Piecewise algebraic surface computation and fairing from a discrete model

Jordi Esteve Alvar Vinacua Pere Brunet
Universitat Politècnica de Catalunya
Barcelona, Catalunya, Spain
e-mail: [jesteve, brunet, alvar]@lsi.upc.edu

September 26, 2005

Abstract

This paper describes a constrained fairing method for implicit surfaces defined on a voxelization. This method is suitable for computing a closed smooth surface that approximates an initial set of face connected voxels. The implicit surface is defined as the zero-set of a tensor-product uniform cubic Bspline. The fairing process is based on increasing the Bspline continuity from C^2 to C^3 on the boundary faces of the voxels. The final surface is guaranteed to stab a predefined subset of voxels.

Keywords: geometric modeling, algebraic surfaces, surface fairing, surface fitting.

1 Introduction

Discrete models like volumetric data sets are widely used in computer graphics to represent 3D objects or to store physically measured information. A surface rendering algorithm is normally used to obtain a pleasant visualization of the data. The surface extraction from the discrete model is a very important step in the rendering process. It must be robust, efficient and capable of producing good pleasant surfaces.

The main goal of this paper is the computation of a smooth surface from discrete models. The input of the algorithm is a closed set of face connected voxels that splits the voxels in an interior and an exterior. The output is a smoothed algebraic surface that approximates this set of voxels. Furthermore, the user may indicate a subset of these voxels as that which must be interpolated. The resulting surface will stab these voxels and approximate the rest while having the same topology as the complete set of face-connected voxels. Other types of surfaces like triangular meshes can be computed from the algebraic surface.

This algorithm has other useful applications like noise removal in data acquired from 3D scans (medical data for example), or repairing triangulations: triangulation algorithms used to construct surface models from scanned 3D points often generate anomalies like excessive genus (through generation of spurious very small handles in complicated regions of the surface), holes or non 2-manifoldness, that our algorithm is able to repai automatically.

Figure 1 shows a 2D version of the problem we are addressing. Colored voxels represent the inpunt collection of 6-connected voxels. As stated above, the user has chosen to mark some voxels as those that must be interpolated (in red) while others (in blue) need only be approximated.

Figure 1b shows the final planar curve obtained in the 2D case. It is a closed curve stabbing the marked voxels with the same topology as the initial set of voxels. Notice it misses some of the non-marked voxels, and it wanders outside the initial set of voxels. This flexibility yields a more pleasant curve or surface.

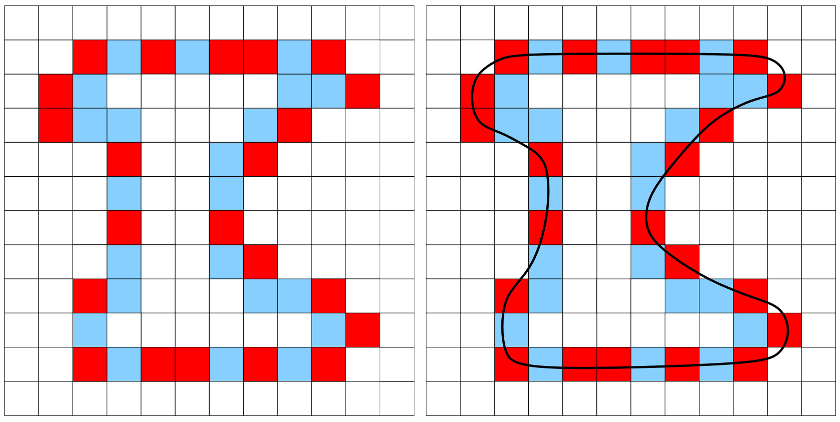

(a) Initial set of voxels (b) Final curve

Figure 1: Planar curve obtained from a 2D set of voxels

A 3D example of our algorithm is shown in figure 9a. Figure 9c shows the smooth surface computed from the initial surface 9b and fitted to stab the marked voxels (in red) of figure 9a. The surface of figure 9b lies inside the initial set of voxels and suffers aliasing. The final surface is fairer while fitting the marked voxels.

Depending on the application, the marked voxels can be critical areas defined by the user or known places of the model that the final surface must interpolate.

In the next section we present some previous work before discussing our proposal in detail. Section 3 presents the algorithm overview and ways in which the collection of face-connected voxels may be obtained from other sets of data. Sections 4, 5, 6 explain the three main steps of the algorithm in more detail: the fairing, the fitting and the updating steps respectively. In section 7 we discuss the necessary changes to deal with trilinear Bspline surfaces instead of tricubic Bsplines. Section 8 includes several examples and results. Finally, we point out some conclusions and possible improvements of our work.

2 Previous work

There is an important amount of literature about fairing algorithms in CAGD. Some of it deals with discrete models like triangular meshes. We will only focus on the fairing methods for continuous Bspline models.

Several fairing metrics have been proposed for these kind of surfaces. Minimum energy metrics [Hadenfeld(1995)] or minimum variation metrics [Moreton and Séquin(1994)], [Lott and Pullin(1988)] are the most used. Other proposed fairing criteria are based on surface areas [Rando and Roulier(1991)] or measuring the light reflection of a surface [Klass(1980)].

It is specially relevant to our work the criterion of Kjellander [Kjellander(1983b)], which states that if the difference in the third derivative on both sides of a Bspline knot is decreased, its energy (the shear forces applied to it) is also decreased. He uses this principle to smooth cubic and bicubic parametric splines in a global fairing method [Kjellander(1983a)]. Poliakoff [Poliakoff(1996)] extends this method to deal with non-uniform cubic parametric splines.

Farin et al. [Farin et al.(1987)Farin, Rein, Sapidis, and Worsey] present a local fairing method to smooth cubic Bspline curves based in the previous ideas. The knot with worst fairness criterion is removed and reinserted. This method is difficult to generalize to surfaces. It has been done in [Hahmann and Konz(1998)] with the help of different search strategies to find the worst knot without looking over the whole surface.

Hahmann [Hahmann(1998)] extends the Kjellander principle to smooth a Bspline surface by increasing the surface continuity from C^2 to C^3 at a knot, at a set of neighboring knots or at a segment between two neighboring patches. Our smoothing algorithm is based on the idea of increasing the continuity at a face between two neighboring voxels.

Piecewise algebraic surfaces defined within a rectangular domain were introduced in [Patrikalakis and Kriezis(with the Bernstein basis and non-uniform Bsplines. Due to the regularity of the voxelization, we have defined the piecewise algebraic surface as the zeroes of a uniform Bspline. See section 3.2 for a detailed definition. In a previous work [Esteve et al.(2001)Esteve, Brunet, and Vinacua] it has been shown how to compute multiresolution representations from this kind of surfaces.

Other types of piecewise algebraic surfaces have been proposed in the literature. In [Sederberg(1984)] planar piecewise curves defined in a triangular domain are described. This idea is extended to algebraic surfaces defined within tetrahedra in [Sederberg(1985)]. Because of the domain, this algebraic surfaces are not suitable for the resolution of our problem.

A conversion method from a voxel or octree representation to a uniform cubic Bspline algebraic surface is described in [Vinacua et al.(1998)Vinacua, Navazo, and Brunet]. The approach used there is different, its computational cost is much higher and its results less satisfactory.

3 Algorithm overview

Before entering into the detailed discussion of our algorithm, the concept of *discrete membrane* is defined and the surface model is presented in the next subsections.

3.1 Discrete membrane

Given a voxelized volume, we call a subset M of these voxels a *discrete membrane* if it separates the remainder of the voxels in two classes (interior and exterior) such that any continuous curve starting at an interior voxel and ending at an exterior voxel must contain points in the interior of M. This is analogous to the requirement that a surface in 3D space be closed, in that it separates the space in two separate path-connected components. Notice that a discrete membrane must be face-connected.

The voxels of the discrete membrane are sorted into two kinds:

- *Hard voxels*: Predefined voxels that must be stabbed by the final surface.
- *Soft voxels*: Voxels added in the vicinity of the hard voxels to get a closed set of face-connected voxels and defining the topology of the discrete membrane.

