Visualization by Subdivision: Two Applications for Future Graphics Platforms

Chand T. John
Computer Science Department
Stanford University

***Abstract*— A fundamental problem in graphics is to have a compact geometric representation of a shape or set of shapes from which a rendering of the shapes can be efficiently computed. In this paper we explore two visualization applications in which geometric relationships between a complex shape and a subdivision of the shape into meaningful parts yields methods for improved visualization, animation, and geometric processing of the shape. Our first application introduces a close relationship between quadratic Bézier curves (QBCs) and iterated function systems (IFSs) that provides a way to compactly represent 2D sets while allowing efficient rendering and morphing of the sets as well. We illustrate the use of QBCs and IFSs in representing, drawing and manipulating 2D clouds which bear a structural and behavioral resemblance to 3D clouds in the real world. Our second application explores four algorithms for segmentation of 3D triangle meshes, so that morphing, compression, classification, and comparison of human teeth (represented as 3D triangle meshes) can be done efficiently. We present the results of applying each segmentation algorithm to teeth and comment on the future of segmentation in graphical data processing.**

I. Introduction

Compact geometric representations of shapes play a central role in compression, manipulation, and rendering of graphical data. One effective method for compactly representing data is to divide the data into parts which can themselves be represented efficiently, and whose interconnections can also be easily described. It is desirable to divide a shape into *meaningful* parts because this simplifies subsequent manipulation and processing of the shape. For example, if we are given a geometric model of the bones and muscles in a human leg, and wish to animate the motion of this leg under certain stresses and strains, it is most useful if the geometry is subdivided into meaningful parts: individual muscles and bones should be represented as separate objects that connect to each other along various tendons and ligaments. A doctor can then make modifications to the geometry which correspond to muscle changes that real surgical procedures would create, and then simulate the motion of this modified leg. It is useful for a doctor to simulate the effect of different forms of surgery on a patient with a musculoskeletal abnormality to visualize such animations in order to decide which type of surgery will have the best effect on the motion of the patient's body. Without a meaningful subdivision of a complex model into parts, the model is of limited value to a user.

Division of a shape into meaningful smaller parts is also useful for geometric processing of the shape or a set of shapes: a human liver can be compared geometrically to a noncancerous one and automatically detected as a geometric anomaly if the cancer is found as a segment separate from the rest of the original liver; the current distribution of temperature in the earth's oceans can be compared to a typical distribution to decide automatically whether a new El Niño is brewing, if regions of significantly varying temperature are segmented separately from each other; or, if we are animating an airplane flying over the west coast of North America, we can initially have the geometric terrain data compressed into meaningful regions, leave most of the data compressed, and uncompress a region's data only when the airplane flies over that region.

Segmenting a large data set into pieces is also useful purely for efficient rendering. Ray tracing and radiosity are two popular methods of rendering scenes, but unfortunately both can be slow in their purest forms. However, if a scene is divided into parts, then discerning which ray hits which part of a scene becomes an order of magnitude faster, making these algorithms practical for visualization. But for this purpose it is less crucial that the segmentations are meaningful, as long as each segment has a reasonable surface area and boundary length.

In this paper we introduce two visualization applications in which the subdivision of a geometric object into meaningful parts is central to having an effective description of the shape. In our first application, described in Section II, we introduce a mathematical relationship between a class of smooth curves known as *quadratic Bézier curves* (QBCs), and *iterated function systems* (IFSs) which can generate complex 2D shapes. The class of shapes constructible by IFSs are known as *self-affine sets*. This relationship allows us to construct complex 2D sets which are represented with no loss of information by a small number of QBCs. Thus this representation facilitates compression, and also animation of a continuously changing 2D shape simply by flexing the curves that represent it. The results can be extended to 3D shapes. Although we have not developed a good algorithm for taking an arbitrary 2D or 3D shape and representing it by a set of curves in this fashion, we are able to create a variety of shapes using these curves. Essentially each curve represents one "part" or "feature" of the overall shape. IFSs alone have been used in the past for generating movies of complex shapes such as clouds [2]. However it is not easy to manipulate an IFS in a way that creates realistic shapes and deformations. Using QBCs, however, not only gives us a greater level of intuitive control over the shape represented by an IFS, but also provides

more control over smooth deformations of the shape than is offered by the IFS's coefficients alone. A more general mathematical relationship between Bézier curves and iterated function systems is given in [10]. We give three examples of the use of QBCs in modeling and deforming 2D cloud-like shapes based on intuitive ideas about how real clouds form in the 3D world.

Visualization, simulation and animation of clouds is needed in two areas: weather prediction, and realistic scene construction for computer games. In meterology, clouds are modeled as the result of thermodynamic interactions in water droplet populations. In computer games and computer-generated movies, it is common to use some physics of light movement but make approximations in such a way that significant computational expense is avoided while a certain level of realism is maintained. Flight simulators are a typical example of games in which realistic cloud rendering is needed [9] [22]. As mentioned before, IFSs have also been used for rendering movies of clouds [2], although this technique is not often used in modern cloud modeling.

In Section II, we introduce an IFS-based cloud modeling technique, but focus more on the shape representation aspect of visualization rather than the rendering. Barnsley [2] illustrates that there are ways to make realistic renderings of objects as long as we can compactly represent the geometry of a complex set. In this paper we demonstrate that we can use QBCs which represent IFS-based clouds, and that the QBCs can be manipulated to also mimick the shape changes that real clouds undergo as they develop.

