\pi}{4}
\]

Approximated by

\[
|G| = |P_1 - P_4| + |P_2 - P_3|
\]
Roberts Cross edge detector

Approximated by

\[ |G| = |P_1 - P_4| + |P_2 - P_3| \]
Sobel edge detector

Approximated by

\[ |G| = |(P_1 + 2 \times P_2 + P_3) - (P_7 + 2 \times P_8 + P_9)| + |(P_3 + 2 \times P_6 + P_9) - (P_1 + 2 \times P_4 + P_7)| \]
Gaussian smoothing

\[ G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \]

Normalize discrete Gaussian kernel to 1

Two parameters to tune:

- K, the dimension of the matrix
- Sigma, the smoothing width...
Blurring, low-pass filter eliminates edges...

Source
Sigma=1, 5x5
Sigma=2, 9x9
Sigma=4, 15x15
Mean filter = Uniform average

Source

Corrupted, Gaussian noise (sigma=8)

Mean filter 3x3
Mean filter = Uniform average

Source       Corrupted, Gaussian noise (sigma=13)       Mean filter 3x3
Median filter

Neighbourhood values:

115, 119, 120, 123, 124, 125, 126, 127, 150

Median value: 124

<table>
<thead>
<tr>
<th>123</th>
<th>125</th>
<th>126</th>
<th>130</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>124</td>
<td>126</td>
<td>127</td>
<td>135</td>
</tr>
<tr>
<td>118</td>
<td>120</td>
<td>150</td>
<td>125</td>
<td>134</td>
</tr>
<tr>
<td>119</td>
<td>115</td>
<td>119</td>
<td>123</td>
<td>133</td>
</tr>
<tr>
<td>111</td>
<td>116</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
</tbody>
</table>

Source  Corrupted, Gaussian noise (sigma=8)  Median filter 3x3
Sharpening: Identity+Laplacian kernel

Source image

\[
S = \begin{bmatrix}
0 & -\lambda & 0 \\
-\lambda & 1 + 4\lambda & -\lambda \\
0 & -\lambda & 0
\end{bmatrix}
\]
Bilateral filtering

Traditional spatial gaussian filtering

\[ J(x) = \sum_{\xi} f(x, \xi) I(\xi) \]
Bilateral filtering

New! gaussian on the intensity difference filtering

\[ J(x) = \frac{1}{k(x)} \sum_{\xi} f(x, \xi) \quad \text{g}(I(\xi) - I(x)) \quad I(\xi) \]

Bilateral Filtering for Gray and Color Images, Tomasi and Manduchi 1998

SUSAN feature extractor...
Bilateral filtering

Traditional spatial gaussian filtering
The kernel shape depends on the image content.

\[ BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\| p - q \|) \cdot G_{\sigma_r}(\| I_p - I_q \|) \cdot I_q \]
Bilateral filtering

- Example of Bilateral filtering
- Low contrast texture has been removed

- Brute-force implementation is slow > 10min (but real-time with GPU and better algorithms)

© 2011 Frank Nielsen
Bilateral filtering

Origin image

Bilateral (3)
Results

Origin Image

One iteration

Five iterations

Bootstrapping

http://people.csail.mit.edu/sparis/siggraph07_course/

© 2011 Frank Nielsen
Bilateral filtering extends to meshes

Source

Bilateral mesh denoising
Harris-Stephens edge detector: Feature points

Aim at finding good feature points

\[ M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \]

Gradient with respect to \( x \), times gradient with respect to \( y \)

Sum over image region – area we are checking for corner
Harris-Stephens edge detector

Measure the corner response as

\[ R = \det M - k (\text{trace } M)^2 \]

Algorithm:
- Find points with large corner response function \( R \) (\( R > \text{threshold} \))
- Take the points of local maxima of \( R \)

\( k \) – empirical constant, \( k = 0.04-0.06 \)

\( \det M = \lambda_1 \lambda_2 \)
\( \text{trace } M = \lambda_1 + \lambda_2 \)

Avoid computing eigenvalues themselves.
Harris-Stephens edge detector

Corner response R
Thresholding $R > c$:

Local maxima of $R$.
Superposing local maxima on source images
Invariant to orientation but
Depends on the scaling of the image

- $R$ depends only on eigenvalues of $M$
- $R$ is large for a corner
- $R$ is negative with large magnitude for an edge
- $|R|$ is small for a flat region
Application to feature matching for image panorama tools

http://www.ptgui.com/download.html
Matrix operations using JAMA
**JAMA : A Java Matrix Package**

Use JAMA library

```
javac -classpath Jama-1.0.2.jar filename.java
```

**Summary of JAMA Capabilities**

<table>
<thead>
<tr>
<th>Object Manipulation</th>
<th>constructors set elements get elements copy clone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elementary Operations</td>
<td>addition subtraction multiplication scalar multiplication element-wise multiplication element-wise division unary minus transpose norm</td>
</tr>
<tr>
<td>Decompositions</td>
<td>Cholesky LU QR SVD symmetric eigenvalue nonsymmetric eigenvalue</td>
</tr>
<tr>
<td>Equation Solution</td>
<td>nonsingular systems least squares</td>
</tr>
<tr>
<td>Derived Quantities</td>
<td>condition number determinant rank inverse pseudoinverse</td>
</tr>
</tbody>
</table>

http://math.nist.gov/javanumerics/jama/

© 2011 Frank Nielsen
**Write a class wrapper around Jama**

Essential operations are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CholeskyDecomposition</td>
<td>Cholesky Decomposition.</td>
</tr>
<tr>
<td>EigenvalueDecomposition</td>
<td>Eigenvalues and eigenvectors of a real matrix.</td>
</tr>
<tr>
<td>LU Decomposition</td>
<td>LU Decomposition.</td>
</tr>
<tr>
<td>Matrix</td>
<td>Jama = Java Matrix class.</td>
</tr>
<tr>
<td>QRD Decomposition</td>
<td>QR Decomposition.</td>
</tr>
<tr>
<td>SingularValueDecomposition</td>
<td>Singular Value Decomposition.</td>
</tr>
</tbody>
</table>
LU Decomposition of rectangular matrices

Product of a lower and upper triangular matrices

\[ A = LU. \]

\[
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
\end{bmatrix}
= \begin{bmatrix}
    l_{11} & 0 & 0 \\
    l_{21} & l_{22} & 0 \\
    l_{31} & l_{32} & l_{33}
\end{bmatrix}
\begin{bmatrix}
    u_{11} & u_{12} & u_{13} \\
    0 & u_{22} & u_{23} \\
    0 & 0 & u_{33}
\end{bmatrix}.
\]

\[
\begin{pmatrix}
    2 & -1 & 0 \\
    -1 & 2 & -1 \\
    0 & -1 & 2
\end{pmatrix}
= \begin{pmatrix}
    1 & 0 & 0 \\
    -1/2 & 1 & 0 \\
    0 & -2/3 & 1
\end{pmatrix} \cdot \begin{pmatrix}
    2 & -1 & 0 \\
    0 & 3/2 & -1 \\
    0 & 0 & 4/3
\end{pmatrix}.
\]
LU Decomposition of rectangular matrices

\[ P^{-1} A = LU \]

\( P \) permutation matrix

LU Decomposition for solving linear systems:

\[ A \{x\} = \{b\} \]

\[ LU \{x\} = \{b\} \]

\[ L \{U.x\} = \{b\} \]

\[ \{U.x\} = \{y\} \]

\[ L\{y\} = \{b\} \]

Trivial to solve since \( L \) is lower triangular matrix
Matrix A=Matrix.random(3,3);
Matrix b = Matrix.random(3,1);
Matrix x= A.solve(b);

System.out.println("A="); A.print(6,3);
System.out.println("b="); b.print(6,3);
System.out.println("x="); x.print(6,3);

LU Decomposition luDecomp=A.lu();
Matrix L=luDecomp.getL();
Matrix U=luDecomp.getU();
int [ ] pivot= luDecomp.getPivot();