Discrete membranes can be easily obtained from an octree representation [Vinacua et al.(1998)Vinacua, Navaz or from a segmentation of a volumetric data set. They can also be computed from a closed surface [Sramek and Kaufman(1999), Andújar et al.(2002)Andújar, Brunet, and Ayala]. In [Esteve et al.(2005)Esteve, Brunet, and Vinacua] the concept of a discrete membrane is introduced and an algorithm for computing it from a variable density cloud of points is presented. In that paper, hard voxels are those that contain initial data points.

3.2 Bspline-based piecewise algebraic surfaces

Before presenting our algorithm, we need to define precisely how our implicit surfaces look like and set some notation.

We first discuss the case of a planar curve, which admits a simple graphical presentation. Let us consider a scalar function $F_C(x, y)$ defined on a plane. Its graph is the set $\mathcal{G} = (x, y, F_C(x, y))$ in the 3D space. Consider the intersection of its graph with the plane $z = 0$ (see figure 3). This algebraic curve is the 2D analogous of the surfaces we intend to use, when f is a uniform tensor-product Bspline.

To model a surface, we define the scalar function $F(x, y, z)$ on a 3D prism. Its zero values produce the expected piecewise algebraic surface.

As in [Vinacua et al.(1998)Vinacua, Navazo, and Brunet] and [Esteve et al.(2001)Esteve, Brunet, and Vinac it is only necessary to define the scalar function on the voxels stabbed by the isosurface. These will form a set of voxels named *Voxel Collection* (see figure 4). Let us now define G as the set of spatial indices of the voxels of the Voxel Collection

$$G = \{(i, j, k),\ V_{ijk} \in \{Voxel{-}Collection\}\}$$

V_{ijk} represents a unit cube with the vertices $(i + \delta_1, j + \delta_2, k + \delta_3)$, with $\delta_1, \delta_2, \delta_3$ being either 0 or 1. Observe that the cardinality of G grows in the same manner as the size of the object does (quadratically if we are modeling a surface).

On the other hand, let us define I as the set of spatial indices of the voxels in the immediate vicinity of the Voxel Collection:

$$I = \{(i, j, k)\ with\ (i + \delta_1, j + \delta_2, k + \delta_3) \in G,\ \delta_1, \delta_2, \delta_3 \in \{-2, -1, 0, +1\}\}$$

The cardinality of I is of the order of 4 times the cardinality of G. Let us now consider that $F(x, y, z)$ is a uniform, tensor-product cubic Bspline function. We are only interested in evaluating it on the Voxel Collection, to find its zero-isosurface:

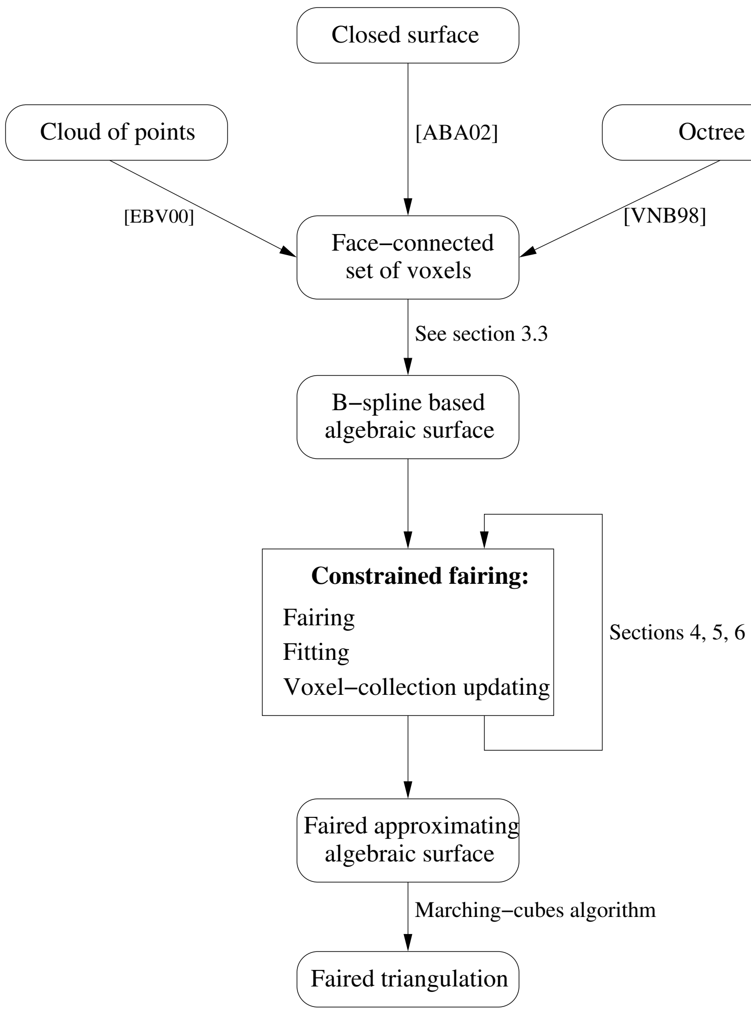


Figure 2: Model conversions around the constrained fairing process

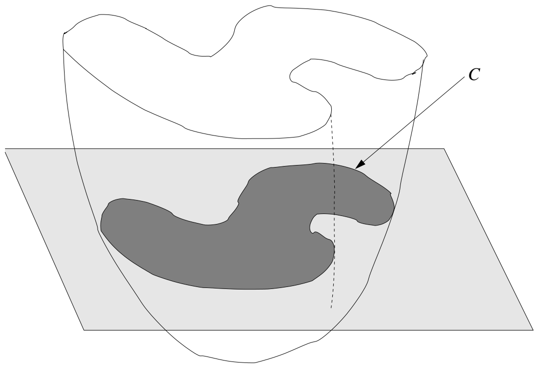


Figure 3: The intersection of the 3D Bspline function with the plane $z = 0$ defines a curve

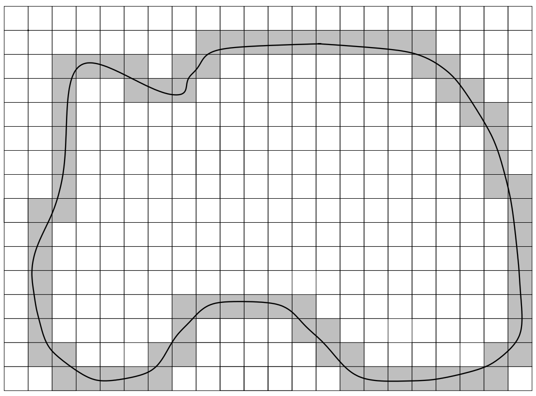


Figure 4: The Voxel Collection is the set of shaded voxels

$$F(x,y,z) = \sum_{(i,j,k)\in I} d_{i,j,k}\ N_i^3(x)\ N_j^3(y)\ N_k^3(z) = 0$$

where $d_{i,j,k}$ is the weight at the grid point (i,j,k). It is a smooth, C^2, piecewise tricubic surface.

In this paper we will work with tricubic Bspline surfaces. In section 7 we will adapt the algorithm to deal with trilinear Bspline surfaces.

3.3 Algorithm

First, we compute an initial piecewise tricubic Bspline algebraic surface from the discrete membrane. This can be easily achieved creating a Voxel Collection with all voxels of the discrete membrane and setting the weights of the Bspline function to $+1, 0, -1$ at the grid points defined by the set of spatial indices I. We set them to $+1$ at the grid points touching an outside voxel, -1 at the grid points touching an inside voxel and 0 in grid points having all his 8 surrounding voxels in the discrete membrane. See in figure 5 a 2D example.