Real clouds can be classified into three main classes: stratiform, cumuliform, and cirriform. Stratiform clouds are formed by gentle mixing of air masses of different temperatures with minimal vertical movement of air: these clouds are commonly associated with steady drizzles and fog. Cumuliform clouds include the puffy "fair weather" clouds typically seen on partly cloudy days, as well as the large cumulus congestus and cumulonimbus clouds that generate thunderstorms. Cirriform clouds are wispy high-level clouds seen at the top of the troposphere, at the highest altitudes where clouds can form. We will demonstrate that we can control the formation of 2D self-affine clouds that mimick the geometry and formation of stratus clouds, cumulonimbus clouds, and lenticular altocumulus (lens-shaped middle altitude) clouds.

In our second application, in Section III, we describe four segmentation algorithms for geometric shapes represented as 3D triangle meshes, and apply these algorithms to human teeth (represented as 3D triangle meshes). Researchers who study statistics and abnormalities of human tooth shapes are interested in producing such segmentations in order to classify and compare teeth in large dental databases. Those who research dental morphology to study genetic factors in bone structures of people in various populations are also interested in automatic shape analysis of teeth. Visualization of developing and changing teeth is itself useful for those who study human teeth and prescribe stage-by-stage dental procedures, and a meaningful decomposition of the shape of a tooth is useful in constructing simulations and measurements of changes in different areas of the tooth. The central problem is to produce a meaningful segmentation of the geometry of a tooth.

Some 3D shape segmentation algorithms are extensions of 2D *image segmentation* algorithms, which have received decades of attention in computer vision. For example, one of the first recent papers on 3D mesh segmentation [16] extends an image segmentation approach based on the concept of "watersheds." Others use approaches based on deformable models for shapes from medical imaging [1], electrical charge distributions [23], implicit surfaces [3], stereoscopic image pairs [12], cutting along edges of high curvature [8], differential geometry [20], and simple surface patch extraction [17]. Some approaches treat a whole surface as a single segment and repeatedly divide it into smaller segments [11], while others treat each point or face of a surface as a single segment and merge neighboring segments together to form larger segments [7]. Some techniques are results of applying ideas from human vision theory [19] to computer vision. Some apply classical approaches from data clustering [13] while others use new approaches based on the topological characteristics of a shape [4]. We will apply versions of four of the above algorithms to human teeth and assess which of them is the best algorithm both for our application and for general geometric data.

II. Controlling Self-Affine Clouds Using QBCs

First we introduce some basic mathematics of QBCs and IFSs. Then we prove the QBCIFS theorem, which relates QBCs and IFSs. Finally we use the theorem to generate animations of 2D self-affine clouds that bear an overall geometric morphology similar to real-world 3D clouds.

A. Fundamentals of Curves and IFSs

1) Quadratic Bézier Curves: Let $P_0, P_1, \ldots, P_n$ be points in the plane. Let $\alpha_0, \alpha_1, \ldots, \alpha_n \in [0,1]$ such that

$$\alpha_0 + \alpha_1 + \cdots + \alpha_n = 1.$$

Then the *barycenter* of the points $\{P_i\}$ with weights $\{\alpha_i\}$ is the point

$$P = \sum_{i=0}^{n} \alpha_i P_i. \tag{1}$$

Simply put, P is the center of mass ("barycenter") of the points $\{P_i\}$ with weights $\{\alpha_i\}$. The process of computing a barycenter using equation 1 is called a *barycentric combination.*

Suppose we are given three distinct, noncollinear points P_0, P_1 and P_2 in the plane. Suppose we are also given a real number $t \in [0,1]$. The *de Casteljau algorithm* proceeds as follows. First compute two barycentric combinations to obtain the intermediate points:

$$P_0^1(t) = (1-t)P_0 + tP_1 \tag{2}$$

$$P_1^1(t) = (1-t)P_1 + tP_2 \tag{3}$$

Then compute a similar barycentric combination over these intermediate points:

$$P_0^2(t) = (1-t)P_0^1(t) + tP_1^1(t) \quad (4)$$
$$= (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2 \quad (5)$$

If for all $t \in [0,1]$ we plotted the point $P_0^2(t)$, we would obtain the *quadratic Bézier curve* with *control points* P_0, P_1 and P_2. The triangle $\triangle P_0P_1P_2$ is called the *control polygon* of the curve. For more on the history and theory of Bézier curves and surfaces, see [6].

2) Iterated Function Systems: An *affine map* w is a transformation mapping each point (x, y) in the plane to a point $w(x, y)$ in the plane such that

$$w(x,y) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}, \quad (6)$$

where a, b, c, d, e and f are real numbers. We may abbreviate equation 6 with the notation $w(X) = AX+T$, where A is the 2×2 matrix containing the numbers a, b, c and d, $X = (x, y)$, and $T = (e, f)$.

An important fact is that barycentric combinations are invariant with respect to affine maps. That is, if w is an affine map and P is a barycenter defined as in equation 1, then

$$w(P) = w\left(\sum_{i=0}^{n} \alpha_i P_i\right) = \sum_{i=0}^{n} \alpha_i w(P_i), \quad (7)$$

i.e. $w(P)$ is still the barycenter of the points $\{w(P_i)\}$ with weights $\{\alpha_i\}$. The proof of this fact is straightforward–see [6].

