On Simulated Annealing and the Construction of Linear Spline Approximations for Scattered Data

Oliver Kreylos[1,2] and Bernd Hamann[1]

[1] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, CA 95616-8562, USA
[2] Institut für Betriebs- und Dialogsysteme, Fakultät für Informatik, Universität Karlsruhe (TH), 76128 Karlsruhe, Germany

Abstract. We describe a method to create optimal linear spline approximations to arbitrary functions of one or two variables, given as scattered data without known connectivity. We start with an initial approximation consisting of a fixed number of vertices and improve this approximation by choosing different vertices, governed by a simulated annealing algorithm. In the case of one variable, the approximation is defined by line segments; in the case of two variables, the vertices are connected to define a Delaunay triangulation of the selected subset of sites in the plane. In a second version of this algorithm, specifically designed for the bivariate case, we choose vertex sets and also change the triangulation to achieve both optimal vertex placement and optimal triangulation. We then create a hierarchy of linear spline approximations, each one being a superset of all lower-resolution ones.

1 Introduction

In several applications one is concerned with the representation of complex geometries or complex physical phenomena at multiple levels of resolution. In the context of computer graphics and scientific visualization, so-called *multiresolution methods* are crucial for the analysis of very large numerical data sets [1–5]. Examples include high-resolution terrain data (digital elevation maps) and high-resolution, three-dimensional imaging data (e. g., magnetic resonance imaging data).

We present an approach for the construction of multi resolution representations of very large scattered data sets using an iterative optimization algorithm and the principle of *simulated annealing* [9–12]. Our goal is the computation of several optimal linear spline approximations to a given scattered data set.

We assume that the given data sets are samples of a real function of one or two variables, with the samples randomly distributed in the function's domain and no known connectivity between them. Each individual linear spline approximation is defined by its control points and, in the case of multivariate functions, by the way these points are connected to form a triangulation. We only place control points at given sample positions and only use the supplied function values at those positions.

1.1 Visualizing Large Data Sets

To create a hierarchy of approximations to a given scattered data set we choose N_k vertices from the set at each hierarchy level k. We ensure that the set of vertices of any hierarchy level $j < k$ is a subset of level k's vertex set. After having decided which vertices to select for a hierarchy level k, that level's vertices are connected in an appropriate way to form a linear spline's control mesh. An example of such a hierarchy in the univariate case is shown in Fig. 1.

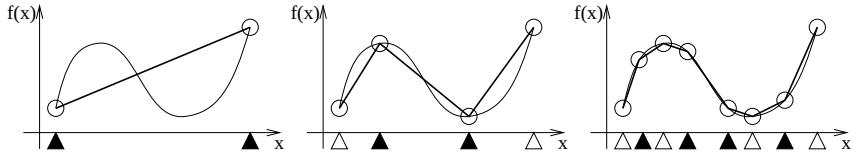

Fig. 1. A hierarchy of approximations in the univariate case. New vertices are inserted at the sites marked by solid triangles.

When representing high-resolution data sets with low-resolution linear spline approximations, one has to be careful where to place the spline's control points and how to connect them in order to achieve a faithful representation of the data set, see Fig. 2.

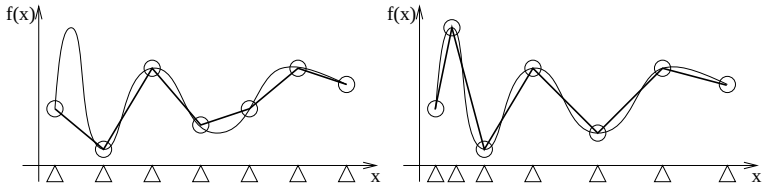

Fig. 2. Uniform vs. optimal control point placement for univariate data.

If the number of vertices for an approximation level is prescribed, one has to address two problems:

1. Which vertices should one choose for the approximation, i. e., how should one create the vertex placement?
2. How should one connect the chosen vertices, i. e., how should one create the connectivity?

In the special case of a function of one variable, we only have to address the first problem, since in the univariate case the connectivity is defined by the chosen sites' numerical order.