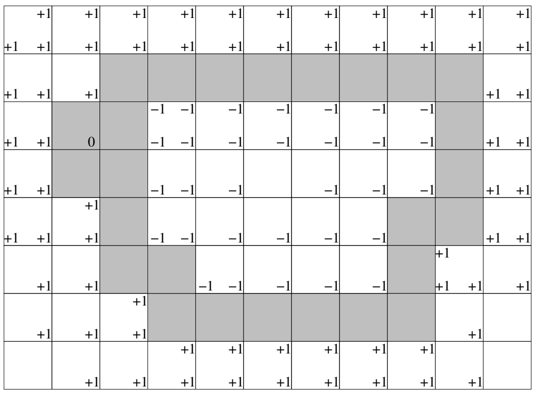


Figure 5: Setting the weights of the Bspline function from a 2D discrete membrane

Alternatively, to start with smoother surfaces, the weights can be assigned in a more flexible way. The weights, always in the interval $[-1, 1]$, will be set depending on the number of outside/inside voxels there are at the 8 surrounding voxels of the grid point. If there are a lot of inside/outside voxels, the weight will lay near to $+1/-1$. If there are few, the weight will lay near to 0. This is the option we have used in our implementation.

Then, the algorithm iterates the following three steps:

1. **Fairing process**: The fairing process is based on increasing the surface continuity from C^2 to C^3 at the boundary faces of the voxels. It is implemented with a local filter that solves a constrained minimization problem. The size of this filter can be tuned to improve performance. The best results are obtained using a wide filter in the first iterations and switching to a narrow one in the last steps. The details are given in sections 4 and 8.

2. **Fitting process**: The fitting process consists of pushing the algebraic isosurface towards the central point of the hard voxels. The weights of each hard voxel are updated with a local filter. This filter minimizes the difference between old and new weights and the distance from the surface to the central point of the hard voxel. The process is discussed in detail in section 5.

3. **Voxel Collection updating**: We must guarantee that the isosurface is inside the Voxel Collection. We thus add to the Voxel Collection those voxels where the surface wanders outside of the initial set. The boundary faces of the Voxel Collection stabbed by the surface are detected and new voxels are added to the Voxel Collection in these places. This is discussed in section 6.

The algorithm stops when the surface is sufficiently smooth. This can be achieved in few iterations (see section 8). Next sections describe these three tasks in more detail.

4 Fairing by increasing the continuity between adjacent voxels

Our smoothing algorithm for cubic Bsplines is based on the idea of increasing their continuity from C^2 to C^3 in a segment between two neighboring patches presented in [Hahmann(1998)]. Our main extensions and contributions in this respect are:

- The fairing principles and algorithms have been extended from bicubic surfaces to tricubic hypersurfaces. Instead of performing the fairing process in a segment between two patches, it is done in a face between two adjacent voxels.
- The linear system that performs the fairing has been factorized yielding a smaller filter. A lower number of weights must be processed at one time.
- The fairing process can be performed simultaneously at the intermediate faces of a variable set of consecutive voxels. So, the locality of the fairing filter can be tuned.

The fairing algorithm is based on the following fairing principles:

Fairing principle 1: A tricubic Bspline function $F(x, y, z)$ of class C^2 is fair at the grid point (u_l, v_m, w_n), $(l, m, n) \in G$, if F is C^3 at (u_l, v_m, w_n).

This is computationally inexpensive (see equations 2 and 3) for Bspline surfaces and hypersurfaces and has the advantage of being local. Furthermore, it only affects one grid point (u_l, v_m, w_n). It can be extended like the 1-Segment fairing step of [Hahmann(1998)]: instead of increasing the smoothness of F at one grid point, it can be increased between two adjacent voxels of the Voxel Collection.

The (μ, ν, σ) partial derivative of F will be denoted as

$$F_{x^\mu y^\nu z^\sigma}(x, y, z) \;=\; \frac{\partial^{\mu+\nu+\sigma} F(x, y, z)}{\partial x^\mu \; \partial y^\nu \; \partial z^\sigma}.$$

The derivatives of a Bspline function can be expressed as a Bspline function of lower degree whose weights $d_{i,j,k}^{(\mu,\nu,\sigma)}$ are linear combinations of the initial weights $d_{r,s,t}$ (see [Farin(1992)]):

$$F_{x^\mu y^\nu z^\sigma}(x, y, z) \;=\; \mu!\nu!\sigma! \sum_i \sum_j \sum_k d_{i,j,k}^{(\mu,\nu,\sigma)} \; N_i^{3-\mu}(x) \; N_j^{3-\nu}(y) \; N_k^{3-\sigma}(z)$$

All mixed third order partial derivatives of F are continuous because $N_i^{3-\mu}(x)$, $N_j^{3-\nu}(y)$ and $N_k^{3-\sigma}(z)$ are at least C^0, if $\mu \leq 2$ and $\nu \leq 2$ and $\sigma \leq 2$ $(\mu + \nu + \sigma = 3)$.

The common face of two voxels aligned, for example, in the Z-direction and located on the grid point (u_l, v_m, w_n) is defined by the parameter domain $[u_l, u_{l+1}] \times [v_m, v_{m+1}]$ (see figure 6). The third order partial derivatives in the Z-direction define a bicubic surface patch at left of the common boundary face and a different bicubic surface patch at right:

$$\begin{aligned}
F_{z^3}(x, y, w_n^-) &= \sum_{i=l-1}^{l+2} \sum_{j=m-1}^{m+2} d_{i,j,n-1}^{(0,0,3)} \; N_i^3(x) \; N_j^3(y) \\
F_{z^3}(x, y, w_n^+) &= \sum_{i=l-1}^{l+2} \sum_{j=m-1}^{m+2} d_{i,j,n}^{(0,0,3)} \; N_i^3(x) \; N_j^3(y) \\
& x \in [u_l, u_{l+1}[\\
& y \in [v_m, v_{m+1}[
\end{aligned}$$

Fairing principle 2: A tricubic Bspline function $F(x, y, z)$ of class C^2 is fair at the common boundary face of two adjacent voxels, if $F_{z^3}(x, y, w_n^-) = F_{z^3}(x, y, w_n^+)$ (see figure 6).

These two bicubic surface patches need to be identical to get C^3 continuity at the common boundary face. Therefore, the 16 weights $d_{i,j,n-1}^{(0,0,3)}$ that define $F_{z^3}(x, y, w_n^-)$ and the 16 weights $d_{i,j,n}^{(0,0,3)}$ that define $F_{z^3}(x, y, w_n^+)$ must be equal:

$$d_{i,j,n-1}^{(0,0,3)} \;=\; d_{i,j,n}^{(0,0,3)}, \qquad \begin{array}{l} i = l-1, \ldots, l+2 \\ j = m-1, \ldots, m+2 \end{array} \tag{1}$$

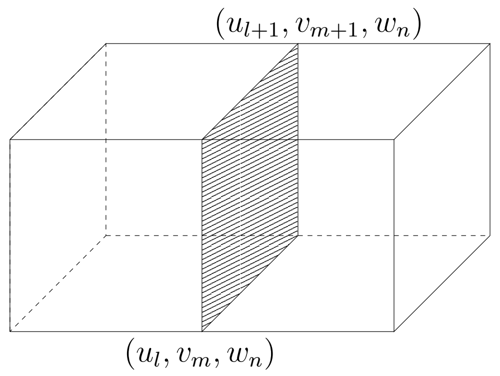


Figure 6: Common boundary face of two adjacent voxels

4.1 Local fairing filter (two voxels)

The problem can be approached as a constrained least-squares approximation on the $4 \times 4 \times 5 = 80$ weights that define the Bspline function in the two adjacent voxels. If $d_{i,j,k}$ denotes the old weights and $\widehat{d}_{i,j,k}$ the new ones, the expression

$$D(\widehat{d}_{i,j,k}) \;=\; \sum_{i=l-1}^{l+2} \sum_{j=m-1}^{m+2} \sum_{k=n-2}^{n+2} \parallel d_{i,j,k} - \widehat{d}_{i,j,k} \parallel^2$$

is a measure of the perturbation of the surface. We thus minimize $D(\widehat{d}_{i,j,k})$ subject to the 16 constraints of equations 1.

