Slime: A new deformable surface

A. J. Stoddart, A. Hilton, J. Illingworth
Dept. of Electronic and Electrical Engineering
University of Surrey, Guildford, GU2 5XH, UK
email: a.stoddart@ee.surrey.ac.uk

Abstract

Deformable surfaces have many applications in surface reconstruction, tracking and segmentation of range or volumetric data. Many existing deformable surfaces connect control points in a predefined and inflexible way. This means that the surface topology is fixed in advance, and also imposes severe limitations on how a surface can be described. For example a rectangular grid of control points cannot be evenly distributed over a sphere, and singularities occur at the poles.

In this paper we introduce a new (G^1 continuous) deformable surface. In contrast to other methods this method can represent a surface of arbitrary topology, and do so in an efficient way. The method is based on a generalization of biquadratic B-splines, and has a comparable computational cost to methods based on traditional tensor product B-splines.

1 Introduction

Deformable surfaces are a useful tool in surface reconstruction. They provide a convenient way to reconstruct continuous surfaces from 3-dimensional data obtained from range or volumetric images.

In this paper we argue that the *underlying representation* of a deformable surface should ideally have several properties. They are as follows:

(i) The ability to efficiently represent curved surfaces.

(ii) The ability to represent a single open or closed surface of any surface topology (Euler characteristic), e.g. sphere, torus, sphere with two handles, etc.

(iii) The ability to adaptively distribute control points according to surface detail.

(iv) The ability to insert a C^0 discontinuity (step edge) or C^1 discontinuity (roof edge) along any curve embedded in the surface.

(v) Low computational complexity.

(vi) Cut and Paste operations, e.g. can two incomplete surfaces be fused easily?

In addition we will obviously need *algorithms* to make effective use of these properties. Sometimes the two issues are connected: some algorithms only work when particular representations are available.

We believe that all existing methods fail on one or more of the above. The objective of this paper is to introduce a new deformable surface. We exploit a

BMVC 1994 doi:10.5244/C.8.28

sophisticated new representation developed by Loop and De Rose [2, 3] for use in computer graphics. Our contribution is to add the dynamics that make it a deformable surface, and suggest a seeding algorithm.

Deformable surfaces with fixed mesh topology are similar to elastic membranes or flexible plates. They may deform but cannot restructure their topology. The surface in this paper allows for considerable scope to build algorithms in which mesh topology changes. Such a surface has properties analogous to a liquid surface film in addition to the usual elastic and stiffness properties of deformable surfaces. For this reason we call the surface 'slime'. In this paper we demonstrate a basic surface reconstruction algorithm, including a seeding algorithm based on voxel occupancy. Further work remains, but we believe that 'slime' has the potential to satisfy all the above criteria.

1.1 Existing work

It is surprising that one of the most widely used representations of 3-dimensional shape on a computer fails immediately on **(i)**. In many practical applications in CAD and graphics the representation used is a polygonal (usually triangular) mesh. It has the advantage of being easy to render and export to numerically controlled machines. It can also represent an arbitrary topology, so it satisfies points **(ii)-(iv)** and **(vi)**. However in order to represent curved surfaces the polygons must be small and therefore the number of polygons can be very large.

Splines or finite elements typically represent curvature much more efficiently. Two representations in widespread use for geometric modelling, computer aided geometric design and deformable surfaces are tensor product finite elements [9] and tensor product B-splines. B-splines have been widely used in vision as active contour models or snakes. They have several very useful characteristics, as well as being fast and robust. Tensor product approaches are always based on an underlying rectangular grid of finite element nodes or spline control points. The (parametric) surface is usually a mapping from a rectangular domain to $\vec{r}(u, v)$.

An inevitable consequence of this is the limited number of topologies that can be modelled. The programmer must choose the topology *a priori* and is restricted thereafter to only that topology. The user may be forced to distribute control points unevenly over the surface, for example the lines of longitude on a sphere all converge at the poles. It is clear that rectangular grid (connection) topologies fail on **(ii)**, **(iii)**, **(iv)**, **(vi)**. Despite this, their simplicity, combined with the large number of approximately cylindrical objects in our environment, ensures that they will always be useful. Rectangular grids are frequently used for spherical objects, with a disc cut out from each pole.

