Variational Subdivision for Laplacian Splines

Technical Report
TR97−291

Henrik Weimer
Joe Warren

Rice University
Department of Computer Science
Houston, TX 77025

August 1997

Abstract

The fundamental problem of geometric design is the representation of curved shapes. Traditionally such shapes are represented by parametric spline curves, e.g. NURBS, which are defined as the minimizers of variational problems. For example, cubic B−splines minimize the bending energy functional.

More recently subdivision curves and surfaces emerged as an alternative means for representing curved shapes. In this framework a curve or surface is defined as the limit of a repeated averaging process of control points. The subdivision scheme is determined by the subdivision mask S which specifies the weights used during the averaging.

A methodology for deriving subdivision schemes from homogeneous variational problems has been outlined in [Warren97]. This report shows the computational details of this approach and presents a method for deriving a subdivision scheme with predefined local support wich produces limit surfaces close to the minimizer of the variational problem. In particular this document contains the detailed derivation of Laplacian Splines discussed in section 4.2 of [Warren97].

The result will be a local stationary subdivision scheme for bounded rectangular grids which produces a limit surface that is close to the minimizer of the original variational problem. By increasing the support of the subdivision basis functions the resulting surface will come arbitrarily close to the real variationally optimal surface.

The *Mathematica* source code for this report can be found under URL http://www.cs.rice.edu/~jwarren/Laplace−tr.nb and an on−line HTML version under http://www.cs.rice.edu/~jwarren/Laplace−tr .

Introduction

Laplacian Splines are defined as the minimizer to the variational problem derived from Laplace’s equation

$$\min_f \int_\Omega \left(\left(\frac{\partial}{\partial_{t_1}} f(t) \right)^2 + \left(\frac{\partial}{\partial_{t_2}} f(t) \right)^2 \right) dt \qquad (1)$$

where the domain Ω is the real plane.The strong form of this equation is exactly Laplace's equation

$$\frac{\partial^2}{\partial_{t_1}{}^2} f(t) + \frac{\partial^2}{\partial_{t_2}{}^2} f(t) = 0. \qquad (2)$$

We want to refer to functions that satisfy this equation as Laplacian Splines.

This report presents a subdivision scheme which produces the minimizer of the variational probelm as stated in (1) for a bounded rectangular grid. This subdivision scheme has global support. We will also show how a local subdivision scheme can be constructed which produces a surface that is close to the minimizer of (1). By increasing the support of the local basis function of this approximating subdivision scheme, the resulting limit surface can lie arbitrarily close to the real Laplacian spline.

Definition of the Energy Matrix

The energy matrix for Laplacian splines is derived from Laplace's equation. We use a generating function representation of the energy mask e(z).

```
d1[i_, n_, x_] := Which[i == 0, 1 - x, i == n, -1/x + 1, True, -1/x + 2 - x]
d2[i_, n_, x_] := Which[i == 0, 1 - x, i == n, -1/x + 1, True, -1/x + 2 - x]
Fact[i_, n_] := If[i == 0 || i == n, 1/2, 1]
```

```
e[i_, j_, n_] :=
 Expand[(Fact[j, n] d1[i, n, x] + Fact[i, n] d2[j, n, y]) x^i y^j]
ue[i_, j_, n_] :=
 Expand[(Fact[j, n] d1[i, n, x^2] + Fact[i, n] d2[j, n, y^2]) x^i y^j]
```

```
GenRow[i_, j_, n_] := Flatten[Transpose[Drop[
    Transpose[CoefficientList[e[i, j, n] + (Sum[x^k, {k, 0, n}]) y^(n+1), {x, y}]], -1]]]
GenRow2[i_, j_, n_] := Flatten[Transpose[Drop[
    Transpose[CoefficientList[ue2[i, j, n] + (Sum[x^k, {k, 0, n}]) y^(n+1), {x, y}]], -1]]]
```

GenRow[i,j,n] now generates the coefficients of the energy matrix for control point (i,j) in an (n+1)x(n+1) grid. You may want to re−evaluate GenRow below for different values of i and j. GenRow2[i,j,n] returns the coefficients of the energy matrix for the subdivided grid (only for control points corresponding to control points on the previous level grid, i.e. before subdivision).

```
MatrixForm[Partition[GenRow[1, 1, 4], 5]]
```

$$\begin{pmatrix} 0 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

En[n] computes the energy matrix for a (n+1) x (n+1) grid, i.e. a (n+1)^2 x (n+1)^2 matrix computing the energy associated with each of the control points.

