Piecewise Linear Approximations of Digitized Space Curves with Applications

Insung Ihm Bruce Naylor

Abstract

Generating piecewise linear approximations of digitized or "densely sampled" curves is an important problem in many areas. Here, we consider how to approximate an arbitrary *digitized 3-D space curve*, made of $n+1$ points, with m line segments. We present an $O(n^3 \log m)$ time, $O(n^2 \log m)$ space, dynamic programming algorithm which finds an optimal approximation. We then introduce an iterative heuristic algorithm, based upon the notions of *curve length* and *spherical image*, which *quickly* computes a *good* approximation of a space curve in $O(N_{iter} n)$ time and $O(n)$ space. We apply this fast heuristic algorithm to display space curve segments and implicit surface patches, and to linearly approximate curved 3D objects, made by rotational sweeping, by *binary space partitioning trees* that are well-balanced.

Keywords : Piecewise linear approximation, Digitized space curves, Computational Geometry, Algebraic curves and surfaces, Binary space partitioning tree

1 INTRODUCTION

The piecewise linear approximation of a digitized or densely sampled curve is an important problem in image processing, pattern recognition, geometric modeling, and computer graphics. Digitized curves occur as boundaries of regions or objects. Such curves, usually represented as sequences of points, may be measured by devices such as scanning digitizers or may be generated by evaluating parametric equations of space curves, or by tracing intersection curves given by implicit surface equations. They can also be obtained from an experiment. For efficient manipulation of digitized curves, they are typically represented in the form of sequences of line segments. While the original curves are made of large sequences of points, their approximations are represented by a small number of line segments that are visually acceptable.

The piecewise linear approximation problem has received much attention, and there exist many approximation algorithms for this problem. Standard line fitting methods such as least squares approximation, Chebycheff approximation and any other nonlinear approximation [Can71, Cd80, Ric64, Mon70] are not well suited for use in a situation where fast and interactive response time is required, since these approximation algorithms perform relatively poorly in terms of computational time and space. On the other hand, the literature in related areas contains many heuristic methods that are more direct and efficient even though, in general, they do not find an optimal approximation [Ram72, PH74, RW74, Wil78, SG80, Pav82, WD84, Rob85, Dun86, FWL89]. This problem was also treated more theoretically in the area of computational geometry. Imai & Iri [II86] presents an $O(n^3)$ time algorithm for finding the best approximation. The time complexity is reduced to $O(n^2 \log n)$ in [MO88, Tou85]. However, most of these works consider only *planar* curves as their input data, and little work has addressed space curve approximation. In many applications, a three dimensional (3D) object is designed with a set of boundary curves in 3D space which are represented as a set of equations or as a sequence of points in 3D space. Hence, having a good approximation method for digitized space curves is essential. In [Kd88] which is one of the few works on 3D space curve approximation, a quintic B-spline is constructed for noisy data, and the length of *Darboux* vector, also known as total curvature, is used as the criterion for segmentation of 3D curves. This method requires construction of quintic B-splines, explicit computation of curvature and torsion, and root solving of polynomials.

In this paper, we consider how to *quickly* produce a *good* piecewise linear approximation of a digitized space curve with a small number of line segments. Our algorithm is based upon the notions of *curve length* and *spherical image*, which are fundamental concepts in differential geometry [Kre59]. In Section 2, we define some notations and give a mathematical formulation of the specific problem we are dealing with. This approximation problem is naturally reduced to a combinatorial minimax problem which can be restated as "Given some number of points, choose a smaller number of points such that the maximum error of approximation is minimized". In Section 3, an optimal approximation is found in $O(n^3 \log m)$-time and $O(n^2 \log m)$-space. We describe, in Section 4, a fast heuristic iterative algorithm which requires $O(N_{iter} n)$-time and $O(n)$-space, where N_{iter} is a number of iterations carried out. Also, the performances of the heuristic algorithm for some test cases are analyzed. In Section 5, we illustrate applications of this fast heuristic algorithm in which space curves and implicit surfaces are adaptively linearized. In Section 6, we also apply the heuristic approximation algorithm to construct adaptive *binary space partitioning trees* for a class of objects made by revolution. It is shown that the linear approximation of a curve can be naturally extended to linearly approximate some class of curved 3D objects in bsp trees that are well-balanced.

2 PRELIMINARIES

We first define a digitized space curve.

Definition 2.1 *Let C be a space curve in three dimensional space. A space curve segment $C(a, b)$ is a connected portion of a curve C with end points a, $b \in \Re^3$.*

In order to define a curve segment without ambiguity, a tangent vector at a might be needed. But we assume this vector is implicitly given.

Definition 2.2 *A digitized space curve segment $\bar{C}(a, b, n)$ of order n is an ordered sequence $\{a = p_0, p_1, p_2, \cdots, p_n = b\}$ of points $p_i \in \Re^3$, $i = 0, 1, \cdots n$, which approximates $C(a, b)$.*

Approximation of a digitized space curve with a small number of line segments results in an approximation error. The quality of approximation is measured in terms of a given error norm that can be defined in many ways. Some of most commonly used ones are

1. *infinite norm* :
$$L_\infty = max \ \ e_i$$
2. *2-norm* :
$$L_2 = (\sum e_i^2)^{\frac{1}{2}}$$
3. *area norm* : L_{area} = *absolute area between curve segment and approximating line segment.*

In this paper, we use L_∞ as an error norm to measure a goodness of an approximation. Note that our algorithms in the later sections are also compatible with L_2.