The size of this least-squares approximation problem and the associated matrix or filter can be reduced. Every constraint in equation 1 is independent of the others because each one is a linear combination of disjoint groups of weights $d_{i,j,k}$. For example, for a fixed i and j, the constraint $d^{(0,0,3)}_{i,j,n-1} \;=\; d^{(0,0,3)}_{i,j,n}$ is a relation between five weights $d_{i,j,n-2}, \ldots, d_{i,j,n+2}$ and none of the other constraints have a dependence on any of these:

$$d^{(0,0,3)}_{i,j,n-1} - d^{(0,0,3)}_{i,j,n} \;=\; d_{i,j,n-2} - 4d_{i,j,n-1} + 6d_{i,j,n} - 4d_{i,j,n+1} + d_{i,j,n+2} \;=\; 0 \qquad (2)$$

Therefore, the fairing can be performed filtering groups of five weights at a time. The expression

$$D(\widehat{d}_{i,j,k}) \;=\; \sum_{k=n-2}^{n+2} \parallel d_{i,j,k} - \widehat{d}_{i,j,k} \parallel^2$$

must be minimized subject to one constraint (equation 2). Applying the technique of Lagrange multipliers we must compute the minimum of this expression:

$$\Phi(\widehat{d}_{i,j,k}, \lambda) \;=\; D(\widehat{d}_{i,j,k}) \;+\; \lambda \left(d^{(0,0,3)}_{i,j,n-1} - d^{(0,0,3)}_{i,j,n} \right)$$
$$\frac{\partial \Phi}{\partial \widehat{d}_{i,j,k}} = 0 \qquad \frac{\partial \Phi}{\partial \lambda} = 0$$

This yields a linear system of 6 equations that defines the local fairing filter:

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 0 & -4 \\ 0 & 0 & 2 & 0 & 0 & 6 \\ 0 & 0 & 0 & 2 & 0 & -4 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 1 & -4 & 6 & -4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \widehat{d}_{i,j,n-2} \\ \widehat{d}_{i,j,n-1} \\ \widehat{d}_{i,j,n} \\ \widehat{d}_{i,j,n+1} \\ \widehat{d}_{i,j,n+2} \\ \lambda \end{bmatrix} = \begin{bmatrix} 2d_{i,j,n-2} \\ 2d_{i,j,n-1} \\ 2d_{i,j,n} \\ 2d_{i,j,n+1} \\ 2d_{i,j,n+2} \\ 0 \end{bmatrix} \tag{3}$$

4.2 Local fairing filter (set of consecutive voxels)

The previous linear system represents a very local filter. In the first steps of the fairing process, it is more convenient to apply wider filters to smooth the surface faster. It is straightforward to extend the filter domain. For example, if we have six consecutive weights defined in the Z-direction $d_{i,j,n-2}, \ldots, d_{i,j,n+3}$ (these weights contribute to the function in 3 consecutive voxels), two constraints can be fixed to get C^3 continuity at the two boundary faces of the 3 consecutive voxels:

$$d^{(0,0,3)}_{i,j,n-1} = d^{(0,0,3)}_{i,j,n}$$

$$d^{(0,0,3)}_{i,j,n} = d^{(0,0,3)}_{i,j,n+1}$$

This yields a linear system with 8 unknowns that defines the local fairing filter for three consecutive voxels:

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & -4 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 6 & -4 \\ 0 & 0 & 0 & 2 & 0 & 0 & -4 & 6 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 \\ 1 & -4 & 6 & -4 & 1 & 0 & 0 & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \widehat{d}_{i,j,n-2} \\ \widehat{d}_{i,j,n-1} \\ \widehat{d}_{i,j,n} \\ \widehat{d}_{i,j,n+1} \\ \widehat{d}_{i,j,n+2} \\ \widehat{d}_{i,j,n+3} \\ \lambda_0 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} 2d_{i,j,n-2} \\ 2d_{i,j,n-1} \\ 2d_{i,j,n} \\ 2d_{i,j,n+1} \\ 2d_{i,j,n+2} \\ 2d_{i,j,n+3} \\ 0 \\ 0 \end{bmatrix}$$

If the length of the interval to filter is L, the filter size is $2 * L - 4$. Our algorithm uses a battery of precomputed filters (the inverses of the previous matrices). The fairing process cyclically filters the defined weights in the directions X, Y and Z to perform the fairing on the voxel boundary faces aligned in the X, Y and Z directions. The size of the filter used depends on the number of consecutive defined weights. We have observed that filters wider than 10 coefficients do not improve the results. So, when the number of consecutive weights is higher than 10, the weights are filtered in subsets of 10 consecutive weights with an overlap of 4.

5 Fitting the surface to hard voxels

The surface is displaced slightly in the fairing step. If the voxels of the discrete membrane are tagged as hard voxels and soft voxels, a fitting step has been designed that attempts to displace the surface to restore proper stabbing of hard voxels.

Let us suppose the piecewise tricubic Bspline algebraic surface has to stab the hard voxel V_{lmn} located at the grid point (u_l, v_m, w_n). Let (x_p, y_p, z_p) be the point inside voxel V_{lmn} that we want the surface to approximate (see figure 7). This point is usually not known, in which case we may use the central point of the hard voxel. We wish to modify the weights so that the point (x_p, y_p, z_p) is a zero of the Bspline function

$$F(x_p, y_p, z_p) \;=\; \sum_{i=l-1}^{l+2} \sum_{j=m-1}^{m+2} \sum_{k=n-1}^{n+2} d_{i,j,k}\; N_i^3(x_p)\; N_j^3(y_p)\; N_k^3(z_p) \;=\; 0$$

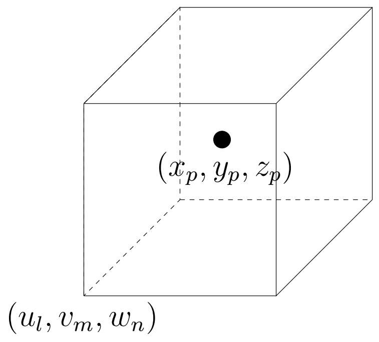


Figure 7: Hard voxel: the surface has to pass near the point (x_p, y_p, z_p)

A least-squares approximation with this constraint could be established. But it is better to solve the problem in a more flexible way, especially since the point (x_p, y_p, z_p) is only approximated by the central point of the voxel. The value $F^2(x_p, y_p, z_p)$ locally grows with the distance from the surface to the point (x_p, y_p, z_p). So, an unconstrained least-squares approximation that minimizes the function $F^2(x_p, y_p, z_p)$ can be stated:

$$\begin{aligned}
\Phi(\widehat{d}_{i,j,k}) \quad &= \quad \alpha \sum_{i=l-1}^{l+2} \sum_{j=m-1}^{m+2} \sum_{k=n-1}^{n+2} \parallel d_{i,j,k} - \widehat{d}_{i,j,k} \parallel^2 \\
&\quad + (1-\alpha) \left(\sum_{i=l-1}^{l+2} \sum_{j=m-1}^{m+2} \sum_{k=n-1}^{n+2} \widehat{d}_{i,j,k}\; N_i^3(x_p)\; N_j^3(y_p)\; N_k^3(z_p) \right)^2
\end{aligned}$$

α is a factor that weighs up the two terms: values closer to one reduce surface displacements, whereas values closer to zero will tend to fit the surface to the point (x_p, y_p, z_p) regardless of the magnitude of its perturbation.

A tricubic Bspline voxel V_{lmn} is defined by $4 \times 4 \times 4 = 64$ weights. To find the minimum solution, the partial derivatives $\frac{\partial \Phi}{\partial \widehat{d}_{i,j,k}} = 0$ are computed yielding a linear system of 64 equations:

$$\begin{aligned}\alpha\ d_{i,j,k} &= \alpha\ \widehat{d}_{i,j,k} \\ &+(1-\alpha)\left(\sum_{r=l-1}^{l+2}\sum_{s=m-1}^{m+2}\sum_{t=n-1}^{n+2}\widehat{d}_{r,s,t}N_r^3(x_p)\ N_s^3(y_p)\ N_t^3(z_p)\right)\ N_i^3(x_p)N_j^3(y_p)N_k^3(z_p)\end{aligned}$$

$$\begin{aligned}i &= l-1,\dots,l+2\\ j &= m-1,\dots,m+2\\ k &= n-1,\dots,n+2\end{aligned}$$

If the point (x_p, y_p, z_p) is approximated by the central point of the voxel, all the basis functions $N_i^3(x_p), N_j^3(y_p), N_k^3(z_p)$ can be calculated beforehand. For a fixed α, the matrix and its inverse are precomputed. This filter is only applied to the hard voxels.