A *compact* set is any set which is closed and bounded. For example, the square $B = [0,1] \times [0,1]$ is a compact set in the plane. An *iterated function system* (IFS) is a set of N affine maps $w_1, \ldots, w_N$. In this paper we will only focus on IFSs with $N = 2$ maps, and thus we shall denote an IFS as a pair $\{w_1, w_2\}$ of affine maps. Define a function W that maps compact sets to compact sets, such that $W(K) = w_1(K) \cup w_2(K)$, where K is any nonempty compact set in the plane. It turns out that if the maps w_1 and w_2 satisfy certain properties, then there is a unique set $L = \lim_{n\to\infty} W^{\circ n}(K)$ called the *attractor* of the IFS $\{w_1, w_2\}$. In this paper we shall show that the condition

$$\lim_{n\to\infty} w_1^{\circ n}(X) = X_1 \text{ and } \lim_{n\to\infty} w_2^{\circ n}(X) = X_2 \quad (8)$$

holds, for any point X in the plane, in order to show that the corresponding function W converges to its attractor. We do not prove this point explicitly, but it appears to be true for our case. If this statement is proven, we will have an alternate proof of convergence of the standard subdivision algorithm for Bézier curves. Note that we require that w_1 and w_2 each have exactly one fixed point, X_1 and X_2, respectively. We also require that $W(L) = L$ for the convergence to hold. with *any* nonempty compact set K and end up with the same attractor L. Here $W^{\circ n}(K)$ denotes the repeated application of the map W to the set K a total of n times. See [2] for a thorough treatment of IFSs.

B. IFSs with QBC Attractors

In this section we describe a connection between the two seemingly unrelated mathematical objects introduced above: QBCs and IFSs. In a nutshell, every QBC is the attractor of some IFS. A generalization of this relationship to all polynomial Bézier curves is described in [10]. We now give a mathematical proof of this relationship.

Let S be a QBC with control points P_0, P_1 and P_2. Let $0 < u < 1$ be an arbitrary real number. If we were to apply the de Casteljau algorithm to compute $P_0^2(u)$, in the process we map the original control polygon $\triangle P_0P_1P_2$ into two new polygons $\triangle P_0P_0^1(u)P_0^2(u)$ and $\triangle P_0^2(u)P_1^1(u)P_2$. Since $P_0^2(u)$ lies on the curve, we have also effectively subdivided the original curve S into S_1, the section of the curve where $t \in [0, u]$, and S_2, the section of the curve where $t \in [u, 1]$. Now define two affine maps w_1 and w_2 satisfying

$$\begin{array}{ccc} w_1(P_0) = P_0 & w_1(P_1) = P_0^1(u) & w_1(P_2) = P_0^2(u) \\ w_2(P_0) = P_0^2(u) & w_2(P_1) = P_1^1(u) & w_2(P_2) = P_2, \end{array}$$

so essentially w_1 maps the original control polygon $\triangle P_0P_1P_2$ into the new control polygon $\triangle P_0P_0^1(u)P_0^2(u)$, and w_2 maps the original control polygon into the new control polygon $\triangle P_0^2(u)P_1^1(u)P_2$. It turns out that these two new polygons are actually the control polygons of S_1 and S_2, respectively, and moreover, w_1 maps S to S_1 while w_2 maps S to S_2. Formally:

Lemma 1: $w_1(S) = S_1$ and $w_2(S) = S_2$. $\triangle P_0P_0^1(u)P_0^2(u)$ and $\triangle P_0^2(u)P_1^1(u)P_2$ are the control polygons of S_1 and S_2, respectively.

Proof: First note that

$$\begin{aligned} w_1(P_0^2(t)) &= w_1((1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2) \\ &= (1-t)^2P_0 + 2t(1-t)P_0^1(u) + t^2P_0^2(u) \\ &= (1-ut)^2P_0 + 2ut(1-ut)P_1 + u^2t^2P_2 \\ &= P_0^2(ut) \end{aligned}$$

and

$$\begin{aligned} w_2(P_0^2(t)) &= w_2((1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2) \\ &= (1-t)^2P_0^2(u) + 2t(1-t)P_1^1(u) + t^2P_2 \\ &= P_0^2(u + (1-u)t). \end{aligned}$$

Since S_1 is the section of S where $t \in [0, u]$, then if t varies from 0 to 1, $P_0^2(ut)$ traces out S_1. The above computation shows that $P_0^2(ut)$ is equal to the barycentric combination of P_0, $P_0^1(u)$ and $P_0^2(u)$ with weights $(1-t)^2$, $2t(1-t)$, and t^2 respectively, which by definition of a QBC implies that S_1 is the QBC with control points P_0, $P_0^1(u)$ and $P_0^2(u)$. A similar argument shows that $\triangle P_0^2(u)P_1^1(u)P_2$ is the control polygon of S_2.

The above equations also show that the point with parameter value t on S is mapped to the point with parameter value t on S_1 by w_1, and is mapped to the point with parameter value t on S_2 by w_2, so running over all values of $t \in [0, 1]$, we have shown that $w_1(S) = S_1$ and $w_2(S) = S_2$. ∎

We now show that w_1 and w_2 satisfy the condition in equation 8, which implies that the IFS $\{w_1, w_2\}$ converges

to an attractor. Let us denote $w_1(X) = A_1X + T_1$ and $w_2(X) = A_2X + T_2$. Let I denote the 2×2 identity matrix.

Lemma 2: For all points X in the plane, we have

$$\lim_{n\to\infty} w_1^{\circ n}(X) = P_0 \text{ and } \lim_{n\to\infty} w_2^{\circ n}(X) = P_2. \quad (9)$$