Definition 2.3 *A piecewise linear approximation $LA(\bar{C}, a, b, m)$ of order m to $\bar{C}(a, b, n)$ is an increasing sequence $\{0 = q_0, q_1, q_2, \cdots, q_m = n\}$ of indices to points in $\bar{C}$. An error $E(LA(\bar{C}, a, b, m))$ of a piecewise linear approximation LA is defined as $\max_{0 \leq i \leq m-1} E_{seg}(i)$ where the i-th segment error $E_{seg}(i)$ is $\max_{q_i \leq j \leq q_{i+1}} dist(p_j, line(p_{q_i}, p_{q_{i+1}}))$, and $dist(x, line(y, z))$ is the Euclidean distance from a point x to a line, determined by two points y and z.*

(Note that, for any point $x \in \Re^3$, and two other points $y, z \in \Re^3$, $(y \neq z)$, $dist(x, line(y, z))$ can be compactly expressed as $\| \ y - x + \frac{(x-y, z-y)}{\|z-y\|}(z-y) \ \|_2$ where $(\cdot, \cdot)$ is a dot product of two vectors and $\| \cdot \|$ is a length of a vector.)

As pointed out in Pavlidis and Horowitz [PH74], the problem of finding a piecewise linear approximation LA can be expressed in two ways :

1. find a $LA(\bar{C}, a, b, m)$ such that $E(LA) < \epsilon$ for a given bound ϵ and m is minimized.
2. find a $LA(\bar{C}, a, b, m)$ that minimizes $E(LA)$ for a given m.

In this paper, we focus mainly on the second type of problem. However, we will also discuss briefly the first type of problem in Section 4.3.5.

Definition 2.4 *Given $\bar{C}(a, b, n)$ and an integer m $(n \geq m)$, the optimal piecewise linear approximation $LA^*(\bar{C}, a, b, m)$ of order m is a piecewise linear approximation such that $E(LA^*) \leq E(LA)$ for any piecewise linear approximation LA of order m. (Note LA^* is not unique.)*

Given these definitions, the problem can be stated as :

Problem 1 *Given $\bar{C}(a, b, n)$ and m, find $LA^*(\bar{C}, a, b, m)$.*

3 AN OPTIMAL SOLUTION

3.1 An Algorithm

A naive algorithm would be as following :

Algorithm 3.1 *(NAIVE)*

> $temp = \infty$;
> for *all the possible* $\binom{n-1}{m-1}$ $LA(\bar{C}, a, b, m)$ do
>   *compute* $E(LA)$;
>   if $E(LA) < temp$ then $LA^* = LA$; $temp = E(LA)$;
> endfor

Note that the problem has a recursive nature, that is, it can be naturally divided into two subproblems of the same type. Dynamic programming, which is a general problem-solving technique widely used in many disciplines [AHU74], can be applied in this case to produce a rather straightforward algorithm. We first give an algorithm which works in case m is a power of 2. Then the algorithm is slightly modified for an arbitrary m.

Define E_{ij}^l to be the error of $LA^*(\bar{C}, p_i, p_j, l)$, that is, the smallest error of all piecewise linear approximations with l segments to the portion of $\bar{C}$ from p_i to p_j. Then E_{ij}^l can be expressed in terms of $E_{ik}^{\frac{l}{2}}$ and $E_{kj}^{\frac{l}{2}}$ as following :

$$E_{ij}^{2^d} = \min_{i<k<j} \max\{E_{ik}^{2^{d-1}}, E_{kj}^{2^{d-1}}\} \quad for \;\; 0 \le i < j \le n \;\; and \;\; d > 0. \tag{1}$$

(Note that $E_{ij}^l = 0$ if $j - i \le l$.)

The recursive relation renders the following dynamic programming algorithm which computes the minimum error E_{0n}^m and its corresponding LA^* :

Algorithm 3.2 *(DYNAMIC)*

> /* basis step */
> for $i = 0$ to $n-1$ do
>   for $j = i+1$ to n do
>   *compute* E_{ij}^1;
>   endfor
> endfor
> /* inductive step */
> for $d = 1$ to $\log m$ do
>   for $i = 0$ to $n - 2^d - 1$ do
>   for $j = i + 2^d + 1$ to n do
> $E_{ij}^{2^d} = \max\{E_{ik'}^{2^{d-1}}, E_{k'j}^{2^{d-1}}\} = \min_{i<k<j} \max\{E_{ik}^{2^{d-1}}, E_{kj}^{2^{d-1}}\}$;

```
        K^d_ij = k';
      endfor
    endfor
  endfor
  construct LA* from K^d_ij;
```

In the basis step, E^1_{ij} is computed by calculating the distances from the points p_k, $i < k < j$ to the line passing through p_i and p_j, and taking their maximum. K^d_{ij} is needed to recursively construct the optimal piecewise linear approximation once E^m_{0n} is computed. Note the recursive relation $LA^*(\bar{C}, p_i, p_j, 2^d) = LA^*(\bar{C}, p_i, p_{K^d_{ij}}, 2^{d-1}) \cup LA^*(\bar{C}, p_{K^d_{ij}}, p_j, 2^{d-1})$.