In a CAD (Computer Aided Design) environment it is common to synthesize complicated objects from a set of tensor product spline patches. The industry standard parametric surface is NURBS (Non-Uniform Rational B-Splines). This includes tensor-product B-splines and the conic sections as a subset. In CAD the topology is supplied by the user during the design process. In vision applications we would prefer not to supply topological information by hand. It is also preferable to avoid the algorithmic complexity of maintaining all the joins, and difficult to ensure that the joins are smooth.

Some authors have attempted to overcome these limitations by moving to a triangulated domain. A triangular tesselation of the domain immediately solves **(iii)** and **(iv)**. It does not automatically solve **(ii)**. In fact, so long as one maintains C^1 continuity one cannot model a surface of arbitrary topology without singularities. For example, when mapping a square domain to a sphere the partial derivatives are singular at the poles.

There is already a considerable Finite Element literature based on triangular elements [8]. It should be noted that there is an obscure but deep-rooted difficulty[1] in maintaining C^1 continuity for triangular elements [7], which leads to the distinction between conformal and non-conformal finite elements. Two attempts have been made to build deformable surfaces based on C^1 continuous triangular elements. McInerney and Terzopolous [11] solve the problem by using conformal finite elements with all partial derivatives up to 2nd order as nodal variables. We have implemented this scheme and find that it is very slow. Also, the mass matrix is non-diagonal and has negative eigenvalues. This means that it it has convergence problems. An alternative solution [10] is to use non-conformal finite elements which have their own drawbacks. Both these methods have the inevitable topology problems associated with maintaining C^1 continuity.

One final method that does comprehensively solve all the topological difficulties is Szelski's oriented particles [4]. The disadvantage is that it does not contain a surface representation. It contains only positions and normals at a finite set of points in space.

1.2 G^1 Continuity

We wish to model free form surfaces made up of small patches or elements which join smoothly. In practice many deformable surfaces enforce C^1 continuity across element boundaries, whereas we argue that it is sufficient to enforce the weaker requirement of G^1 continuity. For a detailed discussion of G^1 continuity see Farin [5] and references therein. In this section we briefly explain what we mean by G^1 continuity.

Consider a curve given in parametric form $\{\vec{r}(u)|u\epsilon D\}$. It is a mapping from a domain $D = [0, 1]$ to a 3-dimensional curve. The tangent vector $\vec{t}$ is given by $d\vec{r}/du$. When a composite curve is formed from two pieces, C^1 continuity is enforced by requiring that the tangent vector be continuous across the join. However, for a given curve, the magnitude of the tangent vector is not invariant under reparameterization, whereas its direction $\hat{t}$ is. G^1 continuity is a requirement that $\hat{t}$ varies continuously across the join. In the case of surfaces, G^1 continuity is a requirement that the tangent plane varies continuously across the join.

To summarize, C^1 continuity relates to the *parametric continuity* of a curve or surface, whereas G^1 continuity relates to *geometric continuity*. We argue that only geometric properties are of importance in computer vision.

[1] This also causes problems for triangle based splines. We recommend careful consideration of this difficulty to anyone using triangles! In our method exactly 4 patches meet at each internal vertex, avoiding the problem.

2 S-Patches

In this section we present a very brief introduction to Bezier curves, Bezier surfaces and a generalization of Bezier surfaces called S-patches. The reader is referred to standard texts [1] for more details on Bezier curves and surfaces, and to Loop and De Rose [2] for more details on S-patches.