Proof: Since we assume that P_0, P_1 and P_2 are distinct and noncollinear, the vectors $P_1 - P_0$ and $P_2 - P_0$ are linearly independent vectors in the plane, so we can write any point X as

$$X = \alpha_0 P_0 + \alpha_1 P_1 + \alpha_2 P_2 \quad (10)$$

for some real numbers α_0, α_1 and α_2 where $\alpha_0+\alpha_1+\alpha_2 = 1$. Now by definition, $P_0 = w_1(P_0) = A_1P_0 + T_1$, so $T_1 = (I - A_1)P_0$, implying that $w_1(X) = A_1(X - P_0) + P_0$. By induction we have $w_1^{\circ n}(X) = A_1^n(X - P_0) + P_0$ for any n. Since $w_1(P_1) = P_0^1(u) = u(P_1 - P_0) + P_0$, then by induction $w_1^{\circ n}(P_1) = u^n(P_1 - P_0) + P_0$, so we have $\lim_{n\to\infty} w_1^{\circ n}(P_1) = \lim_{n\to\infty}(u^n(P_1-P_0)+P_0) = P_0$. Also, since $w_1(P_2) = P_0^2(u) = P_0+2u(1-u)(P_1-P_0)+u^2(P_2-P_0)$, then by induction $A_1^n(P_2 - P_0) = 2u^n(1 + u + \cdots + u^{n-1})(1-u)(P_1-P_0)+u^{2n}(P_2-P_0)$, so $\lim_{n\to\infty} w_1^{\circ n}(P_2) = \lim_{n\to\infty} A_1^n(P_2 - P_0) + P_0 = P_0$. So, $\lim_{n\to\infty} w_1^{\circ n}(X) = \lim_{n\to\infty} w_1^{\circ n}(\alpha_0P_0+\alpha_1P_1+\alpha_2P_2) = \lim_{n\to\infty}(\alpha_0 w_1^{\circ n}(P_0)+\alpha_1 w_1^{\circ n}(P_1) + \alpha_2 w_1^{\circ n}(P_2)) = (\alpha_0 + \alpha_1 + \alpha_2)P_0 = P_0$, where the second equality follows because $w_1^{\circ n}$ is an affine map and thus preserves barycentric combinations. The argument follows similarly for w_2. ∎

Note the above result also illustrates that w_1 and w_2 each have unique fixed points P_0 and P_2, respectively. For example, we have shown that repeated application of w_1 to any point in the plane converges in the limit to P_0, which would not be the case if w_1 had some other fixed point. The above facts lead to the main result, the QBCIFS theorem.

Theorem 1: (QBCIFS Theorem) Any quadratic Bézier curve with control points P_0, P_1 and P_2 is the attractor of an iterated function system $\{w_1, w_2\}$ where w_1 and w_2 are defined as above.

In short, given any QBC S, we can pick a point on the curve and split S at that point into two pieces S_1 and S_2. Then we define w_1 to be an affine map that transforms S into S_1, and define w_2 to be an affine map that transforms S into S_2. This gives us an IFS $\{w_1, w_2\}$ whose attractor is S itself.

C. Controlling IFS Clouds with QBCs

We have developed a method for finding an IFS whose attractor is a given QBC. Suppose we have several such QBCs $S_1, S_2, \ldots, S_n$. Suppose that we have constructed an IFS $I_i = \{w_{2i-1}, w_{2i}\}$ whose attractor is the curve S_i, for $i = 1, 2, \ldots, n$. Then we can combine all the w_j into one IFS $I = \{w_1, w_2, \ldots, w_{2n}\}$ whose attractor is a dusty cloud-like fractal. We can move control points of the curves S_i in a smooth fashion and recalculate the attractor of the aggregate IFS I to produce an animation that shows the original fractal being deformed smoothly. We would also have chosen arbitrary constraints u_i for each IFS I_i, and these values can also be varied to smoothly deform the attractor of I. If the curves S_i are chosen and deformed appropriately, we can create animations of two-dimensional clouds that form and grow like real clouds. Although the choice of these curves and their deformations is not brought down completely to a science by this technique, we have made a considerable improvement over the existing technique of arbitrarily continuously varying the IFS I itself, because our method offers more intuition over the geometric changes in the attractor.

Three examples will show how the results from earlier can be used to draw and make animations of growing two-dimensional cloud-like structures. To aid in choosing appropriate Bézier curves, we use some basic (not rigorous) knowledge of the actual physics underlying cloud formation, as well as the shapes of the curves themselves. For more on the basic physics of cloud formation, see [5]. For an explanation of the different types of clouds that form, see [14].

Note that the pictures of the 2D clouds in this paper are not beautifully shaded. Although it is not hard to modify the picture generation process to produce more nicely shaded images, here we will present only completely white points of each cloud over a completely black background, so that the direct result of pooling the IFSs of several QBCs is presented. The pictures are generated using the random iteration algorithm [2]. One way to create shaded images would be to plot pixels with a dim gray intensity, and each time a pixel is hit an additional time by the algorithm, increase the intensity of that pixel. Thus the "denser" areas of a 2D cloud will appear brighter, just as the denser parts of a real cloud reflect more light and appear brighter than less dense areas.

Example 1: Stratus. Stratus clouds are formed and exist in environments where overlying warm air in a stable atmosphere mixes benignly with underlying cool air to form relatively flat clouds at the border of the two air layers. The cloud generally forms from top to bottom. We mimick this cloud formation by starting with one QBC with control points $(-10, 0)$, $(0, 0)$ and $(10, 0)$. This curve is a line segment that lies on the x-axis between $x = -10$ and $x = 10$. Now we make three extra copies of this curve, so that we have four copies of the curve in all. We transform the first copy into a new curve with control points $(-10, 0), (0, 1)$ and $(10, 0)$. All that was done in this transformation is that the middle control point, $(0, 0)$, was moved to $(0, 1)$. Next, we transform the second copy of the original curve into a curve with control points $(-10, -3), (0, -4)$, and $(10, -3)$ by vertical shifting of the original control points. We transform the third copy of the original curve into a new curve with control points $(-10, 0), (0, -1.5)$, and $(10, -3)$. Finally, the fourth copy is transformed to a curve with control points $(-10, -3), (0, -1.5)$, and $(10, 0)$.