System.out.println("L="); L.print(6,3);
System.out.println("U="); U.print(6,3);
for(int i=0;i<pivot.length;i++)
    System.out.print(pivot[i]+" ");
System.out.println("\n");
QR Decomposition

\[ A = QR, \]

Q: Orthogonal matrix
R: Upper triangular matrix

Useful for solving least squares problem

```java
import Jama.*;

class QRDecompositionTest {
    public static void main(String args[])
    {
        Matrix A=Matrix.random(3,3);
        QRDecomposition qr=new QRDecomposition(A);
        System.out.println("A=");A.print(6,3);
        Matrix Q=qr.getQ();
        System.out.println("Q=");Q.print(6,3);
        Matrix R=qr.getR();
        System.out.println("R=");R.print(6,3);
    }
}
```

A=  
0.821 0.098 0.374
0.057 0.151 0.036
0.994 0.150 0.703

Q=  
-0.637 0.131 0.760
-0.044 -0.990 0.134
-0.770 -0.052 -0.636

R=  
-1.290 -0.185 -0.781
0.000 -0.145 -0.023
0.000 0.000 -0.158
Cholesky decomposition

\[ A = LL^*, \]

For a **symmetric positive definite matrix** \( A \), Cholesky decomposition yields:

- \( L \) is a lower triangular matrix
- \( L^* \) is the conjugate transpose
```java
import Jama.*;

class CholeskyDecompositionTest {
    public static void main(String args[]) {
        double [][] array = new double[3][3];
        int i, j;

        for (i = 0; i < 3; i++)
            for (j = 0; j <= i; j++) {
                array[i][j] = Math.random();
                array[j][i] = array[i][j];
            }

        Matrix A = new Matrix(array);
        CholeskyDecomposition cd = new CholeskyDecomposition(A);

        System.out.println("A="); A.print(6, 3);
        Matrix L = cd.getL();
        System.out.println("L="); L.print(6, 3);
        if (cd.isSPD())
            {L.times(L.transpose()).print(6, 3);}
    }
}
```

Matrix `A`:
```
0.688  0.342  0.046
0.342  0.411  0.345
0.046  0.345  0.962
```

Matrix `L`:
```
0.829  0.000  0.000
0.412  0.491  0.000
0.055  0.658  0.725
```

If `cd` is SPD, then:
```
0.688  0.342  0.046
0.342  0.411  0.345
0.046  0.345  0.962
```
Application of Cholesky decomposition: A multivariate normal

Box-Muller transform for standard normal variate: $\sqrt{-2 \ln U_1 \cos(2\pi U_2)}$

$U_1, U_2$ are independent uniform distributions (Math.random())

Multivariate normal density: $f_X(x) = \frac{1}{(2\pi)^{k/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)'\Sigma^{-1}(x - \mu)\right)$,

We obtain a random sample $z$ from $N(\mu, \Sigma)$ from its Cholesky decomposition

$\Sigma = A A^T$

by taking a vector of normal variates

$x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$

with $x_i = \sqrt{-2 \ln u_1 \cos(2\pi u_2)}$ where $u_1$ and $u_2$ are two uniformly drawn numbers (the Box-Müller transform), and scale and shift this vector as follows:

$z = \mu + Ax$
import Jama.*;

class Gaussian
{
  int dim; // dimension
  Matrix mu; // mean
  Matrix Sigma; // variance-covariance matrix
  Matrix A; // Auxiliary for Cholesky decomposition
  Matrix Precision; // Auxiliary for inverse of the covariance matrix

  // Constructor
  Gaussian(Matrix m,Matrix S){
    this.Sigma=S;
    ...
    CholeskyDecomposition cd=new CholeskyDecomposition(Sigma);
    this.A=cd.getL(); // Sigma=L L'
  }

  // Draw a normal variate
  Matrix draw(){
    Matrix x=new Matrix(dim,1); // column vector
    for(int i=0;i<dim;i++)
    {
      double u1=Math.random();
      double u2=Math.random();
      // Box-Muller transform
      double r=Math.sqrt(-2.0*Math.log(u1))*Math.sin(2.0*Math.PI*u2);
      x.set(i,0,r);
    }
    return mu.plus(A.times(x));
  }
}