1.2 Finding Optimal Approximations

Our approach to finding an optimal linear spline approximation for a given, fixed number of vertices N_k is based on an iterative optimization algorithm. First, we create an initial configuration, then we improve this configuration by changing its vertex placement and its connectivity in every step. We judge a configuration's quality by its L^2 distance from the scattered data set. Since this optimization problem is high-dimensional and generally involves local minima in abundance, the algorithm of simulated annealing is well suited to construct "good" linear spline approximations [12, 9].

Simulated annealing is an iterative method that applies random changes to the current configuration and accepts a step depending on the resulting change of the error measure and a value called "temperature." This value determines the probability of accepting a step that increased the error measure: The higher the temperature, the higher the probability of accepting a bad step. The so-called "annealing schedule" determines how fast the temperature is decreased during the iteration.

In the case of two or more variables the quality of a configuration depends on both vertex placement and connectivity. There are two different ways to proceed:

1. One can ignore the optimization of the connectivity by enforcing a fixed type of connectivity throughout the iteration process; in the bivariate case, an obvious candidate is the Delaunay triangulation [6]. Under this constraint the algorithm can proceed exactly as in the univariate case.
2. One can attempt to optimize both parts of the configuration in parallel. For example, before each step one could randomly decide to either move a vertex or *swap* a common edge of two adjacent triangles.

2 The Optimization Algorithm

We now describe the individual steps of our algorithm. Algorithm 1 is a high-level description. The subsequent sections describe the important steps in more detail.

Algorithm 1: Optimal linear spline approximation.

```
Create initial configuration (vertex placement and connectivity);
Determine initial temperature and create annealing schedule;
While iteration is not finished {
  Change current configuration;
  Calculate change in error measure;
  Undo iteration if rejected by simulated annealing; }
Return current configuration;
```

2.1 Creating an Initial Configuration

Our approximations are defined over the original sites' convex hull. In the univariate case, we cover the convex hull by choosing the leftmost and the rightmost original vertices and distribute the rest of vertices uniformly between them. In the multivariate cases cover the convex hull by always selecting all non-interior vertices; then we choose the rest of vertices randomly from the original data set. In the bivariate case, we define the initial connectivity by a Delaunay triangulation of the initial vertices' sites.

2.2 Creating an Annealing Schedule

A reasonable heuristic to define the initial temperature is to apply some steps of the iteration scheme and to define the initial temperature in a way that the annealing algorithm initially accepts an "expected bad" step with a probability of one half. Next, we lower the temperature in steps, leaving it constant for a fixed number of iterations and scaling it by a fixed factor afterwards.

2.3 Changing the Current Configuration

The simulated annealing algorithm's core is its iteration step. In principle, one can use any method to change the current configuration, but we have found out that the "split" approach, shown in Algorithm 2, works very well.

Algorithm 2: Changing the current configuration.

```
if(acceptWithProbability(moveProbability)) { /* move a vertex */
  Choose an interior vertex v;
  Estimate v's contribution vE to the error measure;
  if(vE < localMovementFactor * E)
    Move v globally;
  else
    Move v locally;
  if(moveProbability == 1) /* Vertex movements only? */
    Restore Delaunay property; }
else { /* swap an edge */
  Choose a swappable edge e;
  Swap edge e; }
```

The constant *moveProbability* is used to control the behaviour of the optimization process for bivariate functions. If this constant's value is one, the algorithm moves a vertex in every step, and after each vertex movement the current triangulation is updated to satisfy the Delaunay property. In the other case the algorithm can either move a vertex or swap an edge, thereby optimizing both vertex placement and triangulation simultaneously.

Estimating a Vertex' Error Contribution. To estimate how much the removal of an interior vertex v would increase the current error measure, we estimate the "volume" of v's platelet: We construct an approximating least squares hyperplane H for all vertices surrounding v. Then we calculate h as v's ordinate-direction distance from H and A as the area of v's platelet, see Fig. 3. We define the error contribution as $\sqrt{A \cdot h^2/2}$ in the univariate case and as $\sqrt{A \cdot h^2/3}$ in the bivariate case, to ensure that the ratio of a vertex' error contribution and the used L^2 error measure is scale-invariant.

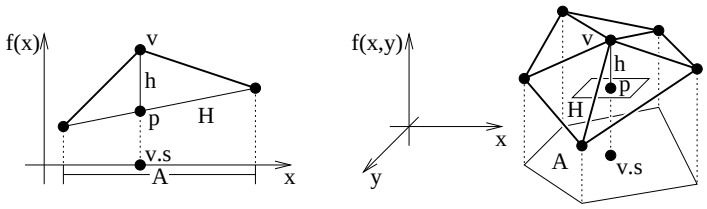

Fig. 3. Estimating a vertex' contribution to the error measure.