UEn[n] computes the energy matrix after upsampling, i.e. a matrix which computes the energy associated with the control points in a (n/2+1) x (n/2+1) grid and then upsamples them to the corresponding (n+1) x (n+1) grid by inserting zeroes as th energy associated with new control points.

```
En[n_] := Flatten[Table[GenRow[i, j, n], {i, 0, n}, {j, 0, n}], 1]
UEn[n_] := Flatten[Table[If[EvenQ[i] && EvenQ[j],
    GenRow[i/2, j/2, n/2], Table[0, {(n/2 + 1)^2}]], {i, 0, n}, {j, 0, n}],
  1]
```

Finding The Exact Subdivision Mask

En[n] as defined above is singular, i.e. not invertible. Thus we can not solve En S = UEn by inverting En. To make En invertible we replace one of the (linearly dependent) rows in En by a constraint representing the preservation of moments. Call these modified matricec NewUn and NewUEn respectivey.

```
Moments[n_] := Flatten[1/n^2 (Table[If[Mod[i, n] == 0 && Mod[j, n] == 0,
      1, If[Mod[i, n] == 0 || Mod[j, n] == 0, 2, 4]], {i, 0, n},
     {j, 0, n}])]
```

```
NewEn[n_] := ReplacePart[En[n], Moments[n], 1/2 ((n + 1)^2 + 1)]

NewUEn[n_] := ReplacePart[UEn[n], Moments[n/2], 1/2 ((n + 1)^2 + 1)]
```

Evaluate the following two lines to compute the exact subdivision matrix carrying control points from a 5x5 to a 9x9 control grid and from a 9x9 to a 17x17 grid. The resulting sequence of subdivision matrices will produce a limit surface that exactly solves Laplace's equation.

```
S5to9 = LinearSolve[N[NewEn[8]], NewUEn[8]];
```

```
S9to17 = LinearSolve[N[NewEn[16]], NewUEn[16]];
```

Unfortunately the resulting subdivision rules have global support (the exact solution must have global support as the variational problem is global). Here are some of the resulting exact subdivision rules rounded to three digits. Note the fast decay of the coefficients.

- **An internal vertex rule:**

$$\text{MatrixForm}\left[\text{Partition}\left[\frac{\text{Round}[\text{S9to17}[\![145]\!] * 100.0]}{100.0}, 9\right]\right]$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.02 & -0.13 & 0.02 & 0 & 0 & 0 \\ 0 & 0 & -0.01 & -0.13 & 1.45 & -0.13 & -0.01 & 0 & 0 \\ 0 & 0 & 0 & 0.02 & -0.13 & 0.02 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- **An internal horizontal edge rule:**

$$\text{MatrixForm}\left[\text{Partition}\left[\frac{\text{Round}[\text{S9to17}[\![146]\!] * 100.0]}{100.0}, 9\right]\right]$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0.03 & 0.03 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & -0.03 & 0.45 & 0.45 & -0.03 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0.03 & 0.03 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