6 Voxel Collection updating

The algebraic surface moves during the fairing and fitting processes. There is no control to prevent the surface from drifting to the exterior of the Voxel Collection. This is acceptable in the vicinity of the soft voxels of the initial discrete membrane. This flexibility is important because the initial surface computed from a discrete membrane has a stepping shape due to the aliasing of the previous discrete model. A smoother surface can be obtained if the surface is allowed to move to the voxels in the vicinity of the Voxel Collection.

However, it is necessary to update the Voxel Collection adding those voxels that share a common boundary face with the Voxel Collection and have zeroes of the Bspline function at the common face. To avoid computing the function in all the points of the boundary face to find out if there is a zero, we use the convex hull property.

For each border face of the Voxel Collection, the 16 coefficients $\overline{d}_{i,j}$ that define the Bspline function on this face are computed. These 16 coefficients define a bicubic Bspline surface. For example, if the shaded face in figure 8 is a border face of the Voxel Collection,

$$F(x,y,w_n) = \sum_{i=l-1}^{l+2}\sum_{j=m-1}^{m+2}\overline{d}_{i,j}\ N_i^3(x)\ N_j^3(y)$$

$$\text{where } \overline{d}_{i,j} = \sum_{k=n-1}^{n+2} d_{i,j,k}\ N_k^3(w_n)$$

A simple test would use the convex hull property on these faces. If the face is touching an outside voxel, one would check that all sixteen coefficients are positive. If it is touching an inside voxel, one would check for negative values.

This, however, is a very conservative test. Many voxels would be added due to the convex hull check that are not really stabbed by the surface. The drawback would be that the fairing process has to deal with more data in the next iterations.

Therefore we use a simple but less conservative test. We divide the bicubic Bspline surface domain into 2×2 equal regions (A, B, C and D, see figure 8) and the maximum and minimum

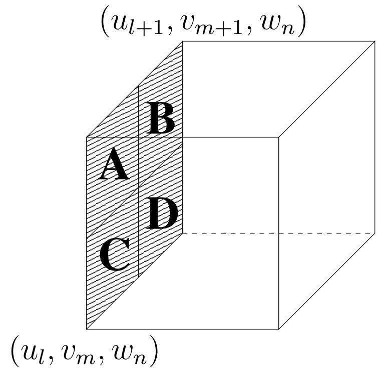


Figure 8: The shaded face is a border face of the Voxel Collection

of each bicubic Bspline basis function $N_i^3(x)N_j^3(y)$ $i, j = 0, \dots, 3$ in each of these four regions are tabulated. These values are shown in tables 1 and 2. Only the first 4 basis functions are shown because the rest are obtained by symmetry.

Table 1: Minima of the first 4 bicubic Bspline basis functions. Each 2×2 tile of the table shows the minimum of the product of two basis functions in each of the four regions A, B, C and D.

	$N_0^3(y)$		$N_1^3(y)$	
$N_0^3(x)$	1/2304	0	23/2304	1/288
	0	0	0	0
$N_1^3(x)$	23/2304	0	529/2304	23/288
	1/288	0	23/288	1/36

From the 16 coefficients that define the bicubic Bspline function at the boundary face, values are computed using the tables 1 and 2 for each region as we will indicate shortly. The function will not have a zero at the boundary face if values have the same sign and agree with the type (inside/outside) of the neighbor voxel.

For example, to test if a face of the Voxel Collection touching an outside voxel is stabbed by the surface, the positive weights are multiplied by the minima of the corresponding basis functions and the negative weights by the maxima. Each one of the four regions is handled independently. For example, for region A we compute:

$$R_A = \sum_{i,j\ /\ \overline{d}_{i,j}>0} \overline{d}_{i,j}\ min_A(N_i^3(x)N_j^3(y)) + \sum_{i,j\ /\ \overline{d}_{i,j}<0} \overline{d}_{i,j}\ max_A(N_i^3(x)N_j^3(y))$$

If the four results are positive there are no zeroes of the function at the boundary face, otherwise the outside voxel will be added to the Voxel Collection by setting the value of the unknown weights that define the function in this voxel. The weights are set in the same way as in section 3.3.

This test has a similar cost to the previous convex hull test; in both of them the most expensive

Table 2: Maxima of the first 4 bicubic Bspline basis functions. Each 2×2 tile of the table shows the maximum of the product of two basis functions in each of the four regions A, B, C and D.

	$N_0^3(y)$		$N_1^3(y)$	
$N_0^3(x)$	1/36	1/288	1/9	23/288
	1/288	1/2304	1/72	23/2304
$N_1^3(x)$	1/9	1/72	4/9	23/72
	23/288	23/2304	23/72	529/2304

task is the computation of the 16 coefficients of the function restricted to the boundary face from the 64 weights of a voxel.

The improvement of this test with respect to the simple convex hull test is high (see tables 3 and 4 in section 8).

7 Trilinear uniform Bspline isosurface

All previous arguments and algorithms can be adapted to deal with piecewise trilinear Bspline algebraic surfaces with the consequent savings in computational cost. First, we define I_l as the set of spatial indices formed by all the voxel vertices of the Voxel Collection:

$$I_l = \{(i,j,k) \ with \ (i+\delta_1, j+\delta_2, k+\delta_3) \in G, \ \delta_1, \delta_2, \delta_3 \in \{-1, 0\}\}$$

Let us now consider the uniform, tensor-product linear Bspline function F_l defined on the regular grid. F_l is an integral, functional spline and we are only interested in evaluating it on the Voxel Collection, to find its zero-isosurface. It is a piecewise trilinear surface:

$$F_l(x,y,z) = \sum_{(i,j,k)\in I_l} d_{i,j,k} \ N_i^1(x) \ N_j^1(y) \ N_k^1(z) = 0$$

The trilinear surfaces are only C^0 continuous and do not have enough smoothness to achieve a high quality rendering. However, they can be very useful as an intermediate model. For example, to get a constrained smooth triangular mesh from a discrete membrane: a trilinear surface is computed from the discrete membrane, later it is smoothed with our algorithm and finally a triangular mesh is computed using, for example, the Marching-Cubes algorithm.

To work with trilinear algebraic surfaces we proceed in a similar way to build the initial trilinear surface from a discrete membrane (see section 3.3). We set the weights in the grid points defined by the set of spatial indices I_l instead of I, whose cardinality is of the order of 2 times lower than the cardinality of I.

The voxel face fairing process is implemented with filters of lower size. For example, with 3 consecutive weights (they contribute to the function in 2 consecutive voxels) we have the linear system:

$$\begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & -2 \\ 0 & 0 & 2 & 1 \\ 1 & -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} \widehat{d}_{i,j,n-1} \\ \widehat{d}_{i,j,n} \\ \widehat{d}_{i,j,n+1} \\ \lambda \end{bmatrix} = \begin{bmatrix} 2d_{i,j,n-1} \\ 2d_{i,j,n} \\ 2d_{i,j,n+1} \\ 0 \end{bmatrix}$$

The constraint embedded in this system to get C^1 continuity in the common face is

$$\widehat{d}_{i,j,n-1} - 2\widehat{d}_{i,j,n} + \widehat{d}_{i,j,n+1} = 0$$

This constraint says that the middle weight $\widehat{d}_{i,j,n}$ is an average of the weights located at its right and left. Being L the length of the interval to filter, the filter size is $2 * L - 2$.

The fitting process is performed with a filter of size 8 instead of 64. It is also applied to the hard voxels of the initial discrete membrane and it uses a factor α to weight up the difference between old and new weights and the distance from the surface to the central point of the hard voxel.