3.2 The Time and Space Complexities

Since E^1_{ij} is computed in $O(j-i)$ time, the basis step requires $O(\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} (j-i)) = O(n^3)$ time. Similarly, $E^{2^d}_{ij}$ can be computed in $O(j-i)$ time. So, the inductive step needs $O(n^3 \log m)$ time. Also, construction of LA^* can be done in $O(m)$ time. These three time bounds are combined into $O(n^3 \log m)$.

With regard to space, the algorithm needs $O(n^2)$ space for storing a table for $E^{2^d}_{ij}$. Also, $O(n^2 \log m)$ space is required to save K^d_{ij}, $d = 1, 2, \cdots, \log m$. Hence, the space complexity is $O(n^2 \log m)$.

3.3 An Algorithm for an Arbitrary m

When m is not a power of 2, we can break m into m' and $m - m'$, where m' is the largest power of 2 less than m. $m - m'$ is then broken if it is not a power of 2. Applying this process repeatedly produces two sequences of numbers, one made of powers of 2, and the other made of non-powers of 2. By maintaining two tables, and synchronizing the order of merging operations, E^m_{0n} can be computed. It is not difficult to see that this modification only increases both time and space complexities by constant factors.

4 A HEURISTIC SOLUTION

Even though the algorithm *DYNAMIC* finds an optimal approximation, the time and space requirement is excessive. As stated in Section 4.3.4, the algorithm is extremely slow even for modest n, for example, $n = 400$. In fact, it is more desirable to generate quickly a good approximation. In this section, we describe a heuristic algorithm which consists of two parts, *computation of an initial approximation* and *iterative refinement of the approximation.* Our heuristic algorithm is based upon the observation that the error of a segment is a function of the length of the curve segment, and the total absolute change of the angles of tangent vectors along the curve segment. It tends that the longer curve segment has the larger segment error. Also, the total angle change is a measure of how much a curve segment is bent. However, it is illustrated in the next two subsections that neither measure alone is a good heuristic. Our heuristic in Section 4.3 is a weighted sum of the two measures, and this simple combined measure yields a good initial guess.

4.1 Curve Length Subdivision

Assume we have a parametric representation $C(t)$ of a curve C. The first heuristic is to divide a curve segment into subsegments with the same curve length where the *curve length* is defined to be $\int_a^b \| \frac{dC(t)}{dt} \| \; dt$. This quantity is usually approximated by the *chord length* as following.

Given a digitized curve $\bar{C}(a,b,n) = \{a = p_0, p_1, \cdots, p_n = b\}$, consider a parametric curve $C(t)$ of a parameter t where $C(0) = p_0$ and $C(l) = p_n$. Then,

$$\begin{aligned}
\int_0^l \| \frac{dC(t)}{dt} \| \; dt &\approx \sum_{i=0}^{n-1} \| \frac{p_{i+1} - p_i}{d(p_i, p_{i+1})} \| \, d(p_i, p_{i+1}) \\
&= \sum_{i=0}^{n-1} \| p_{i+1} - p_i \| \\
&= \sum_{i=0}^{n-1} d(p_i, p_{i+1})
\end{aligned}$$

where $d(p,q)$ is the Euclidean distance between two points p and q in $\Re^3$.

Algorithm 4.1 *(LENGTH)*

/* let $L_{seg}(i,j)$ be $\sum_{k=i}^{j-1} d(p_k, p_{k+1})$ */
compute total $= \sum_{k=0}^{n-1} d(p_k, p_{k+1})$;
$seglength = ceil(total/m)$;
$q_0 = 0$; $i = 0$;
while $i < m - 1$ do
    find the largest j *such that* $L_{seg}(q_i, j) < seglength$;
    $q_{i+1} = j$; $i = i + 1$;
endwhile
$q_m = n$;

Figure 1(upper left) and Figure 2 (leftmost) indicate that this algorithm produces a *LA* which approximates $\bar{C}$ quite well in flat regions of a curve, and poorly in highly curved regions.

4.2 Spherical Image Subdivision

Consider a curve $C(s)$ with an *arc length* parameter s [Kre59, O'N66]. When all unit tangent vectors $T(s)$ of $C(s)$ are moved to the origin, their end points will describe a curve on the