Global Vertex Movement. If v's error contribution is smaller than a constant *localMovementFactor* times the current error measure E, we assume that v is currently located in a "flat" region of the function and should be moved away from this region. We move v *globally* to a randomly chosen new site not already being part of the current configuration. By doing this we assure that vertices get driven away from nearly flat regions of a function in early stages of the iteration. Figure 4 shows the process of actually moving a vertex globally in detail.

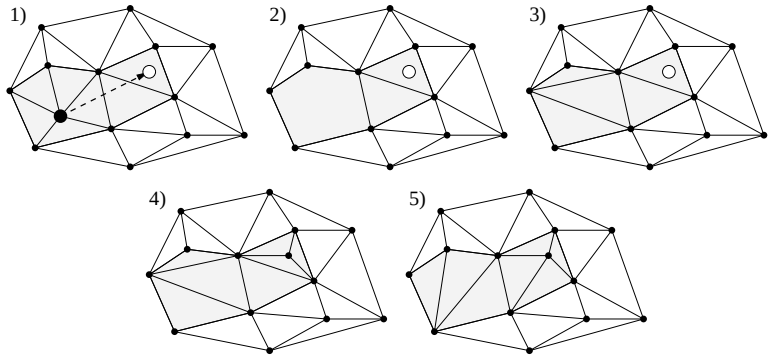

Fig. 4. Moving a vertex globally in the bivariate case. 1) initial state; 2) removing the vertex; 3) filling the hole; 4) inserting new vertex; 5) restoring the Delaunay property (only if the connectivity is ignored during optimization).

Local Vertex Movement. When a vertex' error contribution is larger than $localMovementFactor \cdot E$, we assume it is currently located in an "important," high-curvature region of the target function, and we attempt to find a better site for this vertex by moving it *locally* to a new, unoccupied site in its platelet. To move a vertex locally, we "slide" the vertex on the line from its old to its new site, dragging the edges connecting it to all surrounding vertices along. Whenever a surrounding simplex becomes degenerate during the vertex' motion, we swap one edge of the affected simplex before moving the vertex any further, see Fig. 5.

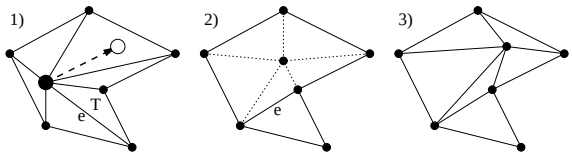

Fig. 5. Moving a vertex locally in the bivariate case: 1) initial state; 2) swapping edge e to prevent triangle T from becoming degenerate; 3) resulting state.

3 Examples and Results

In this section we show some of the experiments we did to evaluate our algorithm's behaviour. We begin with univariate and bivariate scalar-valued functions, and we then use a slight variation of the algorithm, where the Euclidian distance between vectors is used to calculate the L^2 error measure, to approximate bivariate vector-valued functions.

3.1 Univariate Scalar-valued Functions

We have tested our approach for these univariate scalar-valued functions:

1. The first test case is the function

$$f(x) = \begin{cases} 2(1-x) & \text{if } x < 1 \\ 4(1-x)(x-2) & \text{if } 1 \le x < 2 \\ 2(x-2) & \text{if } 2 \le x < 3 \\ 2(x-3)^2 & \text{if } 3 \le x \end{cases}, \quad x \in [0,4],$$

   and a linear spline approximation with 14 vertices, see Fig. 6. Our algorithm finds a very good approximation, although the function has discontinuities in both the zeroeth and first derivatives. In the two quadratic sections $[1,2]$ and $[3,4]$ the sites are uniformly distributed; we thus assume that the resulting approximation is globally optimal.

2. The second test case is the function

$$f(x) = 4 \sum_{n=0}^{3} \frac{\sin\big((2n+1)x\big)}{2n+1} \quad , \quad x \in [0, 4\pi],$$

the fourth-order Fourier approximation of a square wave, and a linear spline approximation with ten vertices, see Fig. 7. The number of vertices is too small to capture all details of the function, but the algorithm still finds a very good approximation.
3. The third test case is the same function as in the second, but this time using a linear spline approximation with 30 vertices, see Fig. 8. Now all the function's important features are present in the approximation.