In this case, the Voxel Collection updating is drastically simplified. Due to the nature of the linear Bspline basis (in each grid point there is only one basis function that does not vanish), we can use the convex hull property to test exactly if there is a zero in a voxel face. We have only to check the sign of the 4 weights located at the voxel face. If all 4 weights have the same sign, there is no zero of the function in this voxel face.

8 Results and discussion

We have tested the proposed algorithm over several discrete models. All discrete membranes have been obtained from point clouds using the discrete membrane shrinking algorithm described in [Esteve et al.(2005)Esteve, Brunet, and Vinacua]. Figure 9 shows an oil pump model: first the discrete membrane, then the initial tricubic Bspline isosurface and finally the same surface after 10 iterations of constrained smoothing with a factor $\alpha = 0.5$. The red voxels are hard voxels (they contain one or more points of the initial set of points), the blue ones are soft voxels without data points. This model is defined over a voxelization of size $109 \times 89 \times 125$.

Figure 10 shows a model of a toy bird. The tricubic Bspline surface has been smoothed during 10 iterations with a factor $\alpha = 0.7$ (10c) and $\alpha = 0.3$ (10d). With higher values of α the surface is not strongly fitted to stab the central point of the hard voxels, getting a smoother surface.

The initial discrete membrane of the bird has a size of $154 \times 210 \times 204$. The initial tricubic Bspline piecewise algebraic surface has 526,131 weights defined. Table 3 shows the number of voxels added in each iteration using the convex hull test and our improved test. Using the improved test 30,928 voxels are added to the Voxel Collection (the final surface has 34,001 new weights yielding an increase of 6.5%), whereas using the convex hull test 160,513 voxels are added (the final surface would have 219,671 new weights that represent a 41.8% of increase).

The last example of tricubic Bspline surfaces is the buddha model shown in figure 11. The constrained fairing algorithm has been applied 15 times with a factor $\alpha = 0.5$. The voxeliza-

Table 3: Voxels added in the Voxel Collection updating stage. Bird model of figure 10c.

Iteration	Convex hull test	Improved test
1	142,456	17,769
2	7,524	2,153
3	1,755	1,657
4	1,340	1,523
5	1,400	1,477
6	1,409	1,469
7	1,167	1,325
8	1,190	1,279
9	1,233	1,153
10	1,039	1,123
TOTAL	160,513	30,928

tion size of the buddha model is $310 \times 746 \times 310$. Table 4 shows the number of voxels added in each iteration using the convex hull test (43% of new weights) and our improved test (7,9% of new weights).

Table 4: Voxels added in the Voxel Collection updating stage. Buddha model of figure 11.

Iteration	Convex hull test	Improved test
1	1,201,752	148,563
2	73,944	24,926
3	21,022	18,344
4	15,033	16,322
5	13,421	14,521
6	12,267	13,377
7	11,445	12,034
8	10,356	10,560
9	9,738	9,653
10	9,020	8,561
11	8,313	7,769
12	7,708	6,906
13	7,235	6,416
14	6,466	5,749
15	5,992	5,156
TOTAL	1,413,712	308,857

Figure 12 shows a trilinear Bspline surface computed from the same discrete membrane of figure 10. The trilinear Bspline surface has been smoothed during 10 iterations with a factor $\alpha = 0.7$. We have used the same parameters to compare the results. The rendering of the final surface has lower quality but is sufficiently good to extract a triangular mesh from it that can be guaranteed to not suffer any topological artifacts (like holes or false handles). Figure 13 is the trilinear Bspline surface computed from the buddha model. In both cases the surface appearance has been improved applying two fairing steps with narrow filters at the end of the process.

The run-times of the previous constrained fairing processes obtained with an AMD-Athlon XP 2000+ CPU and 256Mb of main memory are listed in table 5. The α factor does not affect to the run-times: the bird tricubic surface computed with $\alpha = 0.3$ has the same cost as $\alpha = 0.7$. The high computational cost in the buddha model, apart from applying more iterations, is due to its large voxelization size and the high curvature and genus of the surface yielding a big Voxel Collection. We can observe the lower computational cost using trilinear surfaces instead of tricubic ones in the bird and buddha model. The time cost is approximately reduced by a factor of 5 if trilinear surfaces are used instead of tricubic.

Table 5: Run-times of the constrained fairing process

Model	Parameters	Time (h:mm:ss)
Oil pump (tricubic)	10 iter. $\alpha = 0.5$	0:02:54
Bird (tricubic)	10 iter. $\alpha = 0.7$	0:06:55
Bird (trilinear)	10 iter. $\alpha = 0.7$	0:01:12
Buddha (tricubic)	15 iter. $\alpha = 0.5$	1:39:47
Buddha (trilinear)	15 iter. $\alpha = 0.5$	0:21:50

9 Conclusions and future work

We have presented a method for fairing a piecewise algebraic surface while constraining it to a discrete membrane. The surface is smoothed by increasing its continuity from C^2 to C^3 on the boundary faces of the voxels. Smoothing iterations are alternated with surface fitting iterations that keep it within the desired region by pulling it towards the central point of the "hard" voxels.

This method is appropriated for obtaining smooth, C^2 continuous surfaces using tricubic Bspline basis functions. It is also a good and fast method to get a smoother discrete model, like a triangular mesh, using trilinear Bspline surfaces as an intermediate model. It can be used, for example, to fix the topological problems of triangular meshes. We have performed this task with the triangular mesh of the buddha model. Our final model does not contain the spurious handles present in the original data and has all the holes filled.

Another interesting application is the noise removal of, for example, medical data.

References

[Andújar et al.(2002)Andújar, Brunet, and Ayala] C. Andújar, P. Brunet, and D. Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics*, 21(2):88–105, April 2002.

[Esteve et al.(2001)Esteve, Brunet, and Vinacua] J. Esteve, P. Brunet, and A. Vinacua. Multiresolution for algebraic curves and surfaces using wavelets. *Computer Graphics Forum*, 20(1):47–58, 2001.

[Esteve et al.(2005)Esteve, Brunet, and Vinacua] J. Esteve, P. Brunet, and A. Vinacua. Approximation of a cloud of points by shrinking a discrete membrane. *Computer Graphics Forum*, 24(4), 2005.

[Farin(1992)] G. Farin. *Curves and surfaces for computer aided geometric design*. Academic Press, 3rd edition edition, 1992.

[Farin et al.(1987)Farin, Rein, Sapidis, and Worsey] G. Farin, G. Rein, N. Sapidis, and A. J. Worsey. Fairing cubic bspline curves. *Computer Aided Geometric Design*, 4:91–103, 1987.

[Hadenfeld(1995)] J. Hadenfeld. Local energy fairing of b-spline surfaces. In M. Daehlen, T. Lyche, and L. L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, pages 203–212, Nashville and London, 1995. Vanderbilt University Press.

[Hahmann(1998)] S. Hahmann. Shape improvement of surfaces. *Computing*, 13:135–152, 1998.

[Hahmann and Konz(1998)] S. Hahmann and S. Konz. Knot-removal surface fairing using search strategies. *Computer Aided Design*, 30:131–138, 1998.

[Kjellander(1983a)] J. A. Kjellander. Smoothing of bicubic parametric surfaces. *Computer Aided Design*, 15:289–293, 1983a.

[Kjellander(1983b)] J. A. Kjellander. Smoothing of cubic parametric splines. *Computer Aided Design*, 15:175–179, 1983b.

[Klass(1980)] R. Klass. Correction of local irregularities using reflection lines. *Computer Aided Design*, 12:73–77, 1980.

[Lott and Pullin(1988)] N. J. Lott and D. I. Pullin. Method for fairing b-spline surfaces. *Computer Aided Design*, 20:597–604, 1988.

[Moreton and Séquin(1994)] H. P. Moreton and C. H. Séquin. Minimum variation curves and surfaces for computer aided geometric design. In N. S. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 123–159, Philadelphia, 1994. SIAM.