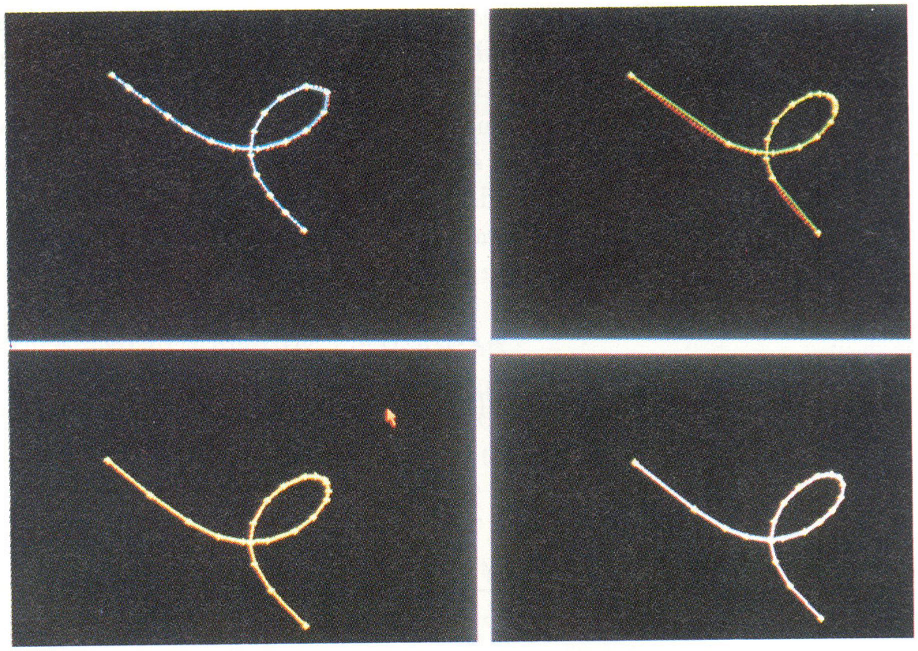

Figure 1: Folium of Descartes

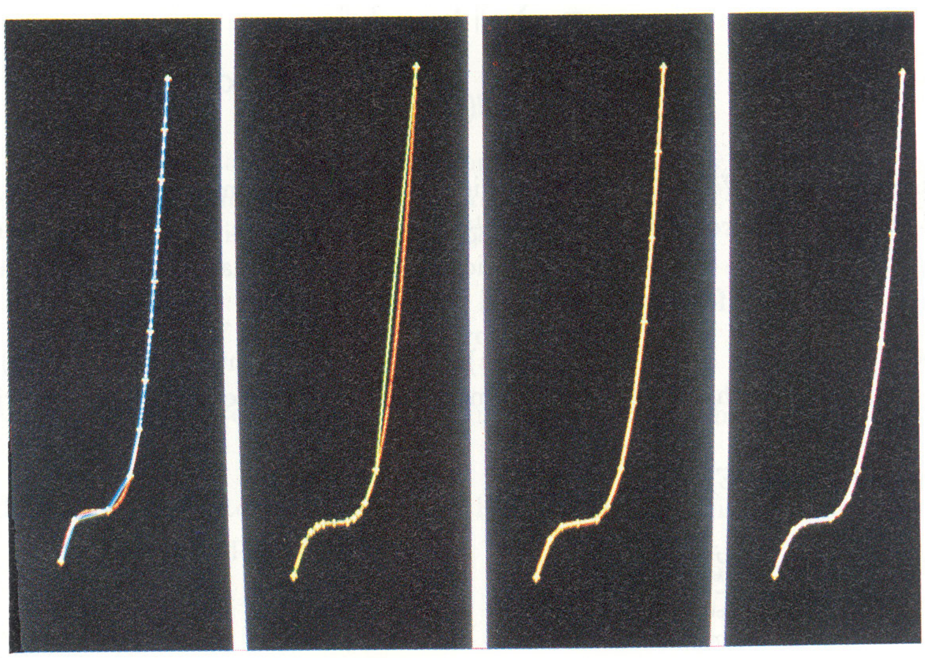

Figure 2: A cubic curve

unit sphere. This curve is called *the spherical image* or *spherical indicatrix* of $C(s)$. Given a curve segment, the length of the corresponding spherical image implies how much the unit tangent vector changes its direction along the curve segment. Hence, it gives us a measure of the degree to which a curve segment is curved. It is easily shown that the curvature $\kappa(s)$ is equal to the ratio of the arc length of the spherical image, and the arc length of $C(s)$. So, the length of the spherical image corresponding to a curve segment $C(s)\ :[0,l]$ is $\int_0^l \kappa(s)\,ds$. ($\int_0^l \kappa(s)\,ds$ is sometimes called *the total curvature* [O'N66], while it also can mean the length of Darboux vector [Kre59].) Practically, the quantity must be approximated.

Given a digitized curve $\bar{C}(a,b,n) = \{a = p_0, p_1, \cdots, p_n = b\}$, consider an imaginary parametric curve $C(s)$ of an arc length parameter s where $C(0) = p_0$ and $C(l) = p_n$. At a point p_i, $s \approx cl(p_0, p_i)$ such that $C(s) = p_i$, where $cl(p_0,p_i) = \sum_{j=0}^{i-1} d(p_j, p_{j+1})$. Then, the curvature is approximated as following :

$$
\begin{aligned}
\kappa(s) &= \| \lim_{\delta s \to 0} \frac{T(s+\delta s) - T(s)}{\delta s} \| \\
&\approx \| \frac{t_{i+1} - t_i}{d(p_i, p_{i+1})} \| \qquad (1)
\end{aligned}
$$

where t_i is an approximated unit tangent vector. (We will discuss how to get t_i shortly.) Then,

$$
\begin{aligned}
\int_0^l \kappa(s)\,ds &\approx \sum_{i=0}^{n-1} \| \frac{t_{i+1} - t_i}{d(p_i, p_{i+1})} \| \, d(p_i, p_{i+1}) \\
&= \sum_{i=0}^{n-1} \| t_{i+1} - t_i \| \\
&= \sum_{i=0}^{n-1} d(t_i, t_{i+1}).
\end{aligned}
$$

