Thin-Plate Splines

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: March 1, 1996
Last Modified: March 2, 2008

Contents

1 Discussion

Recall that natural cubic splines are piecewise cubic polynomial and exact interpolating functions for tabulated data $(x_i, f(x_i))$. The globally constructed spline has continuous second-order derivatives. The second derivatives at the end points are zero (no bending at end points). It is also possible to clamp the endpoints by specifying zero first derivatives there. The spline curve represents a thin metal rod that is constrained not to move at the grid points.

Thin-plate splines is a physically based 2D interpolation scheme for arbitrarily spaced tabulated data $(x_i, y_i, f(x_i, y_i))$. These splines are the generalization of the natural cubic splines in 1D. The spline surface represents a thin metal sheet that is constrained not to move at the grid points.

The idea is to build a function $f(x, y)$ whose graph passes through the tabulated data and minimizes the bending energy function

$$\int\int_{\mathbb{R}^2} (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2)\,dxdy.$$

For tabulated points $\{(x_i, y_i, z_i)\}_{i=1}^n$, the minimizing function is of the form

$$f(x,y) = \sum_{j=1}^{n} a_j E(\|(x - x_j, y - y_j)\|) + b_0 + b_1 x + b_2 y$$

where $E(r) = r^2 \log(r^2)$ and $\|\cdot\|$ indicates length of a vector. The coefficients a_j and b_j are determined by requiring exact interpolation. This leads to

$$z_i = \sum_{j=1}^{n} E_{ij} a_j + b_0 + b_1 x_i + b_2 y_i$$

for $1 \le i \le n$ where $E_{ij} = E(\|(x_i - x_j, y_i - y_j\|)$. In matrix form this is

$$\mathbf{z} = A\mathbf{a} + B\mathbf{b}$$

where $A = [E_{ij}]$ is an $n \times n$ matrix and where B is an $n \times 3$ matrix whose rows are $[1\ x_i\ y_i]$. An additional implication is that

$$B^{\mathrm{T}}\mathbf{a} = \mathbf{0}.$$

These two vector equations can be solved to obtain

$$\mathbf{a} = A^{-1}(\mathbf{z} - B\mathbf{b}) \text{ and } \mathbf{b} = (B^{\mathrm{T}}A^{-1}B)^{-1}B^{\mathrm{T}}A^{-1}\mathbf{z}.$$

It is also possible to provide a smoothing term. In this case the interpolation is not exact. The modification is to use the equation

$$\mathbf{z} = (A + \lambda I)\mathbf{a} + B\mathbf{b}$$

where $\lambda > 0$ is a smoothing parameter and I is the $n \times n$ identity matrix.

The thin-plate splines code in `Wm4IntpThinPlateSpline2.*` is an implementation which allows the smoothing parameter. The default value is $\lambda = 0$. Sample code is given below. Although the tabulated data in the example is on a regular grid, arbitrary spacing is allowed. Try to avoid data sets for which the (x, y) locations are nearly collinear.

```
#include "Wm4IntpThinPlateSpline2.h"
using namespace Wm4;

#include <iostream>
using namespace std;

int main ()
{
    cout.setf(ios::fixed);

    // tabulated data on a 3x3 regular grid, points of form (x,y,f(x,y))
    const int iQuantity = 9;
    float afX[iQuantity] =
    {
        0.0f, 0.5f, 1.0f,
        0.0f, 0.5f, 1.0f,
        0.0f, 0.5f, 1.0f
    };

    float afY[iQuantity] =
    {
        0.0f, 0.0f, 0.0f,
        0.5f, 0.5f, 0.5f,
        1.0f, 1.0f, 1.0f
    };

    float afF[iQuantity] =
    {
        1.0f, 2.0f, 3.0f,
        3.0f, 2.0f, 1.0f,
        1.0f, 2.0f, 3.0f
    };

    // resample on a 7x7 regular grid
    const int iResample = 6;
    const float fInv = 1.0f/(float)iResample;
    int i, j;

    // no smoothing, exact interpolation at grid points
    IntpThinPlateSpline2 kSplineNoSmooth(iQuantity,afX,afY,afF);
    cout << "no smoothing (smooth parameter is 0.0)" << endl;
    for (j = 0; j <= iResample; j++)
    {
        for (i = 0; i <= iResample; i++)
            cout << kSplineNoSmooth(fInv*i,fInv*j) << ' ';
        cout << endl;
    }
    cout << endl;
    // small amount of smoothing
    float fSmooth = 0.1f;
    IntpThinPlateSpline2 kSplineSmooth(iQuantity,afX,afY,afF,fSmooth);
    cout << "smoothing (smooth parameter is 0.1)" << endl;
    for (j = 0; j <= iResample; j++)
    {
        for (i = 0; i <= iResample; i++)
            cout << kSplineSmooth(fInv*i,fInv*j) << ' ';
        cout << endl;
    }
    cout << endl;

    return 0;
}
```

The output of the program is

```
no smoothing (smooth parameter is 0.0)
1.000000 1.365376 1.708851 2.000000 2.291150 2.634624 3.000000
1.747152 1.918864 1.992348 2.000000 2.007652 2.081136 2.252848
2.545341 2.518463 2.284589 2.000000 1.715411 1.481537 1.454660
3.000000 2.804738 2.413528 2.000000 1.586472 1.195262 1.000000
2.545341 2.518463 2.284589 2.000000 1.715412 1.481537 1.454660
1.747153 1.918865 1.992348 2.000000 2.007653 2.081137 2.252848
1.000000 1.365376 1.708851 2.000000 2.291150 2.634624 3.000000

smoothing (smooth parameter is 0.1)
1.141271 1.452767 1.747002 2.000000 2.252999 2.547233 2.858729
1.730097 1.888967 1.970424 2.000000 2.029576 2.111033 2.269904
2.359144 2.361507 2.200737 2.000000 1.799262 1.638493 1.640856
2.717458 2.587118 2.302354 2.000000 1.697646 1.412882 1.282543
2.359144 2.361507 2.200737 2.000000 1.799263 1.638493 1.640856
1.730097 1.888967 1.970424 2.000000 2.029576 2.111033 2.269903
1.141271 1.452767 1.747002 2.000000 2.252999 2.547233 2.858729
```

A similar implementation for 3D data sets with samples of the form $(x, y, z, f(x, y, z))$ is in the files `Wm4IntpThinPlateSpline3.*`.