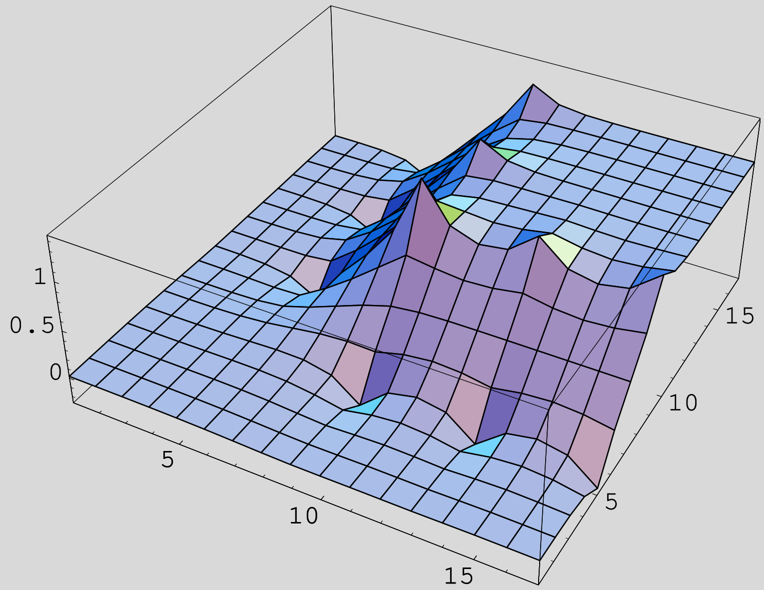
- **An internal face rule:**

```
MatrixForm[Partition[Round[S9to17[[163]] 100.0] / 100.0, 9]]
```

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	0	0	0	0	0	0
0	0	0	0	0.24	0.24	0	0	0
0	0	0	0	0.24	0.24	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

Apply the subdivision matrices to some control points

```
ListPlot3D[Partition[S9to17 . S5to9 . {0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1}, 17],
 PlotRange → All]
```




```
- SurfaceGraphics -
```

Note that the limit surface will be the exact solution to Laplace’s equation.

Finding a Sparse Approximation to the Subdivision Mask

The exact solution to the equation En S = U En as computed above leads to subdivision rules with global (but exponentially decaying) support.

In the sequel we want to find a subdivision scheme with local support that approximates the solution well. Our approach is to set up an unknown subdivision matrix with variable entries at locations that enforce the desired symmetries and sparsity. Call this subdivision matrix Shat. Now we can solve for the actual coefficients in Shat by minimizing the expression
En Shat = U En in the infinity norm. Recall that the infinity norm of a matrix is the maximum of the sums of the absolute values of entries in any row.

■ Set up a sparse subdivision matrix with the desired symmetry and sparsity

- **Some helper functions**
- **Definition of the Sparse Matrix**

```
Shat[n_] :=
 Flatten[Table[oneFlatRow[j, i, n], {j, 0, 2 n - 2}, {i, 0, 2 n - 2}], 1]
```

Now Shat[n] computes a sparse (variable) subdivision matrix carrying the control points from a bounded nxn grid to a (2n−1)x(2n−1) grid.

For example Shat[3] subdivides a 3x3 grid into a 5x5 grid. Note that we enforced sparsity and symmetry in the structure of this matrix.

```
MatrixForm[Shat[3]]
```

■ Set up a Linear Programming Problem to Solve for the Actual Coefficients in the Subdivision Matrix

Starting from the equation E S = U E we set up a linear programming problem to solve for S. The linear program will find an S such that the norm ‖ E S − U E ‖ is minimal, where ‖.‖ denotes the L_infinity norm, i.e. the sum of the absolut values of the elements in any one row should be minimal.

One crucial observation is that the symmetry and sparsity structure of S as defined above laeds only to a finite and fixed number of different rows in E S − U E, i.e. it is sufficient to minimize this small and fixed number of rows once to find a minimal solution for any size of S.

We want to solve here separately for the purely interior subdivisin rules as solving the whole system is beyond the abilities of Mathematica. We solved for the complete system using the CPLEX linear programming package.

- **Some helper functions to set up the linear program**
- **Get the distinct rows in E S − U E**

```
DSmUD[n_] := En[2 n - 2] . Shat[n] - UEn[2 n - 2]
```

Evaluate the following line to find all the different rows in E S − U D.

```
uniqueRows = UniqueRows[DSmUD[5]];
```

Note that using a larger system leads to the same distinct rows in E S − U D:

```
moreUniqueRows = UniqueRows[DSmUD[7]];
```

```
Dimensions[uniqueRows]
Dimensions[moreUniqueRows]
```

```
{15}
```

```
{15}
```