As shown in the previous section, the IFSs associated with multiple curves can be pooled into a single aggregate IFS whose attractor is an irregular fractal. However, this fractal can be transformed continuously simply by continuously transforming the control points of the Bézier curves that represent this aggregate IFS. In the previous paragraph, four identical

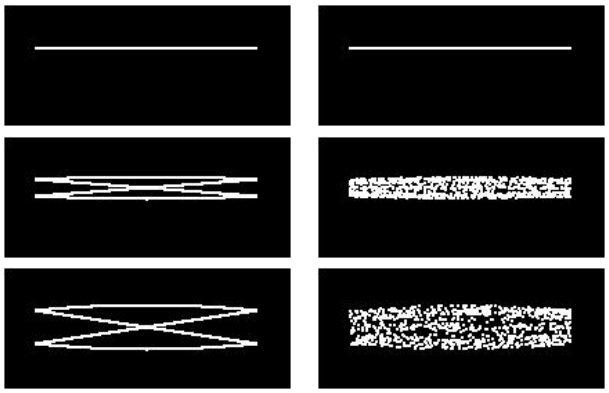

Fig. 1. Graphical illustration of stratus cloud formation by representation of the cloud as the attractor of an IFS that is created by combining the IFSs associated with a set of QBCs. The pictures on the left are the QBCs that were used to generate the corresponding pictures on the right.

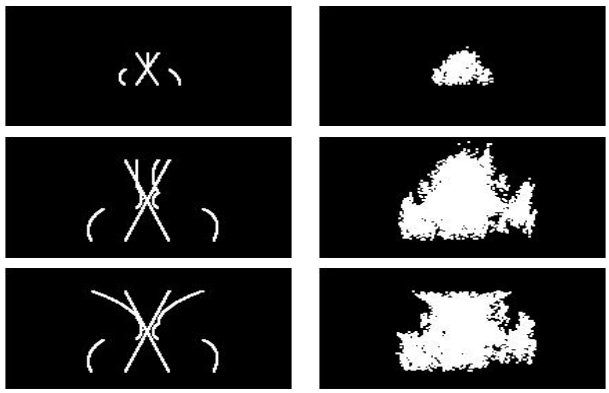

Fig. 2. The growth of a cumulonimbus cloud is illustrated above using QBCs by application of the QBCIFS theorem. The pictures on the left are the QBCs that were used to generate the corresponding pictures on the right. The first cloud is a 2D representation of a fair-weather cumulus cloud. The second picture represents a towering cumulus congestus cloud. Finally, the third picture represents a cumulonimbus cloud, with an anvil shape on top.

curves make up the initial set of curves, whose IFSs are pooled into one IFS whose attractor ends up being the initial curve. But as these curves are transformed into the new curves described above, the pooled IFS associated with these curves has an attractor that is continuously transformed from a line segment (the initial fractal) into a stratiform cloud (the final fractal). So, the continuous transformation of Bézier curves as described above can be used to graphically illustrate the formation of a stratus cloud. The transformations used on the curves have some relationships to the physical mixing of air layers associated with the formation of such a cloud.

Figure 1 illustrates the graphical modelling of stratus cloud formation as described above.

Example 2: Cumulonimbus. Updrafts in moist, unstable air cause small, puffy cumulus clouds to form above the condensation level. If updrafts continue, and sufficient moisture exists, then the small cumulus clouds will become towering cumulus congestus clouds. If updrafts push the cloud top up to the tropopause, then vertical growth is halted and an anvil shape appears at the top of the cumulus cloud, forming a cumulonimbus cloud.

We illustrate such a process by starting with eight identical flat curves (instead of four, as with the stratus example) and continuously transforming them upward to form a cumulus cloud, which then grows further into a cumulus congestus cloud, and finally, two curves in the congestus cloud are elongated horizontally to form an anvil-headed cumulonimbus. See Figure 2.

Example 3: Lenticular cloud. A lenticular cloud is a lens-shaped altocumulus cloud. Lenses are parabolic in shape. So, the relationship between quadratic curves and fractals is quite appropriate for the graphical illustration of a lenticular cloud. The lenticular cloud is modelled by two curves that are nearly identical. See Figure 3.

In all space-filling clouds (Examples 1 and 2), two space-filling curves that formed an X-shape were used. Around these space-filling curves, there were other curves that defined the external shapes of the clouds. Together, the space-filling and external curves generated the two-dimensional fractal clouds that they were meant to represent.

Fig. 3. A lenticular cloud, or lens-shapes altocumulus cloud, is modeled above by two nearly identical QBCs. These QBCs are shown in the first picture. The resulting fractal lenticular cloud is shown in the second picture.

III. Tooth Shape Segmentation

We apply four segmentation algorithms to human teeth, represented as 3D triangle meshes. We then discuss which of these algorithms is the best for our application and in general. Finally we speculate on the future of mesh segmentation algorithms, in particular for graphics and medical data processing applications. Throughout this section, we shall use the following pairs of terms interchangeably: (a) faces and triangles, and (b) segments and clusters.

A. Bottom-Up Clustering

Garland et al [7] describe a bottom-up algorithm for segmenting polygonal meshes in order to speed up ray tracing, collision detection, and radiosity computations. Here we describe and apply a simple variation of their algorithm. The input to the algorithm is a *closed triangle mesh*. A closed mesh is one which is topologically equivalent to a sphere; every edge is part of exactly two triangles. The steps of the algorithm are as follows.

1) Number the triangles in the mesh from 1 to N. This ordering usually already exists in the mesh data structure. Let M denote the number of *pairs* of *adjacent* (sharing an edge) triangles.
2) Initially, each triangle is in its own segment. We can store this information in an array S of N integers, where the value in the ith slot of the array is the number of the

segment containing triangle i. Initially, triangle 1 is in segment 1, triangle 2 is in segment 2, etc., so $S[i] = i$ for each $i = 1, \ldots, N$.