3.2 Bivariate Scalar-valued Functions

We have tested our approach for these bivariate scalar-valued functions:

4. The fourth test case is the function

$$f(x, y) = 2 \sum_{i=0}^{2} \sum_{j=0}^{2} \frac{\sin\big((2i+1)x\big)}{2i+1} \cdot \frac{\sin\big((2j+1)y\big)}{2j+1} \quad , \quad x, y \in [0, 2\pi],$$

the third-order Fourier approximation of a bivariate square wave, and a linear spline approximation with 50 vertices and a general triangulation, see Fig. 9. The number of vertices is too small to capture all details of the function, but the algorithm still finds a decent approximation.
5. The fifth test case is the same function as in the fourth, but this time using a linear spline approximation with 250 vertices and a general triangulation, see Fig. 10. Due to the high number of vertices the iteration takes much longer to converge, but captures all details of the target function.
6. The sixth test case is a scattered data set consisting of 37,594 vertices, resulting from a laser scan of a Ski-Doo hood and a linear spline approximation with 1,000 vertices and a general triangulation, see Fig. 11. This case shows that our algorithm can be used in surface reconstruction, as long as the source data can be interpreted as a bivariate, scalar valued function.

3.3 Bivariate Vector-valued Functions

We have applied our method to these RGB color image data sets:

7. The seventh test case is a photograph of the Golden Gate Bridge in San Francisco, resampled to a resolution of 329×222 pixels, see Fig. 12, and a linear spline approximation with 400 vertices and a general triangulation, see Fig. 13. The RGB image is interpreted as a bivariate vector-valued function, defined by samples positioned at the pixels' centers.
8. The eigth test case is the same function as in the seventh, but this time approximated by a linear spline with 1,600 vertices and a general triangulation, see Fig. 14. The resulting linear spline is a superset of the result of experiment seven as defined in Sect. 1.1. It is hard to see in these low-quality reproductions, but the approximation is very close to the original image.

4 Conclusions and Future Work

In this paper we presented a method to calculate optimal linear spline approximations to functions defined by scattered data, using an iterative optimization technique governed by the simulated annealing algorithm. Our method is a generalization of the data-dependent triangulation method discussed by Schumaker [9]. We have demonstrated that our method performs well for univariate and bivariate scalar-valued functions. Furthermore, we have found that our algorithm approximates RGB images very well, even when using only a small number of vertices. Our technique provides an interesting alternative way to transform images to a storage-efficient, resolution-independent representation.

The main areas for future research are the generalization of our algorithm to functions of three and more variables and the application of our method to image and video compression. If one treats video data as time-varying bivariate vector-valued functions, and exploits the strong frame coherence of video streams especially in tele-conferencing, our algorithm might lead to a real-time video compression method for this kind of video streams.

5 Acknowledgements

This work was supported by grants and contracts awarded to the University of California, Davis, including the National Science Foundation under contract ACI 9624034 (CAREER Award), the Office of Naval Research under contract N00014–97–1–0222, the Army Research Office under contract ARO 36598–MA–RIP, the NASA Ames Research Center through an NRA award under contract NAG2–1216, the Lawrence Livermore National Laboratory through an ASCI ASAP Level-2 under contract W–7405–ENG–48 (and B335358, B347878), and the North Atlantic Treaty Organization (NATO) under contract CRG.971628. We also acknowledge the support of Silicon Graphics, Inc., and thank all members of the Visualization Thrust at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

1. Bonneau, G.-P., Hahmann, S. and Nielson, G. M., *BLaC-wavelets: A multiresolution analysis with non-nested spaces*, in Yagel, R. and Nielson, G. M., eds., Visualization '96 (1996), IEEE Computer Society Press, Los Alamitos, CA, pp. 43–48
2. Eck, M., DeRose, A. D., Duchamp, T., Hoppe, H., Lounsbery, M. and Stuetzle, W., *Multiresolution analysis of arbitrary meshes*, in Cook, R., ed., Proc. SIGGRAPH 1995, ACM Press, New York, NY, pp. 173–182
3. Gieng, T. S., Hamann, B., Joy, K. I., Schussman, G. L. and Trotts, I. J., *Constructing hierarchies for triangle meshes*, in IEEE Transactions on Visualization and Computer Graphics 4(2) (1998), pp. 145–161
4. Hamann, B., *A data reduction scheme for triangulated surfaces*, in Computer Aided Geometric Design 11(2) (1994), pp. 197–214