The simple *forward-difference* approximation (1) to $\kappa(s)$ can be replaced by the popular *central-difference* approximation $\frac{d(t_{i-1}, t_{i+1})}{d(p_{i-1}, p_i) + d(p_i, p_{i+1})}$ which is a much better approximation when the points are close together. Integration can be also replaced by a better approximation formula. See [Cd80] for more numerical techniques.

In this second heuristic method, $\bar{C}(a,b,n)$ is subdivided into $LA(\bar{C},a,b,m) = \{0 = q_0, q_1, q_2, \cdots, q_m = n\}$ such that each subsegment has the same length of the spherical image.

Algorithm 4.2 *(IMAGE)*

> /* let $I_{seg}(i,j)$ be $\sum_{k=i}^{j-1} d(t_k, t_{k+1})$ */
> *compute total* $= \sum_{k=0}^{n-1} d(t_k, t_{k+1})$;
> *segind* $=$ *ceil*(*total*/m);
> $q_0 = 0$; $i = 0$;
> while $i < m - 1$ do
>   *find the largest* j *such that* $I_{seg}(q_i, j) <$ *segind*;
> $q_{i+1} = j$; $i = i + 1$;
> endwhile
> $q_m = n$;

The quantity $I_{seg}(q_i, q_j)$ is an approximating measure of the length of the spherical image of the segment from p_{q_i} to $p_{q_{i+1}}$, that is, $I_{seg}(q_i, q_j)$ is a total absolute change of the angles of tangent vectors. Hence, this algorithm is sensitive to high curvature. We can see $IMAGE$ returns a LA which approximates $\bar{C}$ poorly in flat portions of a curve, and very well in highly curved portions in Figure 1 (upper right) and Figure 2 (the second from left).

In the above algorithm, tangent vector information is used to subdivide a curve. If the digitized space curve has been generated from equations, say a parametric equation or two implicit equations, the tangent vector at each sample point can be computed directly from them. When instead a digitized curve has been given in terms of a sequence of points, or direct computation of tangent vectors from given equations is expensive, the tangent vector t_k to a curve C at p_k still can be approximated by averaging the directions of the neighboring lines of p_k in $\bar{C}$. In our implementation, the tangent vector is approximated by 5 successive points as follows [Pie87] :

$$t_k = \left(1 - \frac{\alpha}{\alpha+\beta}\right) \cdot v_i + \frac{\alpha}{\alpha+\beta} \cdot v_{i+1},$$

where $\alpha = \| v_{i-1} \times v_i \|, \beta = \| v_{i+1} \times v_{i+2} \|$, $v_i = p_i - p_{i-1}$, and $\times$ means a cross product of two vectors. In case the digitized curve is open, the Bessel conditions are applied for the tangents at the end points as follows [deB78] :

$$\begin{aligned} v_0 &= 2v_1 - v_2, & v_{-1} &= 2v_0 - v_1, \\ v_{n+1} &= 2v_n - v_{n-1}, & v_{n+2} &= 2v_{n+1} - v_n. \end{aligned}$$

4.3 Heuristic Subdivision

Now, we give a heuristic algorithm which combines the two techniques. It consists of two steps : generation of an initial piecewise linear approximation LA_0, and iterative refinement of piecewise linear approximation LA_k to produce LA_{k+1}.

4.3.1 Computation of An Initial Approximation : LA_0

An initial LA_0 is computed by an algorithm which is a combination of $LENGTH$ and $IMAGE$.

The weight, α is a parameter which controls the relative emphasis between curve length and spherical image, and is empirically chosen.

Algorithm 4.3 *(INIT)*

> *select some value of* α $(0 \le \alpha \le 1)$;
> *compute total* $= \sum_{k=0}^{n-1} (\alpha \cdot d(p_k, p_{k+1}) + (1-\alpha) \cdot d(t_k, t_{k+1}))$;
> $segsum = ceil(total/m)$;
> $q_0 = 0$; $i = 0$;
> while $i < m - 1$ do

find the largest j such that $\alpha \cdot L_{seg}(q_i, j) + (1-\alpha) \cdot I_{seg}(q_i, j) < segsum$;
$q_{i+1} = j$; $i = i + 1$;
endwhile
$q_m = n$;

See Figure 1 (bottom left) and Figure 2 (the third from left).

4.3.2 Iterative Refinement of Approximations : LA_k

The "hybrid" algorithm *INIT* generally produces a good piecewise linear approximation. The next step is to *diffuse errors* iteratively in order to refine the initial approximation. Note each segment is made of a sequence of consecutive points of a digitized curve, and it is approximated by a line connecting its end points. Usually, an error of a segment decreases as either of its end points is assigned to its neighboring segment. Hence, the basic idea in the following iterative algorithm is to move one of end points of a segment with larger error to its neighboring segment with less error, expecting decrease of the total error of the new LA. In the kth step of the following algorithm $ITER$, each segment of LA_k is examined , diffusing, if possible, its error to one of its neighbors. LA_k tends to quickly converge to a minimal LA which is a local minimum. See Figure 1 (bottom right), Figure 2 (rightmost), and Figure 3.