- **Partition these rows of E S − U E into three categories:**
  purely interior
  interior and boundary
  interior, boundary and corner

```
cornerVars = {S1[0, 0], S1[0, 1], S1[1, 1], S2[0, 0],
   S2[1, 0], S2[0, 1], S2[1, 1], S3[0, 0], S3[0, 1], S3[1, 1]};
boundaryVars = {S4[0, 0], S4[0, 1], S4[1, 0], S4[1, 1], S5[0, 0],
   S5[0, 1], S5[1, 0], S5[1, 1], S7[0, 0], S7[0, 1], S8[0, 0], S8[0, 1]};
interiorVars =
  {S6[0, 0], S6[0, 1], S6[1, 1], S9[0, 0], S9[0, 1], S9[0, 2], S10[0, 0]};
```

```
interiorRows = grepRules[uniqueRows, interiorVars];
boundaryRows = grepRules[uniqueRows, interiorVars ⋃ boundaryVars];
cornerRows =
  grepRules[uniqueRows, interiorVars ⋃ boundaryVars ⋃ cornerVars];
```

```
Dimensions[interiorRows]
Dimensions[boundaryRows]
Dimensions[cornerRows]
```

```
{3}
```

```
{10}
```

```
{15}
```

■ **Set up the systems of constraints**

```
Off[General::"spell1"]
interiorConstraints = AllMatrixConstraints[interiorRows, objective];
boundaryConstraints = AllMatrixConstraints[boundaryRows, objective];
cornerConstraints = AllMatrixConstraints[cornerRows, objective];
On[General::"spell1"]
```

■ **Set up constraints to enforce constant and linear precision**

```
One[n_] := Flatten[Table[1, {x, 1, n}, {y, 1, n}]]
```

```
tmpOne = One[5];
tmpOneSubdiv = One[9];
tmpS5 = Shat[5];
```

```
eigenConstraints[relevant_] := Flatten[
  Table[{tmpS5⟦relevant⟦i⟧⟧ . tmpOne == tmpOneSubdiv⟦relevant⟦i⟧⟧},
   {i, 1, Dimensions[relevant]⟦1⟧}]]
```

```
interiorEigenConstr = eigenConstraints[{21, 30, 31}];
boundaryEigenConstr = eigenConstraints[{19, 20, 21, 28, 29, 30, 31}];
cornerEigenConstr =
 eigenConstraints[{1, 10, 11, 19, 20, 21, 28, 29, 30, 31}];
```

■ **Finally solve for the coefficients for the interior**

Remember that all variables cann be positive or negative but the solver handles only positive variables. Thus all variables X in the system have to replaced by terms of the form (Xpositive – Xnegative).

```
Off[General::"spell1"]
posNegSubstitution =
 GetPosNegSubst[interiorVars ⋃ boundaryVars ⋃ cornerVars, "foo"];
On[General::"spell1"]
```

```
allInteriorConstraints =
 interiorConstraints ⋃ interiorEigenConstr /. posNegSubstitution;
allInteriorVars = ExtractVars[allInteriorConstraints];
```

```
interiorCoeffs = ConstrainedMin[
   objective, N[allInteriorConstraints], N[allInteriorVars]][[
  2]]
```

```
{fooN20 → 0, fooN21 → 0.111111, fooN22 → 0,
 fooN27 → 0, fooN28 → 0, fooN4 → 0, fooP20 → 0.0138889, fooP21 → 0,
 fooP22 → 1.38889, fooP27 → 0.0277778, fooP28 → 0.444444,
 fooP4 → 0.25, objective → 0.444444, SlackNeg1 → 0.0138889,
 SlackNeg2 → 0, SlackNeg3 → 3.05311 × 10^-16, SlackNeg4 → 0.0277778,
 SlackNeg5 → 0.0416667, SlackNeg6 → 0.222222, SlackNeg7 → 0,
 SlackNeg8 → 0, SlackNeg9 → 0, SlackPos1 → 0, SlackPos2 → 0.111111,
 SlackPos3 → 0, SlackPos4 → 0, SlackPos5 → 0, SlackPos6 → 0,
 SlackPos7 → 0.0555556, SlackPos8 → 0.0555556, SlackPos9 → 0}
```