3) Create a matrix A with M rows and 2 columns. The first column should be filled with pairs (i, j) of triangle indices. The second column should contain a "score" $s(i, j)$ that we assign to each pair (i, j) of adjacent triangles. We can choose any score function s that we wish, as long as the score measures the "flatness" of each pair of triangles. Garland et al use a quadric metric to compute scores. We used a simpler measure. First, for every triangle in the mesh, we compute its oriented outward normal vector. Then to any pair (i, j) of adjacent triangles with normal vectors $\mathbf{n_i}$ and $\mathbf{n_j}$, we assigned the score $1 - \mathbf{n_i} \cdot \mathbf{n_j}$.
4) Sort the rows of the above matrix A in decreasing order of scores.
5) Choose a *threshold score* s_{min} that lies in the range of scores contained in A.
6) Repeat for each pair (i, j) of adjacent triangles, in order of the sorted score list: "merge" the clusters containing the two triangles. That is, look at the segment numbers $S[i]$ and $S[j]$ of i and j. If $S[i] = S[j]$, then i and j are already in the same segment, so we can skip this pair and go on to the next pair in the matrix A. If $S[i] \neq S[j]$, then we want to take all of the triangles in segment $S[j]$ (j's segment) and put them into the segment $S[i]$ (i's segment). This is easy to do: simply scan through S and replace all occurrences of $S[j]$ with $S[i]$.
7) Stop merging when the next score in the list is lower than s_{min}. The result is a segmentation of the original mesh.

Results: See Figure 4. The segments tend to be small, and fail to capture the main bumps on a tooth surface. This is because small local fluctuations in the bumpiness of the mesh cause the growth of a segment to suddenly halt before it reaches a reasonable size.

Note: each pair of adjacent triangles corresponds to a unique edge of the mesh. Thus, if for each triangle from 1 through N, we count its three edges, then at the end we will have counted $3N$ edges total. But since each edge belongs to exactly two triangles, we will have counted every edge twice. Thus the number of edges, or number of pairs of adjacent triangles, is $M = 3N/2$. Thus the array A we created in the above algorithm is of linear size. Note also that M must be an integer, implying that N must be even: it is impossible to make a closed triangle mesh with an odd number of triangles. The simplest closed triangle mesh is a tetrahedron, which is made up of four triangles.

B. Top-Down Clustering

Instead of merging small clusters to form larger ones, we can take the opposite route and treat the whole mesh as one huge cluster, and then divide it up into smaller clusters. Katz and Tal [11] use such an approach in order to speed up animations of segmentable objects. We use a variation of their algorithm, which has the following steps. The input is not just a triangle mesh, but also a positive integer k that represents the number of segments into which the mesh will be decomposed.

1) Choose a face with the minimum sum of distances to all other faces in the mesh. Let this be the first $representative face$, f_1.
2) For $i = 2, 3, \ldots, k$ choose the ith representative face f_i to be the face with the maximum possible minimum distance from all previously chosen representative faces $f_1, \ldots, f_{i-1}$.
3) For each nonrepresentative face f in the mesh, compute its distances $d_1, d_2, \ldots, d_k$ to each of the representative faces $f_1, f_2, \ldots, f_k$. Assign f to the segment represented by f_i, a representative face closest to f.

There are many ways to measure the distances in step 3 of the algorithm. We use two methods:

1) *Geometric distance*: The distance between faces f and g is the distance between the centroids of f and g in three-dimensional space.
2) *Geodesic distance*: The distance between faces f and g is the distance between the vertices representing f and g when representing the mesh as a graph whose vertices are the centroids of each face and the edges have weights equal to the distances between the centroids of adjacent faces in three-dimensional space. Geodesic distance takes much longer to compute than geometric distance.

Results: See Figure 5. The segments are much better than the bottom-up approach–the segments conform more to the general hill and valley shapes on a tooth. However, it is still not clear that this segmentation really captures the essential features of a tooth that characterize what type of tooth it is.

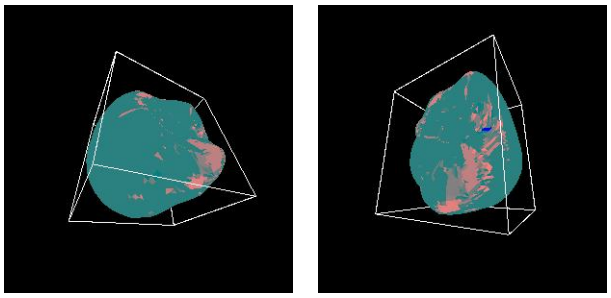

Fig. 4. Decomposition of a tooth using the pairwise merging algorithm based on the bottom-up clustering algorithm of Garland et al. The two pictures show two views of the same tooth. The blue area is one of the segments constructed by the algorithm. Each shade of pink denotes a different segment. Note how the segments tend to be very small and localized, indicating that the local structure of the mesh is a poor guide to its overall structure.

C. Watershed Segmentation

Watersheds were one of the earliest approaches used in 3D mesh segmentation [16]. The basic idea is to view a 3D shape as a piece of land with hills and valleys, similar to the earth. Page et al extend these ideas by combining watersheds and fast marching methods [19]. The algorithm we used is based on their work. The steps in our algorithm are as follows.