Algorithm 4.4 *(ITER)*

compute LA_0 *from INIT*;
$k = 0$;
do until *(satisfied)*
  compute errors of segments in LA_k*;*
  $curmax = E(LA_k(\bar{C}, a, b, m)$;
  for $i = 0$ to $m - 1$ do
    if *the error of i-th segment is larger than*
      that of either of its neighboring segments
    then *move the i-th segment's end points to the neighbor*
      only if this change does not result in segment errors
      larger than curmax;
    endif
  endfor
  $LA_{k+1} = LA_k$;
enddo

4.3.3 The Time and Space Complexities

First, $O(n)$ time is needed in order to approximate the tangent vector at each point. The algorithm *INIT* needs to scan the points and tangent vectors to compute L_{seg} and I_{seg} first,

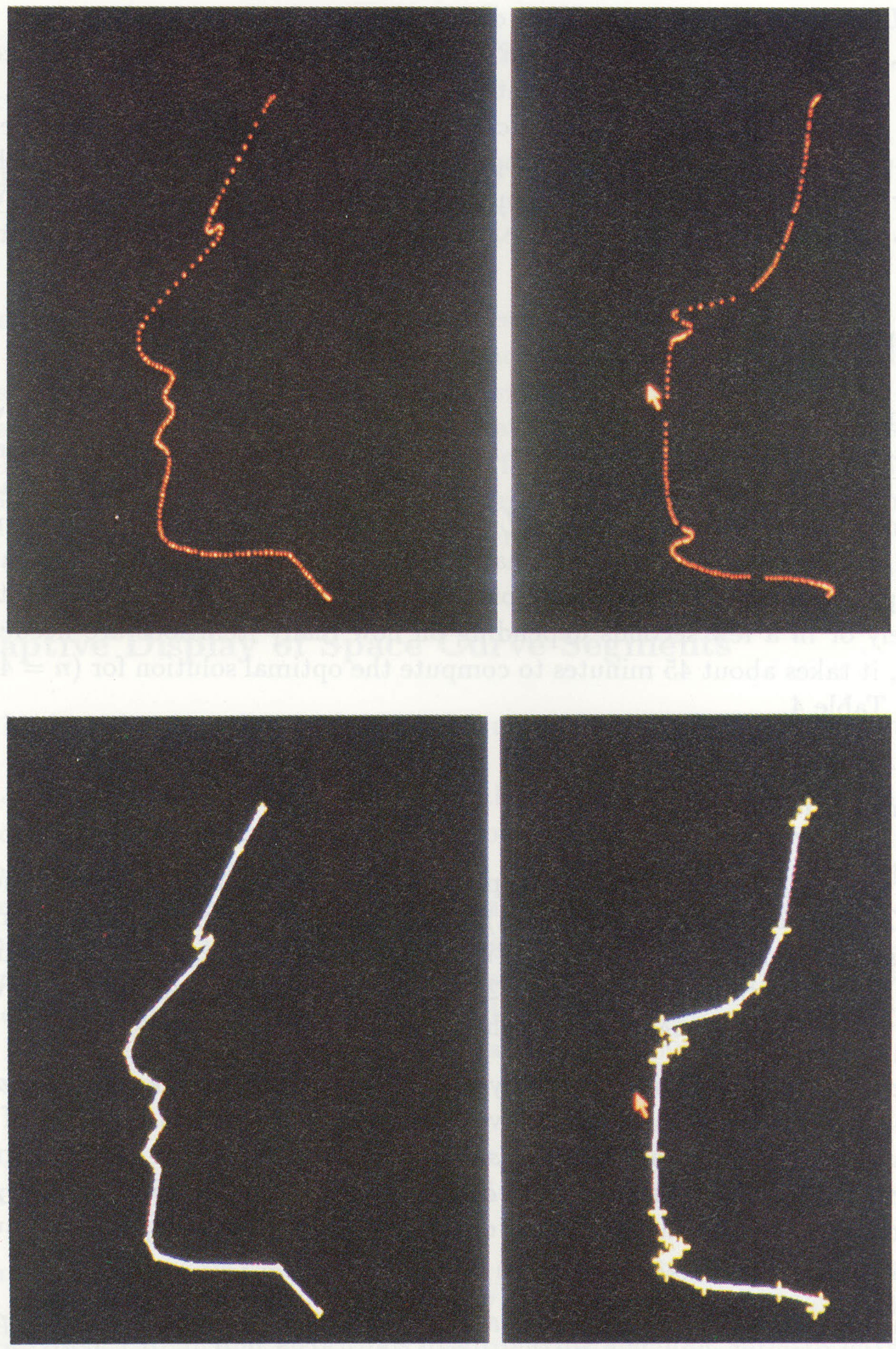

Figure 3: A human profile and a goblet

and then L_{seg} and I_{seg} are scanned to divide the digitized curve. Hence, it takes $O(n)$ time. Now, consider the algorithm *ITER*. First, the segment errors of LA_k is computed in $O(n)$ time. In the *for* loop, each segment and its two neighbors are examined, hence, each segment is examined twice. Since for each segment, the segment error must be computed, $O(n)$ computation is needed by the *for* loop. So, *ITER* takes $O(N_{iter} \cdot n)$ time where N_{iter} is the number of iterations. So, the time complexity of the heuristic algorithm is $O(N_{iter} \cdot n)$, and it is easy to see $O(n)$ space is sufficient for storing input data and intermediate computations.