5. Hamann, B., Jordan, B. J. and Wiley, D. A., *On a construction of a hierarchy of best linear spline approximations using repeated bisection*, in IEEE Transactions on Visualization and Computer Graphics 4(4) (1998)
6. de Berg, M., van Kreveld, M., Overmars, M. and Schwarzkopf, O., *Computational Geometry* (1990), Springer-Verlag, New York, NY
7. Edelsbrunner, H., *Algorithms in Combinatorial Geometry* (1987), Springer-Verlag, New York, NY
8. Preparata, F. P., Shamos, M. I., *Computational Geometry*, third printing (1990), Springer-Verlag, New York, NY
9. Schumaker, L. L. *Computing Optimal Triangulations Using Simulated Annealing*, in Computer Aided Geometric Design 10 (1993), pp. 329–345
10. Nielson, G. M., *Scattered data modeling*, in IEEE Computer Graphics and Applications 13(1) (1993), pp. 60–70
11. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes in C*, 2nd ed. (1992), Cambridge University Press, Cambridge, MA
12. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., in Journal of Chemical Physics 21 (1953), pp. 1087–1092

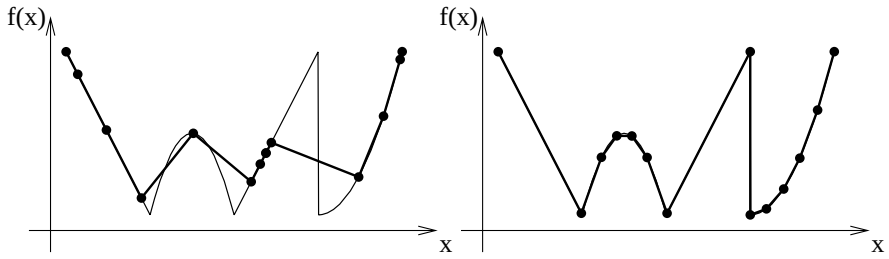

Fig. 6. First experiment. Left: initial vertex placement; right: final vertex placement.

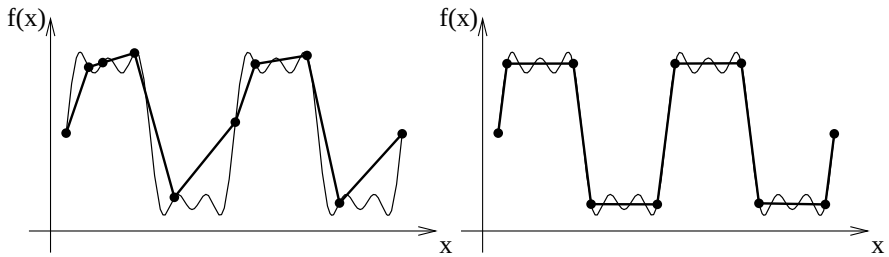

Fig. 7. Second experiment. Left: initial vertex placement; right: final vertex placement.

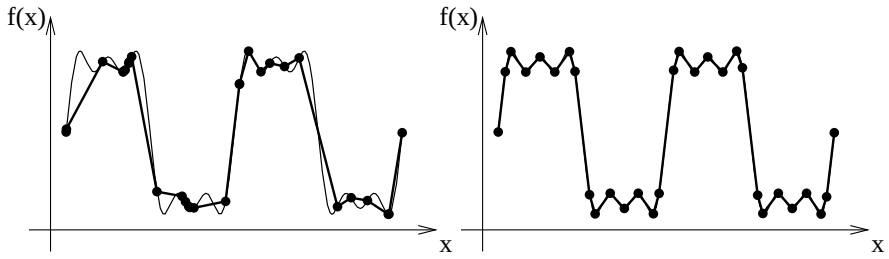

Fig. 8. Third experiment. Left: initial vertex placement; right: final vertex placement.