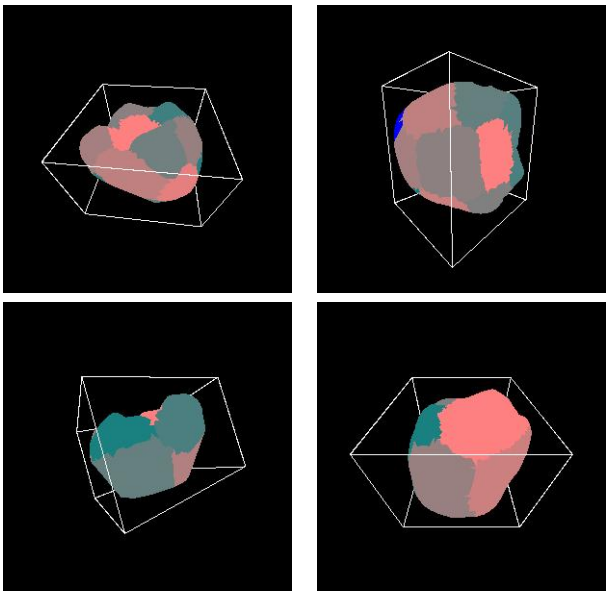

Fig. 5. Top-down decomposition of teeth. The top left image shows the result of dividing a tooth into $k = 20$ segments using geometric distance measurements between faces. The top right image also shows a tooth with $k = 20$ segments but using geodesic distance. The bottom left image shows a tooth divided $k = 10$ segments using geometric distance. The bottom right image shows a tooth divided into $k = 10$ segments using geodesic distance.

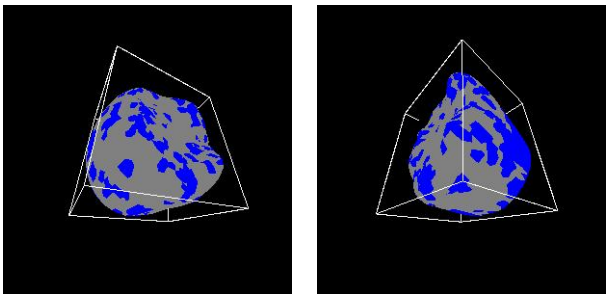

Fig. 6. Initial catchment basins on a tooth using curvature information as in the fast marching watersheds algorithm of Page et al.

1) Compute principal curvatures and directions at each vertex of the mesh.
2) Threshold the regions of positive curvature to get an initial marker set of vertices.
3) Apply mathematical morphology to clean up the marker set.
4) Grow each "catchment basin" in the marker set so that every vertex is assigned to some segment. This yields the final segmentation.

Results: See Figure 6. Essentially the curvature computation [21] yields nonsensical values. This is because our local information is bad, since our data is a reduced and meshed version of the original data, implying that a lot of important local information was thrown out during processing. The catchment basins in Figure 6 yield no useful information about the true hills and valleys on the surface of the tooth.

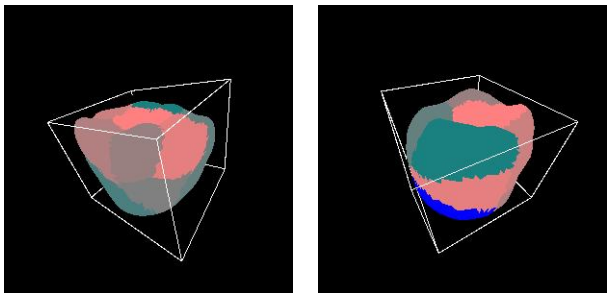

Fig. 7. (Left) Segmentation of a tooth into $k = 10$ segments using Lloyd's algorithm for k-means clustering with $N = 1$ extra iteration. (Right) Segmentation of a tooth into $k = 10$ segments using Lloyd's algorithm with $N = 4$ extra iterations.

D. Lloyd's Algorithm

Lloyd's algorithm is a popular variation of k-means clustering which works as follows. The input is not only a triangle mesh, but also integers $k \geq 2$ and $N \geq 0$.

1) Randomly pick k faces $f_1, \ldots, f_k$ on the mesh.
2) Construct k clusters $C_1, \ldots, C_k$ as follows. Initially, for each $i = 1, 2, \ldots, k$, $C_i = \{f_i\}$. Then for every other face f in the mesh, assign f to the cluster C_j where f_j is the closest of the initial k faces to f.
3) Repeat N times: compute the centroid of each of the k clusters, and then recompute the corresponding clusters, just as before, but using these new cluster centroids.

Results: See Figure 7. The segments are similar in nature to those of the earlier top-down clustering. They still do not give a complete description of the features of a tooth, but the segment shapes do conform to some extent to the main cusps on a tooth.

IV. Conclusion

We have shown that 2D clouds can be depicted using the idea that all quadratic Bézier curves are attractors of IFSs. The formation and change of stratiform and cumuliform clouds can be illustrated by continuously transforming a set of curves in ways that relate to the graphical and physical nature of the formation processes. Because a fractal cloud generated from Bézier curves shows the shapes of the curves in its own shape, this technique of drawing fractal clouds enables us to illustrate the formation of stratus clouds from stable air mixtures, the growth of cumulonimbus clouds from unstable air to small cumulus to cumulus congestus to anvil-headed thunderclouds, and the shapes of lenticular altocumulus clouds.

The techniques of this paper can be extended to three dimensions using an analogous relationship between IFSs and Bézier surfaces. The addition of color may require a six dimensional IFS, where the three spatial coordinates are combined with the three color coordinates (redness, greenness, blueness). Quite possibly, the physics used in choosing curves can be extended to a more rigorous level. The physics that is used in picking Bézier curves or surfaces may become more complicated in higher dimensions, as wind shear and convective cells would

induce a nontrivial amount of lateral motion, and the addition of color would present further complications.

Many improvements remain to be made. However, Section II of this paper does present a new application of a simple idea that relates two different types of geometrical objects. This method for generating images and animations of real scenes combines the advantages of both Bézier curves and iterated function systems. Extensions of these ideas may prove to be beneficial to some areas of visualization.