4.3.4 Performance

We have implemented both the optimal and heuristic algorithms on a Sun 4 workstation and a Personal Iris workstation. Table 1– 5 in Appendix A show their performances for test data. The integer in the parenthesis is the number of iterations needed to arrive at the local minimum. The bottom row (LA_k/LA^*) of each table indicates the performance of our heuristic algorithm, and it is observed that it approximates the optimal solution reasonably well. The program for the heuristic algorithm computes the approximate solution quickly (immediately or in a few seconds depending on how many iterations are needed.) On the other hand, it takes about 45 minutes to compute the optimal solution for ($n = 404$,$m = 64$) example of Table 4.

4.3.5 The Center of Mass

We now briefly consider the following type of the piecewise linear approximation problem : "find a $LA(\bar{C}, a, b, m)$ such that $E(LA) < \epsilon$ for a given bound ϵ and m is minimized." Even though our heuristic algorithm was invented for an arbitrary number of subsegments, we can use it for dividing a segment into 2 subsegments. One simple algorithm would be to recursively divide a curve segment until the error of subsegment is less than ϵ.

If a curve segment is to be divided into only two subsegments, the notion of the *center of mass* can be applied. As before, assume we have a parametric representation $C(s)$ of a curve C, where s is an arc length parameter, and $\kappa(s)$ is its curvature. Consider a curve segment define by an interval $[0, l]$. Then the *center of curvature*, defined by $c_\kappa = \int_0^l s\kappa(s)\, ds / \int_0^l \kappa(s)\, ds$, can be used as a heuristic that divides a curve segment $C(s) : [0, l]$ into two subsegments $C(s) : [0, c_\kappa]$ and $C(s) : [c_\kappa, l]$.

Again, c_κ needs to be approximated. For a digitized curve $\bar{C}(a, b, n) = \{a = p_0, p_1, \cdots, p_n = b\}$, consider an imaginary parametric curve $C(s)$ of an arc length parameter s where $C(0) = p_0$ and $C(l) = p_n$. Then, at a point p_i, $s \approx cl(p_0, p_i)$ such that $C(s) = p_i$, where $cl(p_0, p_i) = \sum_{j=0}^{i-1} d(p_j, p_{j+1})$. Together with the approximation of the denominator given before, the following expression results in an approximation of c_κ :

$$\begin{aligned}
\int_0^l s\kappa(s)\, ds &\approx \sum_{i=0}^{n-1} cl(p_0, p_i) \parallel \frac{t_{i+1} - t_i}{d(p_i, p_{i+1})} \parallel d(p_i, p_{i+1}) \\
&= \sum_{i=0}^{n-1} cl(p_0, p_i) \parallel t_{i+1} - t_i \parallel
\end{aligned}$$

$$= \sum_{i=0}^{n-1} cl(p_0, p_i) d(t_i, t_{i+1}).$$

5 APPLICATION I : DISPLAY OF SPACE CURVES AND SURFACES

Curves and surfaces in geometric modeling are represented in either the parametric or implicit form. Each form has its own advantages and disadvantages. For instance, implicit curves and surfaces naturally define half spaces, and ray-surface intersections are easily computed, while the parametric form is well suited for generating points along a curve or surface. In order for complex objects, made of curves and surfaces, to be effectively manipulated, it is very desirable to have a fast and adaptive display method. In this section, we consider how a good approximation to a curve or surface is generated quickly.

5.1 Adaptive Display of Space Curve Segments

Our heuristic algorithm is well suited to producing a piecewise linear approximation of a space curve segment in the parametric or implicit form. First, the curve segment is densely sampled, and then the linear approximation algorithm filters the sampled points, producing a good approximation to the curve segment. Points on a parametric curve are easily generated. A curve, represented by two implicit surfaces or an implicit surface and a parametric surface, can be traced using a surface intersection algorithm (for example, [BHHL88]). The space curve tracing algorithm is very fast when the degrees of curves are in a reasonable range and there are no singular points along the curve segment. As seen in the examples, only a small number of line segments, adaptively filtered, can approximate a curve segment well, resulting in fast display. Figure 1, 2 and 4 are examples of planar curves, and Figure 5 and 6 are those of nonplanar curves.

5.2 Adaptive Display of Implicit Surface Patches

Algebraic surfaces have become increasingly important as they lend themselves well to some applications in geometric modeling like creation of blends and offsets [MS85, Sed85, HH86, War86, RO87, BI88, PK89], and algorithms for displaying algebraic surfaces have emerged.