[Patrikalakis and Kriezis(1989)] N. M. Patrikalakis and G. A. Kriezis. Representation of piecewise continous algebraic surfaces in terms of bsplines. *The visual computer*, 5:360–374, 1989.

[Poliakoff(1996)] J. F. Poliakoff. An improved algorithm for automatic fairing of non-uniform parametric cubic splines. *Computer Aided Design*, 28:59–66, 1996.

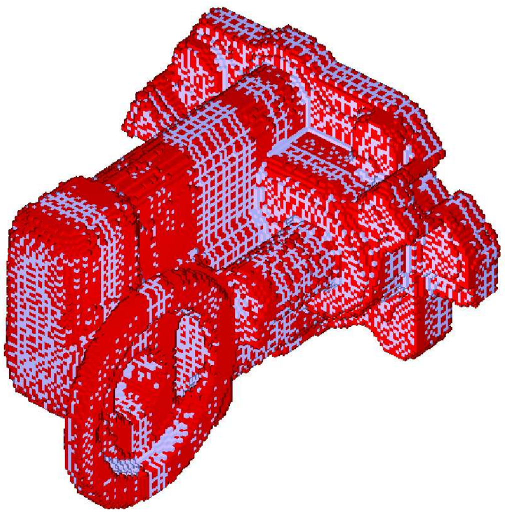
[Rando and Roulier(1991)] T. Rando and J. A. Roulier. Designing faired parametric surfaces. *Computer Aided Design*, 23:492–497, 1991.

[Sederberg(1984)] T. W. Sederberg. Planar piecewise algebraic curves. *Computer Aided Geometric Design*, 1:241–255, 1984.

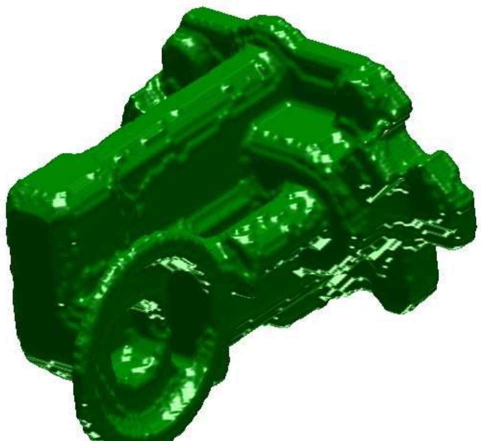
[Sederberg(1985)] T. W. Sederberg. Piecewise algebraic surface patches. *Computer Aided Geometric Design*, 2:53–59, 1985.

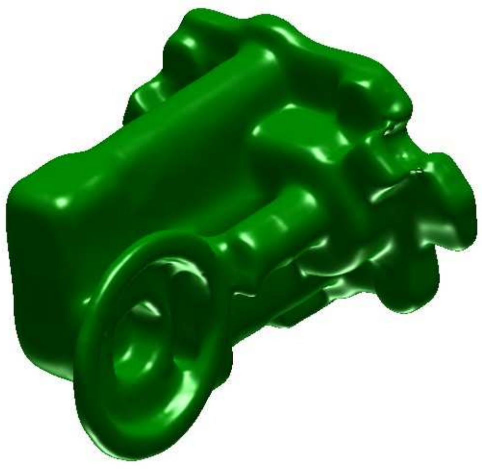
[Sramek and Kaufman(1999)] M. Sramek and A. E. Kaufman. Alias-free voxelization of geometric objects. *IEEE transactions on visualization and computer graphics*, 5(3):251–267, 1999.

[Vinacua et al.(1998)Vinacua, Navazo, and Brunet] A. Vinacua, I. Navazo, and P. Brunet. Octtrees meet splines. In G. Farin, H. Bieri, G. Brunett, and T. DeRose, editors, *Geometric Modelling, Computing [Suppl]*, volume 13, pages 225–233. Springer-Verlag, 1998.