When discussing Bezier curves it is useful to replace the usual single parameter u with two parameters u_1 and u_2 and a constraint that $u_1 + u_2 = 1$. This is a notational convenience and makes more explicit the link between Bezier curves and barycentric variables. A depth d Bezier curve $C = \{\vec{r}(u_1, u_2) | u_1 \epsilon [0, 1], u_1 + u_2 = 1\}$ is defined in terms of $d + 1$ control points $\vec{r}_i$ as

$$\vec{r}(u_1, u_2) = \sum_{i=0}^{d} \vec{r}_i B_i^d(u_1, u_2), \qquad u_1 + u_2 = 1 \tag{1}$$

(We use the term depth to avoid confusion with other quantities.) The order d polynomials $B_i^d(u_1, u_2)$ are called the Bernstein-Bezier polynomials and given by

$$B_i^d(u_1, u_2) = \frac{d!}{i!(d-i)!} u_1^i u_2^{d-i} \tag{2}$$

It is easy to verify that the first derivative at $u_0 = 0$ depends only on the first two control points. Bezier curves also have the useful property that the tangent vector at $u_0 = 0$ points in the direction $\vec{r}_1 - \vec{r}_0$.

A depth d tensor product Bezier surface $S = \{\vec{r}(u_1, u_2, v_1, v_2) | (u_1, v_1) \epsilon D, u_1 + u_2 = 1, v_1 + v_2 = 1\}$ may be constructed as

$$\vec{r}(u_1, u_2, v_1, v_2) = \sum_{i,j=0}^{d} \vec{r}_{ij} B_i^d(u_1, u_2) B_j^d(v_1, v_2) \tag{3}$$

Here the mapping is from a (unit) square domain D in (u_1, v_1) space to a 4-cornered surface patch. Each edge is a Bezier curve of depth d.

The Bezier curve maps a one dimensional parameter space (represented by two variables) to 3 scalars, i.e. position. The Bezier curve admits an elegant generalization called a B-form that maps a $[(k+1)$-variate$]$ k-dimensional parameter space onto any number of scalars. The vertices of the domain are called a Bezier simplex, and must be an affinely independent set of points i.e. 2 points in 1D, a triangle or 3 points in 2D, a tetrahedron or 4 points in 3D, and so on.

Firstly we must define multivariate Bernstein-Bezier polynomials. For these we will need a notation for multi-indices $\vec{i} = \{i_1, i_2, ...i_{k+1}\}$. The symbol $\hat{e}_j$ denotes a multi-index whose components are all zero except for the j component which is 1. It is useful to define a modulus of a multi-index as $|\vec{i}| = i_1 + i_2 + ... + i_{k+1}$. The k-variate depth d Bernstein-Bezier polynomials are a simple generalization of equation (2).

$$B_{\vec{i}}^d(u_1, u_2, ...u_{k+1}) = \frac{d!}{i_1! i_2! ... i_{k+1}!} u_1^{i_1}, u_2^{i_2}, ...u_{k+1}^{i_{k+1}}, \qquad |\vec{i}| = d \tag{4}$$

When $k = 2$ we get the familiar Bezier triangle, which maps 3 barycentric variables on a 3-sided domain in 2 dimensions to a surface.

Our interest is parametric surfaces, i.e. mappings from a 2-dimensional domain to a 3-vector position. Above we have presented mappings for 3-sided and 4-sided domains. How do you generalize to a mapping from an n-sided domain? This is a more difficult problem and an elegant solution was recently developed called the S-patch [2]. It *both unifies and generalizes* the Bezier triangle and the tensor product 4-sided Bezier patch.

The trick is to map the coordinates of a domain point $p = (u, v)$ in an n-sided domain D onto a Bezier simplex. Typically the n-sided domain is chosen as a regular n-gon with n vertices p_i, $i = 1..n$. Define the fractional areas $\alpha_i(p)$ as the area of a triangle enclosed by points p, p_i, and p_{i+1} divided by the total area of D. Now form n new variables $\pi_i(p)$ by

$$\pi_i(p) = \alpha_1(p) \times ...\alpha_{i-2}(p)\alpha_{i+1}(p)...\alpha_n(p) \tag{5}$$

Then form normalized variables $l_i(p)$

$$l_i(u, v) = l_i(p) = \frac{\pi_i(p)}{\pi_1(p) + ... + \pi_n(p)} \tag{6}$$

The variables $l_i(p)$ have very special properties. They are invariant under affine transformations. By construction they partition unity, i.e. they are barycentric variables. They equal 1 at the vertex i and are zero at all other vertices. They are non-zero only on edges adjacent to vertex i. In the case of a triangular domain the l_i reduce to exactly the usual area-based barycentric variables.