We also applied four segmentation algorithms to human teeth, represented as 3D triangle meshes. In general, methods that relied on local information to form segments (bottom-up clustering and watershed segmentation) performed poorly for our data, whereas the methods that segment based on the global structure of the shape performed much better (top-down clustering and Lloyd's algorithm). The best algorithm in general for noisy data or data whose local information has been tampered with or removed (such as our data), is Lloyd's algorithm. It is a top-down method but also repeatedly refines its own segmentation until it converges close to a good final segmentation–in spirit, this is just like an author starting with a rough draft of a conference paper and repeatedly proofreading it until it has become a polished final draft. Clearly the final result is better than the first attempt. However, for a *specific* application, people tend to make a special segmentation algorithm that is based on an existing algorithm: for instance, region-growing segmentations in medical imaging originated from the basic idea of bottom-up segmentation. Lloyd's algorithm is more complicated than the first two algorithms we introduced, and so the basic bottom-up and top-down algorithms will continue to coexist with Lloyd's algorithm in the future as a starting point for development of more sophisticated segmentation techniques. Watershed segmentation is another fairly simple and popular segmentation algorithm, which will likely be in use in the future–currently it is still used in medical imaging applications.

For our specific application, new segmentation algorithms that better capture the features of a tooth may need to incorporate information about how real teeth grow and form their shapes. We are working on constructing a segmentation algorithm based on a growth model of the hormonal mechanism by which cusps on teeth grow in stages. This may highlight a more general need for greater use of scientific principles in the application of computer vision and graphics techniques, rather than purely geometric and statistical approaches.

In general, the concept of representing a complex shape by its parts has proven repeatedly to be a useful method of enhancing visualization, manipulation, and geometric processing of data throughout graphics. The techniques we discussed in this paper illustrate some of the potential of this concept to improve the state of the art in computer graphics.

ACKNOWLEDGMENTS

The work on teeth was supported and supervised by Dr. Leonidas Guibas in the Department of Computer Science at Stanford University. The tooth mesh data was provided by Align Technology.

REFERENCES

[1] Z. Bao, L. Zhukov, I. Guskov, J. Wood, D. Breen. Dynamic Deformable Models for 3D MRI Heart Segmentation. *Proceedings of the International Society for Optical Engineering*, 4684: 398-405, 2002.

[2] M. Barnsley. *Fractals Everywhere*, Academic Press: San Diego, CA, 1988.

[3] R. Benlamri, Y. Al-Marzooqi. 3-D Surface Segmentation of Free-Form Objects using Implicit Algebraic Surfaces. *Proceedings of the VIIth Digital Image Computing: Techniques and Applications*, C. Sun, H. Talbot, S. Ourselin, T. Adriaansen (Eds.), 2003.

[4] T. Dey, J. Giesen, S. Goswami. Shape Segmentation and Matching with Flow Discretization. *Proceedings of the Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science 2748, F. Dehne, J.-R. Sack, M. Smid (Eds.), 25-36, 2003.

[5] J. A. Dutton. *Dynamics of Atmospheric Motion*, Dover: Mineola, NY, 1995.

[6] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press: San Diego, CA, 1993.

[7] M. Garland, A. Willmott, P. Heckbert. Hierarchical Face Clustering on Polygonal Surfaces. *ACM Symposium on Interactive 3D Graphics*, 2001.

[8] L. Guillaume, D. Florent, B. Atilla. Constant Curvature Region Decomposition of 3D-Meshes by a Mixed Approach Vertex-Triangle. *Journal of International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 12(2): 245-252, 2004.

[9] M. J. Harris, A. Lastra. Real-Time Cloud Rendering. *Computer Graphics Forum*, Blackwell Publishers, vol. 20, 76-84, 2001.

[10] C. T. John. All Bézier Curves are Attractors of Iterated Function Systems. *New York Journal of Mathematics*, to appear.

[11] S. Katz, A. Tal. Hierarchical Mesh Decomposition Using Fuzzy Clustering and Cuts. *ACM Transactions on Graphics*, 22(3):954-961, 2003.

[12] R. Koch. Surface Segmentation and Modeling of 3-D Polygonal Objects from Stereoscopic Image Pairs. *International Conference on Pattern Recognition (ICPR)*, 233-237, 1996.

[13] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, A. Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7): 881-892, 2002.

[14] D. M. Ludlum. *The Audubon Society Field Guide to North American Weather*, Alfred A. Knopf: New York, 1991.

[15] B. B. Mandelbrot. *The Fractal Geometry of Nature*, W. H. Freeman: New York, 1983.

[16] A. Mangan, R. Whitaker. Partitioning 3D Surface Meshes Using Watershed Segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4): 308-321, 1999.

[17] A. McIvor, D. Penman, P. Waltenberg. Simple Surface Segmentation. *Digital Image Computing - Techniques and Applications/Image and Vision Computing New Zealand (DICTA/IVCZN)*, 141-146, 1997.

[18] J. R. Munkres. *Topology: A First Course*, Prentice-Hall: New Delhi, India, 1987.

[19] D. Page, A. Koschan, M. Abidi. Perception-based 3D Triangle Mesh Segmentation Using Fast Marching Watersheds. *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, Vol. II: 27-32, 2003.

[20] T. Srinark, C. Kambhamettu. A Novel Method for 3D Surface Mesh Segmentation. *Proceedings of the 6th IASTED International Conference on Computers, Graphics, and Imaging*, 212-217, 2003.

[21] G. Taubin. Estimating the Tensor of Curvature of A Surface from A Polyhedral Approximation. *Proceedings of the 5th International Conference on Computer Vision*, 902-908, 1995.

[22] N. Wang. Realistic and Fast Cloud Rendering in Computer Games. Proceedings of the SIGGRAPH 2003 Conference on Sketches and Applications, Session: Simulating Nature, 2003.

[23] K. Wu, M. Levine. 3D Part Segmentation Using Simulated Electrical Charge Distribution. *Proceedings of the 1996 IEEE International Conference on Pattern Recognition (ICPR)*, 1: 14-18, 1996.