(a) Discrete membrane

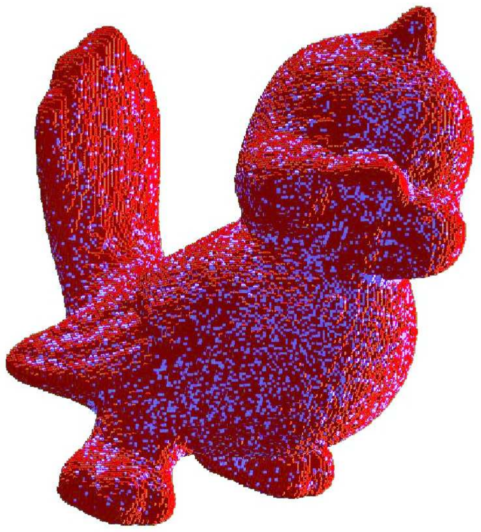


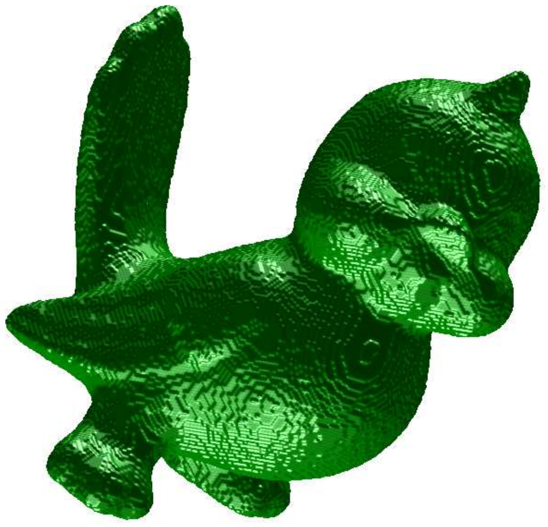


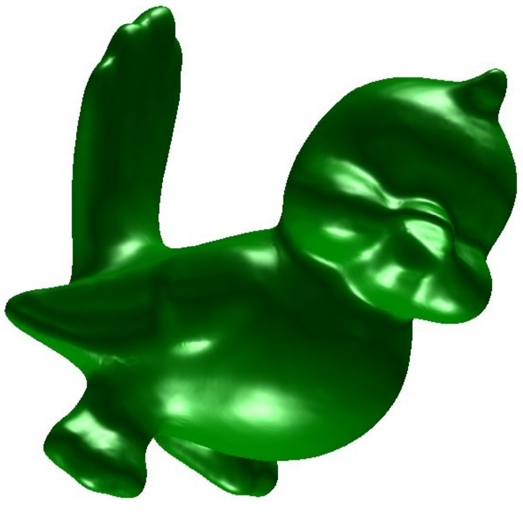
(b) Initial surface

(c) Smoothed surface. 10 iterations. $\alpha = 0.5$

Figure 9: Tricubic Bsplines. Oil pump model


(a) Discrete membrane


(b) Initial surface


(c) Smoothed surface. 10 iterations. $\alpha = 0.7$

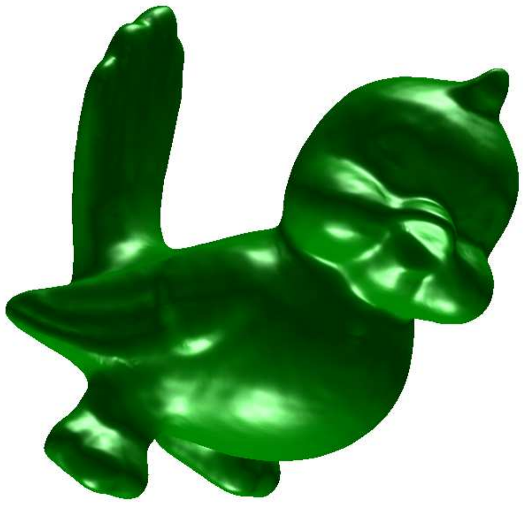
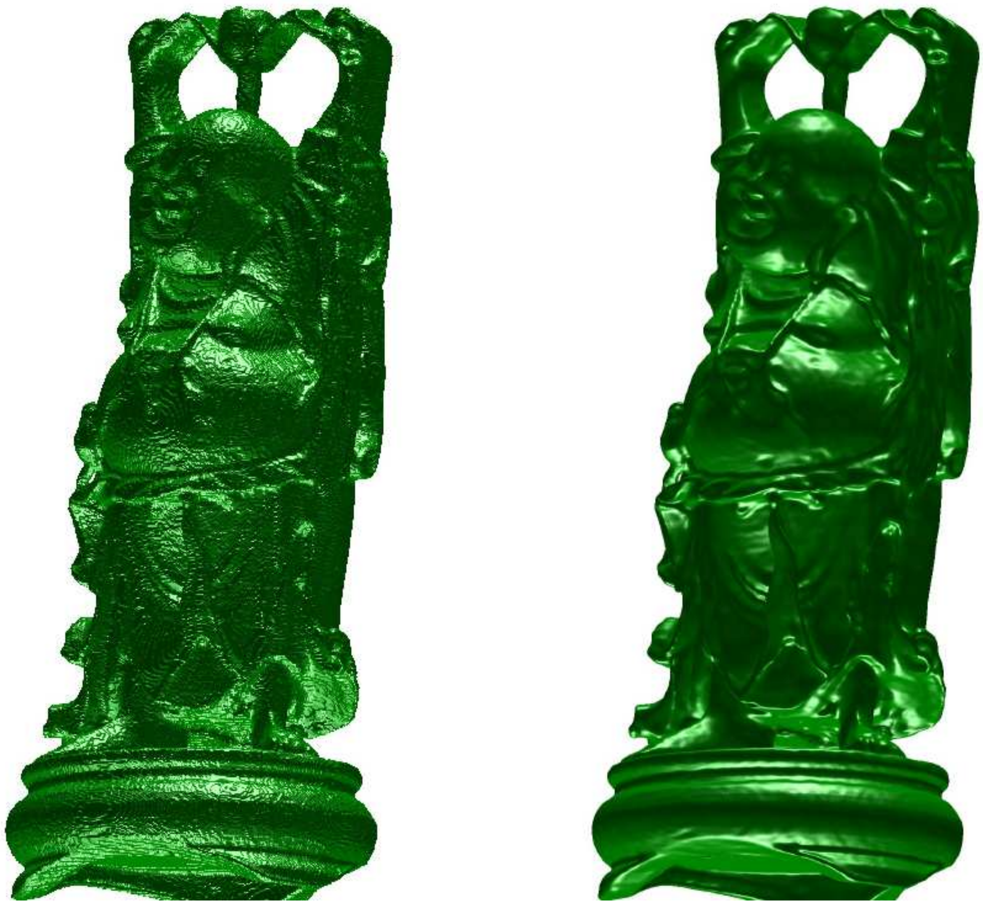

(d) Smoothed surface. 10 iterations. $\alpha = 0.3$

Figure 10: Tricubic Bsplines. Bird model



(a) Initial surface (b) Smoothed surface. 15 iterations. $\alpha = 0.5$

Figure 11: Tricubic Bsplines. Buddha model

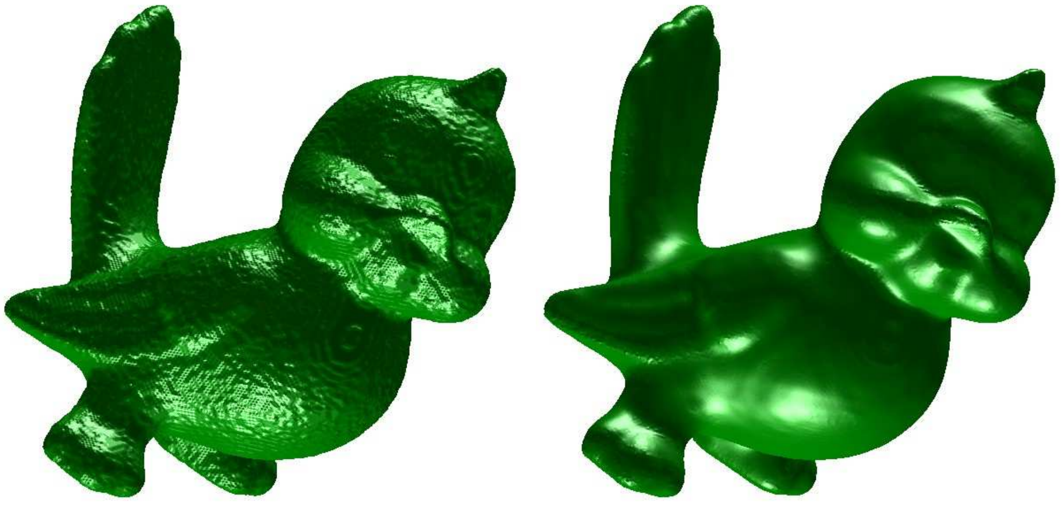
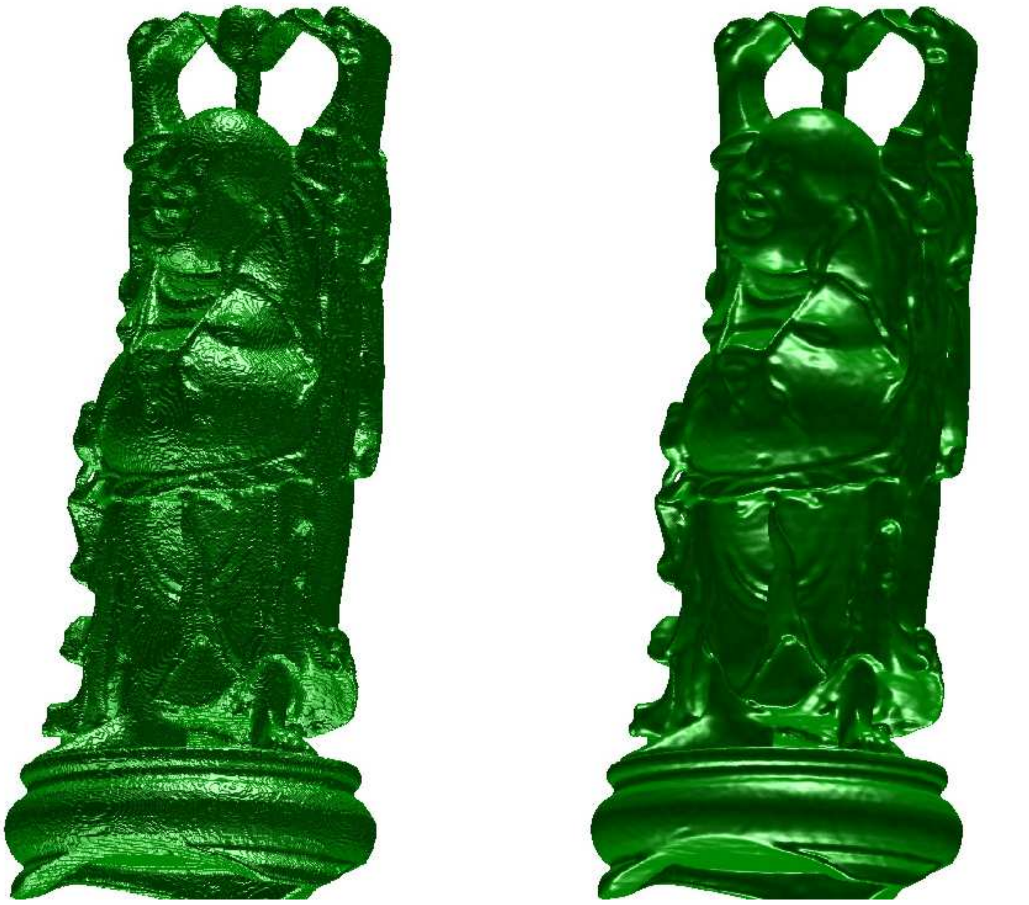

(a) Initial surface (b) Smoothed surface. 10 iterations. $\alpha = 0.7$

Figure 12: Trilinear Bsplines. Bird model



(a) Initial surface (b) Smoothed surface. 15 iterations. $\alpha = 0.5$

Figure 13: Trilinear Bsplines. Buddha model