The S-patch is now simply defined in terms of the variables $l_i(p)$. It is a mapping $S = \{\vec{r}(u, v) | (u, v) \epsilon D_n\}$ where D_n is a n-sided domain polygon and

$$\vec{r}(u, v) = \sum_{|\vec{i}|=d} \vec{r}_{\vec{i}} B^d_{\vec{i}}(l_1, l_2, ...l_n) \tag{7}$$

S-patches, like Bezier triangles, have several very useful properties. They satisfy the de Casteljau recurrence relation [5], which can be used to compute position, and also perform depth elevation [i.e. generating a new set of control points of higher depth that gives the identical surface]. The boundary is a Bezier curve. The tangent plane on the boundary depends only on boundary control points and the adjacent layer of control points. The surface is contained in the convex hull of the control points. The vertex control points are interpolated. The surface is affine invariant.

3 Generalized Biquadratic B-splines

Bezier curves or surfaces are useful in their own right, but in vision problems B-splines are often more convenient. A major advantage of the B-spline is that by construction it joins together a number of Bezier curves or surfaces with C^n continuity.

Loop and De Rose [3] used the S-patch to build a generalization of the bi-quadratic B-spline surface that can describe arbitrary topology and is G^1 continuous everywhere. Due to space limitations it is not possible to include a full description of their surface, and the reader is referred to their papers. However we will give a short overview, starting with a construction of an ordinary biquadratic B-spline surface in terms of 4-sided tensor product Bezier surface patches. For this we need a recipe to obtain the Bezier control points for the component Bezier patches from the B-spline control points.

The B-spline control points are connected as a rectangular mesh. Each face is 4-sided, and each vertex has 4 edges (except at boundaries). Each B-spline control point corresponds to a 4-sided tensor product Bezier patch. There are 9 Bezier control points for the patch, 4 are constructed as the midpoints of edges, 4 as centroids of faces, and the 9th is identical to the B-spline control point. It can be shown [5] that the tangent plane is continuous at all inter-patch boundary points in this construction. In other words the bi-quadratic tensor product B-spline is C^1 continuous.

Loop and De Rose [3] show how to generalize the tensor product quadratic B-spline. In their scheme the control points form a mesh in which each vertex may have any number of edges coming from it, but all faces must have 4 sides. An n edged vertex (spline control point) corresponds to an n-sided S-patch. They give a recipe to compute all the n-sided S-patch control points $\vec{r}_{\vec{i}}$ from the generalized spline control points.

Firstly we make up a Sabin net for each patch. This consists of $2n+1$ points, the central control point $\vec{v} = \vec{q}_j$, the centroids of each adjacent face $\vec{f}_l, l = 1..n$, and the midpoints of each adjacent edge $\vec{p}_l, l = 1..n$. The Sabin net for a 4-gon is identical to the 4 sided Bezier patch control points. The S-patch control points are depth 5, and are given as linear combinations of the Sabin net control points in [3]. This paper also contains a proof that the overall surface is G^1 continuous.

The control mesh has another two less obvious requirements. The edges around each node must be sorted into a cyclic order in order to be passed to the stiffness or rendering routines. In addition some rendering software requires information on the sign of the normal for each surface patch. In practice this is achieved by sorting the neighbour list with respect to an outward normal. This last requirement is impossible for non-orientable surfaces, e.g. a Moebius strip.