- **Here are the resulting subdivision rules for the interior:**

```
InteriorS5 = N[tmpS5] /. N[posNegSubstitution] /. interiorCoeffs;
```

```
MatrixForm[Partition[InteriorS5[[31]], 5]]
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{MatrixForm}\left[\text{Partition}\left[\frac{\text{Round}[\text{InteriorS5}[[40]] * 100.0]}{100.0}, 5\right]\right]$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0.03 & 0.03 & 0 & 0 \\ 0 & 0.44 & 0.44 & 0 & 0 \\ 0 & 0.03 & 0.03 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{MatrixForm}\left[\text{Partition}\left[\frac{\text{Round}[\text{InteriorS5}[[41]] * 100.0]}{100.0}, 5\right]\right]$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & -0.11 & 0.01 & 0 \\ 0 & -0.11 & 1.39 & -0.11 & 0 \\ 0 & 0.01 & -0.11 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Sparse Solution for the Complete Grid

Mathematica's linear optimizer "ConstrainedMin" simply seems to run forever on the linear program arising from the complete set of constraints. We used an external solver, CPLEX, to solve for these coefficients instead.

Note that by construction these subdivision rules have constant precision (enforced in the constraints).

Here are the results:

```
cornerCoeffs = {S1[0, 0] → 1.430966, S1[0, 1] → -0.227169,
  S1[1, 1] → 0.02337218, S10[0, 0] → 0.25, S2[0, 0] → 0.4309657,
  S2[0, 1] → 0.09808379, S2[1, 0] → 0.4475783, S2[1, 1] → 0.02337218,
  S3[0, 0] → 0.2178049, S3[0, 1] → 0.2834315, S3[1, 1] → 0.2153321,
  S4[0, 0] → -0.1427126, S4[0, 1] → 0.05264114, S4[1, 0] → 1.450619,
  S4[1, 1] → -0.2704763, S5[0, 0] → 0.004644042000000001,
  S5[0, 1] → 0.05624038000000001, S5[1, 0] → 0.4709391,
  S5[1, 1] → 0.4072921, S6[0, 0] → 0.002322021, S6[0, 1] → -0.1111111,
  S6[1, 1] → 1.435156, S7[0, 0] → 0.4198616000000001,
  S7[0, 1] → 0.0801384, S8[0, 0] → 0.2308056, S8[0, 1] → 0.2691944,
  S9[0, 0] → 0.02777778, S9[0, 1] → 0.4444444};
```

```
Sparse5 = tmpS5 /. cornerCoeffs;
```

```
Sparse9 = Shat[9] /. cornerCoeffs;
```

```
MatrixForm[Partition[Sparse5⟦11⟧, 5]]
```

$$\begin{pmatrix} 0.2178049 & 0.2834315 & 0 & 0 & 0 \\ 0.2834315 & 0.2153321 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
MatrixForm[Partition[Sparse5⟦37⟧, 5]]
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -0.1427126 & 0.05264114 & 0 & 0 & 0 \\ 1.450619 & -0.2704763 & 0 & 0 & 0 \\ -0.1427126 & 0.05264114 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
MatrixForm[Partition[Sparse5〚40〛, 5]]
```

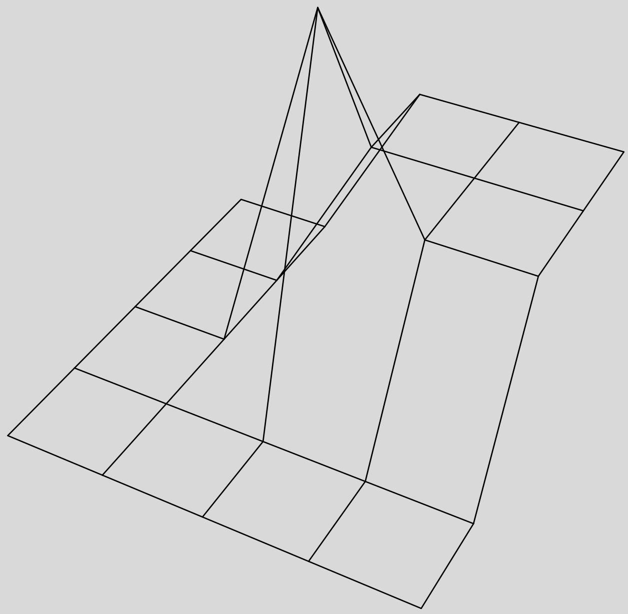
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0.02777778 & 0.02777778 & 0 & 0 \\ 0 & 0.4444444 & 0.4444444 & 0 & 0 \\ 0 & 0.02777778 & 0.02777778 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
MatrixForm[Partition[Sparse5〚41〛, 5]]
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0.002322021 & -0.1111111 & 0.002322021 & 0 \\ 0 & -0.1111111 & 1.435156 & -0.1111111 & 0 \\ 0 & 0.002322021 & -0.1111111 & 0.002322021 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