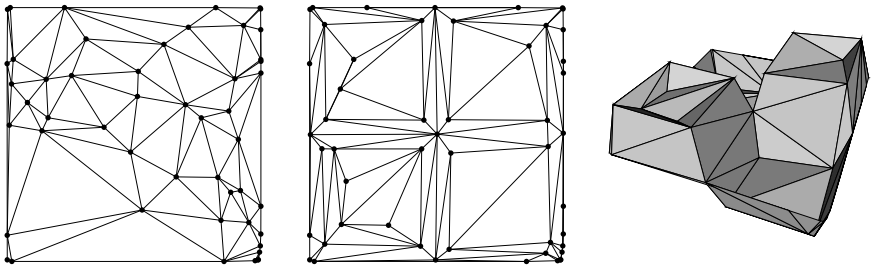

Fig. 9. Fourth experiment. Initial and final configurations and flat-shaded rendering.

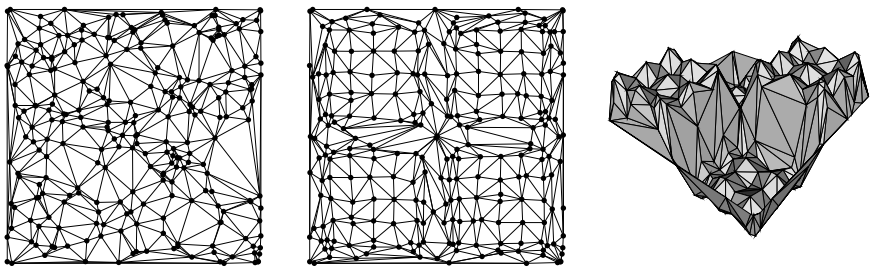

Fig. 10. Fifth experiment. Initial and final configurations and flat-shaded rendering.

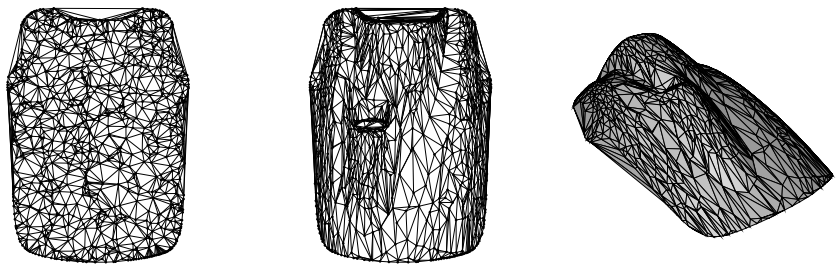

Fig. 11. Sixth experiment. Initial and final configurations and flat-shaded rendering.

Fig. 12. Seventh experiment. Original RGB image, 329×222 pixels.

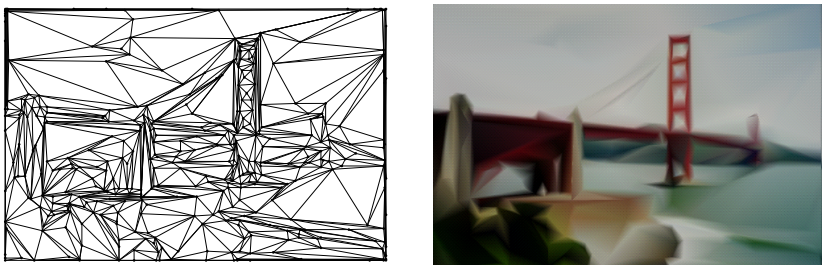

Fig. 13. Seventh experiment. Left: final configuration; Right: final linear spline approximation.

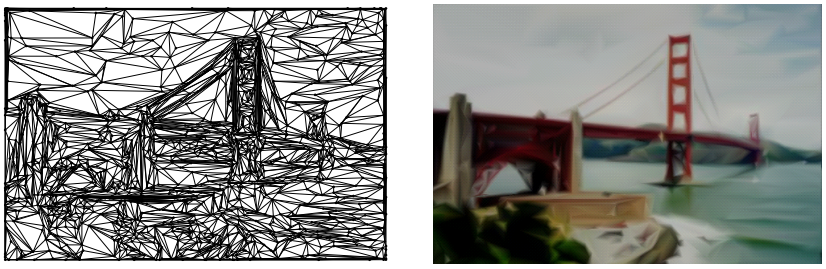

Fig. 14. Eigth experiment. Left: final configuration; Right: final linear spline approximation.