4 Deformable surfaces

The generalized biquadratic B-spline is a useful G^1 surface representation. The next step is to construct a deformable surface. The idea of deformable surfaces is that they should find a compromise between smoothness and a fit to data. More precisely they should minimize a cost E which is a weighted sum of the smoothness cost E_s and the data cost E_d, i.e. minimize

$$E = E_s + \lambda E_d \tag{8}$$

The data cost should be the distance squared from a data point to the closest point on the surface. However to simplify the computation it is at present computed as

the distance squared from a data point to the nearest control point. For coarse grids this can lead to significant bias. A frequently used smoothness cost is the thin plate

$$E_s = \int dudv \left\{ \left| \frac{\partial^2 \vec{r}}{\partial u^2} \right|^2 + 2 \left| \frac{\partial^2 \vec{r}}{\partial u \partial v} \right|^2 + \left| \frac{\partial^2 \vec{r}}{\partial v^2} \right|^2 \right\} \tag{9}$$

The smoothness cost may be reduced to a matrix equation in the B-spline control points q_j and the stiffness matrix $K_{jj'}$, i.e.

$$E_s = q^T K q \tag{10}$$

A local minimum of (8) is equivalent to solving the matrix equation $Kq = F$ where $F = -\lambda \partial E_d / \partial q$. Because q is typically a very large vector, and K is sparse it is often useful to find the minimum by introducing a time-dependence $q_j(t)$ and treating the problem as a dynamical problem:

$$\gamma \dot{q} + Kq = F \tag{11}$$

For convenience the 'drag' matrix γ is chosen to be diagonal, and $\dot{q} = \partial q / \partial t$. After some time, if $\dot{q} \rightarrow 0$ then we have found a local minimum of E. This formulation also has the advantage that it can be used for tracking [6].

We have computed this stiffness matrix for 3- and 4-sided patches and computation for the 5- and 6-sided patches is in progress. However in the meantime, to demonstrate the operation of the deformable surface, we have been succesfully using a much simpler stiffness matrix. It is cheaper to compute, but slightly inferior (based on qualitative observations) to the thin plate stiffness. In our experience it has the merit of being very numerically stable.

The alternative stiffness force which replaces Kq is given as follows. Compute the midpoint of all n neighbours of a control point as $\vec{m}$. The force on the control point is then proportional to $\vec{m} - \vec{q_j}$. A force in the opposite direction, multiplied by $1/n$, is applied to each of the neighbours. This force is heuristic, but it should be noted that for the case of a quadratic Bezier curve it is identical to the thin rod force. In practice this 'stiffness' is very fast, simple and robust.

To solve the dynamical equations of motion of the system we use a quality-controlled Euler method.

5 Voxel Seeding

The deformable surface approach only finds local mimima. Most deformable surface algorithms include a heuristic mechanism for seeding the surface with an initial shape that is likely to converge to the desired global minimum. For example the data may be surrounded by a spherical 'balloon' that 'deflates' until it encounters the data. It should be noted that at this point it is almost always the case that the programmer has built in a predefined topology. The surface topology and connection topology have been fixed by hand, perhaps incorporating prior knowledge.

The representation described in this paper offers much potential to develop algorithms in which no predefined topology is needed. In this section we describe

a seeding method that works for complete surface data sets from closed (solid) objects. [By complete we mean no large unsampled areas of surface.]

To be a valid surface the seed must consist of a control mesh that is 4-faced everywhere, and this invites comparison with the surface of a set of voxels. Consider a set of cubes on a regular lattice. Suppose that each cube is either solid or empty. A surface mesh that encloses any set of solid cubes, with a node at each vertex, will consist of 4 sided faces. Such a control mesh is a valid seed.

In figure 1 we show an example of this process. A set of voxels was loaded by hand to generate a chair shape. The seat is a set of 5×5 occupied voxels, with legs of length 4 voxels. The voxel set has been converted to a surface. In order to render the surface we reduce it to a triangular mesh, as this is acceptable to most rendering software. This means that 4-gons have been split into 2 triangles. It is convenient to split 5-gons into 5 triangles, exploiting 5-fold rotational symmetry. The left hand part of figure 1 shows the coarsest rendering of each S-patch, and next to it we refine each triangle into 9 subtriangles. This shows the smooth curvature much better.