■ Now apply the sparse S to some control points and compare this visually to the exact solution

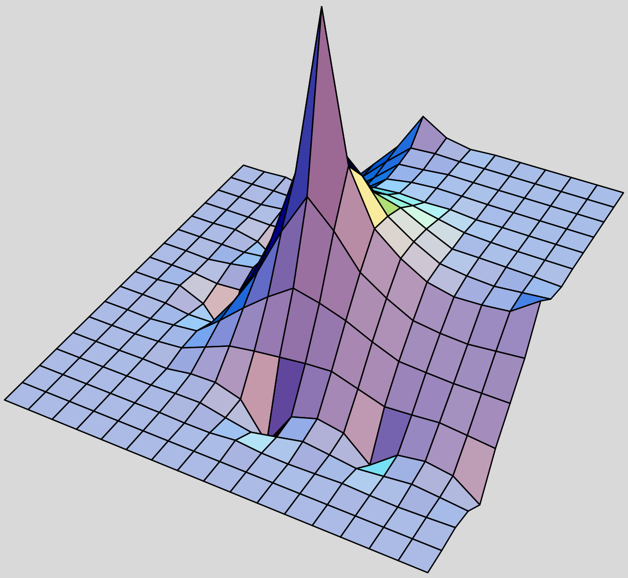
```
ListPlot3D[Partition[{0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1}, 5],
 PlotRange → All, Boxed → False, Axes → False, BoxRatios → {1, 1, 1},
 HiddenSurface → False]
```



```
- SurfaceGraphics -
```

The sparse approximation based on 5x5 basis functions:

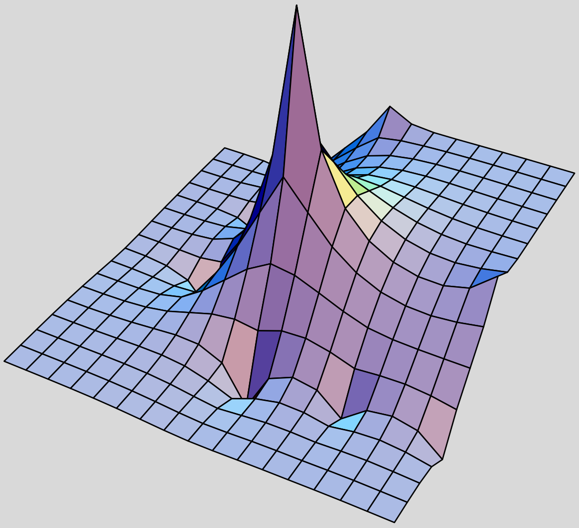
```
ListPlot3D[Partition[Sparse9 . Sparse5 . {0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1}, 17],
 PlotRange → All, Boxed → False, Axes → False,
 BoxRatios → {1, 1, 1}]
```



```
- SurfaceGraphics -
```

The exact subdivision process with globally supported basis functions:

```
ListPlot3D[Partition[S9to17 . S5to9 . {0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1}, 17],
  PlotRange → All, Boxed → False, Axes → False,
  BoxRatios → {1, 1, 1}]
```



```
- SurfaceGraphics -
```

References

[Warren97] Joe Warren: Subdivision Schemes for Variational Splines. July 1997, submitted for publication to CAGD.

Acknowledgements

This work has been supported in part by NSF grant CCR−9500572 and TATP grant 003604−010.