We have shown how to seed the surface from voxel occupancy. But where do we obtain the voxel occupancy from? A simple solution is to divide a rectangular volume enclosing the data into a regular grid of cubic voxels. Each voxel is marked as 1 if it contains a data point or 0 if not. A single 26 neighbor dilation followed by an 6 neighbor erosion usually fills any gaps.

In this paper we assume that the data represents a finite set of solid objects. We therefore fill the interior of the voxel set. Clearly we require a closed layer of voxels surrounding our object in order that interior and exterior are well defined. The surface of the voxel set is then used as a seed, which completes our description of the seeding process,

In the next example we fit a scattered data set of 6000 (x,y,z) coordinates taken from a range scan of a foot. The program makes no assumptions about the ordering of the data. In figure 2 we show the initial voxel seeding, and next to it the final result. The voxel size was chosen to be 1/3rd of the smallest edge of the smallest rectangular box fully enclosing the data. The surface converged in about 20 iterations. This took a few seconds on a SUN Sparcstation 2. At present the slowest part of the algorithm is the geometric sorting necessary to establish the nearest control point to each data point. Fast standard algorithms exist to solve this problem, but we have not yet implemented one. For volumetric data geometric sorting is not necessary.

Although the surface topology of the foot is the same as the sphere, this example demonstrates a non-trivial connection topology of the control mesh. The control mesh is well adapted to the foot shape, as can be seen by the even distribution of the triangles.

6 Conclusion and Future Work

We have demonstrated the feasibility of a new deformable surface called slime. It has the potential to satisfy all the criteria specified in the introduction. How do we know that the restricted topology [4-edged faces] of slime is capable of satisfying

(ii)-(iv)? The answer is simple. Clearly a triangular tesselation satisfies **(ii)-(iv)**. However every triangular tesselation can be converted to a slime mesh by the scheme illustrated in figure 3. It therefore follows that a slime mesh always exists to satisfy **(ii)-(iv)**. It should be noted that we still favour a rectangular grid where possible, and two adjacent triangles meshed as shown in figure 3 can be reduced to a 'nicer' mesh by removing two nodes and reconnecting the internal edges as shown in figure 4.

To realize its potential more work should be done. Work is in progress on the thin plate stiffness matrix for the 5-gon and 6-gon. This is not a trivial computation, but has the advantage that it may be done off-line. Once computed the two stiffness matrices will be available to other workers as 11×11 and 13×13 matrices of floating point numbers.

Insertion of crease edges should be addressed as well as open surfaces. The data cost should be enhanced to something that corresponds more closely to the perpindicular distance to the surface, but is still reasonably fast. In general terms there is much scope for building topology control algorithms for slime.

7 Acknowledgements

George Matas is acknowledged for the use of his fast and efficient graph library. Paul Hoad is acknowledged for assistance in preparing diagrams using his package Xmgf. Tim McInerney assisted by supplying the mannequin foot data which was sampled by a Cyberware scanner.

References

[1] M. E. Mortenson, Geometric Modeling, John Wiley & Sons, New York, (1985).

[2] C. T. Loop, T. D. DeRose, A multisided generalization of Bezier surfaces, ACM Trans. on graphics, 8(3) 204-234 (1989).

[3] C. T. Loop, T. D. DeRose, Generalized B-spline surfaces of arbitrary topology, ACM Computer Graphics, 24(4) 347-356 (1990).

[4] R. Szelski, D. Tonneson, D. Terzopolous, Modeling Surfaces of Arbitrary Topology with Dynamic Particles, IEEE CVPR, New York June 1993, 82-87.

[5] G. Farin, Curves and Surfaces for Computer Aided Geometric Design, Academic Press, Boston (1990).

[6] D. Terzopolous, R. Szelski, Tracking with Kalman Snakes, Chapter 1 of Active Vision, ed. A. Blake, A. Yuille, MIT Press, (1993).

[7] O. C. Zienckiewicz, The Finite Element Method, McGraw-Hill, London, 3rd Edition, (1977), see section 10.3, page 231.

[8] G. Dhatt, G. Touzot, The finite element method displayed, John Wiley & Sons, Chichester, (1984).

[9] I. Cohen, L. D. Cohen, N. Ayache , Using deformable surfaces to segment 3-D images and infer differential structure, CVGIP: Image Understanding 56(2) 243-263 (1992).

[10] G. Celniker, D. Gossard, Deformable curve and surface finite-elements for free-form shape design (SIGGRAPH) Computer Graphics 25(4) 257-266 (1991) .

[11] T. McInerney, D. Terzopolous, A Finite Element Model for 3D Shape Reconstruction and Nonrigid Motion tracking, International Conference in Computer Vision, Berlin, May 1993, 518-523.

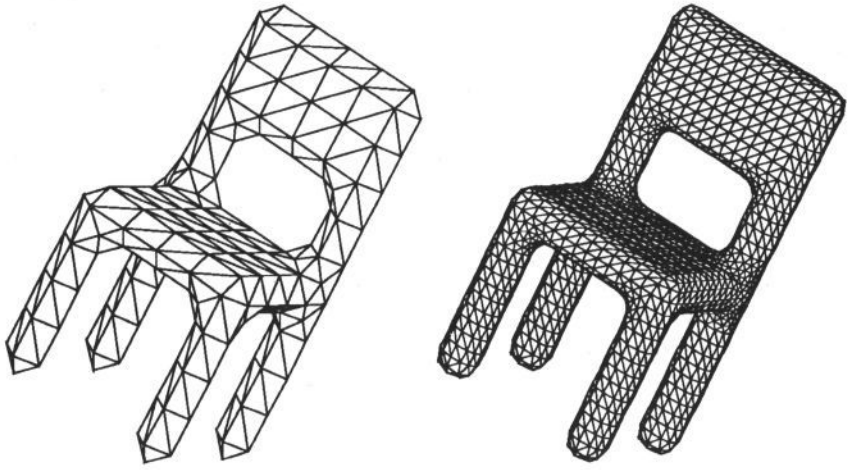

Figure 1: A chair created from a set of voxels

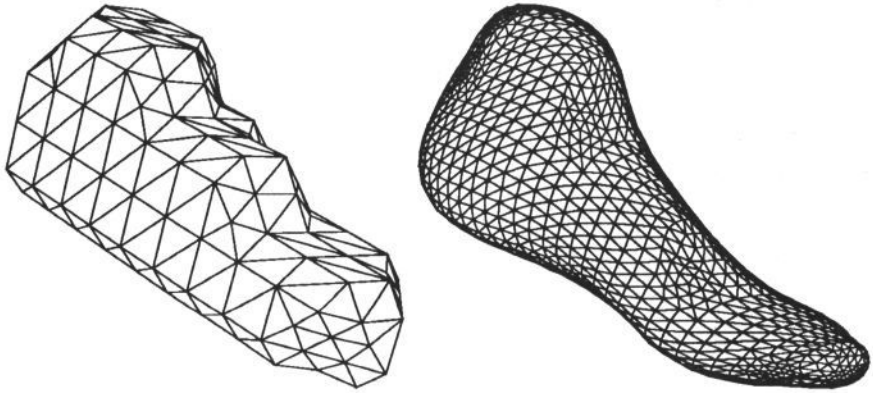

Figure 2: A foot: The voxel based seed, and the final result

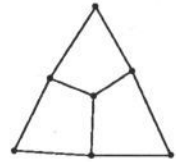

Figure 3: A triangle divided into 4-gons

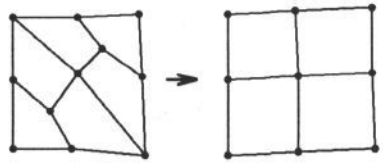

Figure 4: How to convert 2 triangles to a rectangular mesh