Hanrahan [Han83] showed that algebraic surfaces lend themselves well to ray tracing. Sederberg and Zundel [SZ89] uses a scan line display method which offers improvement in speed and correctly displays singularities. Even though both approaches produce very good images, the computation cost is expensive and the process is static in the sense that operations on objects, like rotation and translation, can not be done dynamically. On the other hand, the polygonization-and-shading technique [AG89, Blo88] uses the capability of the graphics hardware which provides very fast rendering. We are currently working on the problem of constructing a complex object made of triangular implicit surface patches, and one difficult problem is how to isolate, numerically, only the necessary part or the triangular patch from

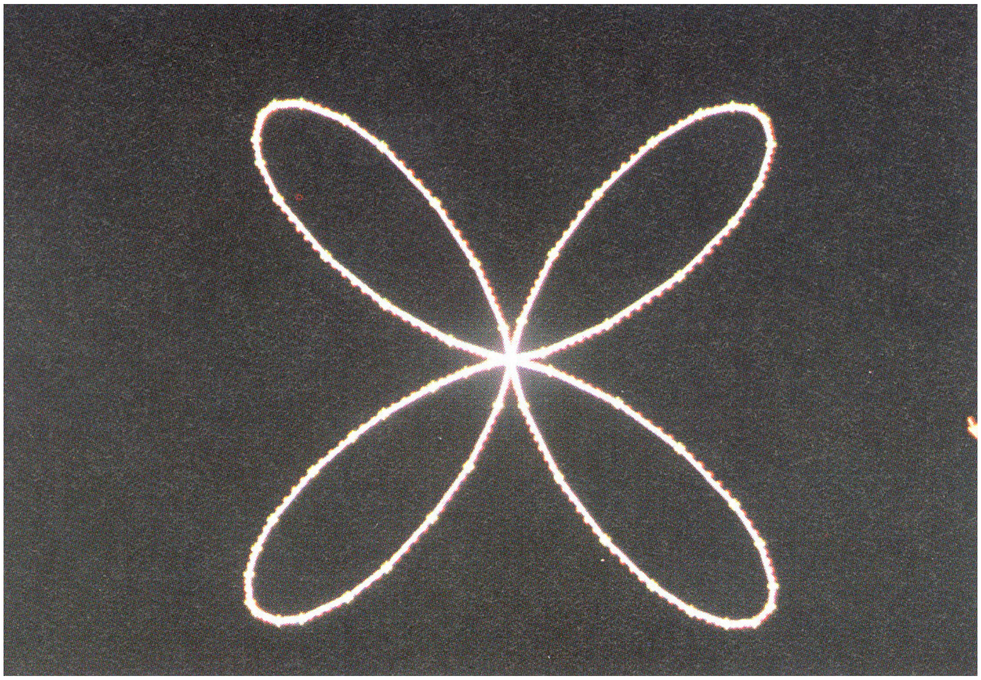

Figure 4: A four-leaved rose

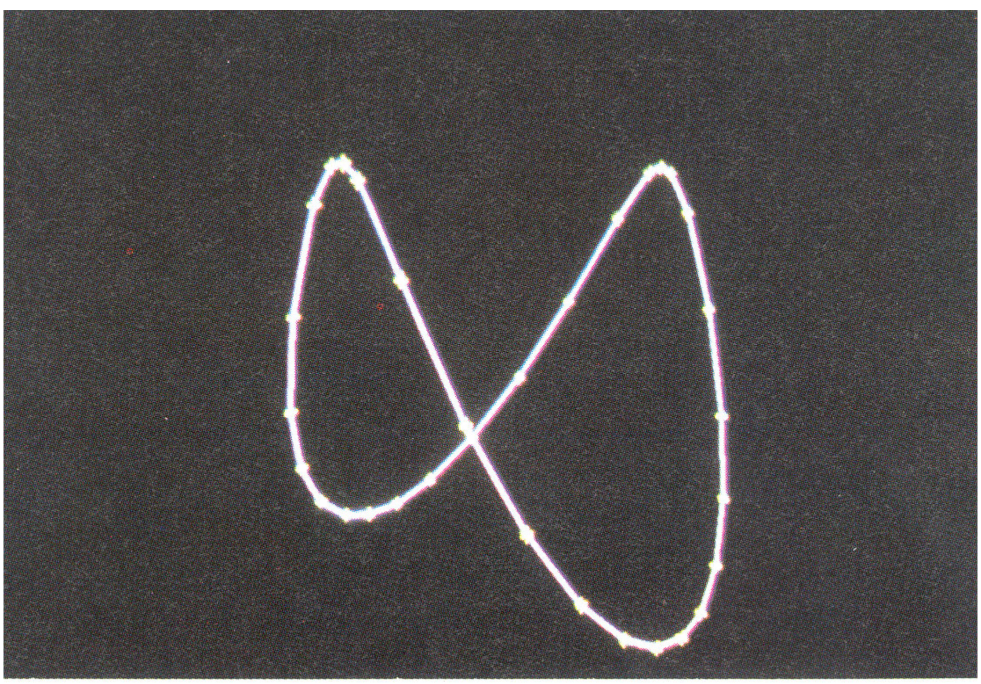

Figure 5: A nonplanar quartic curve

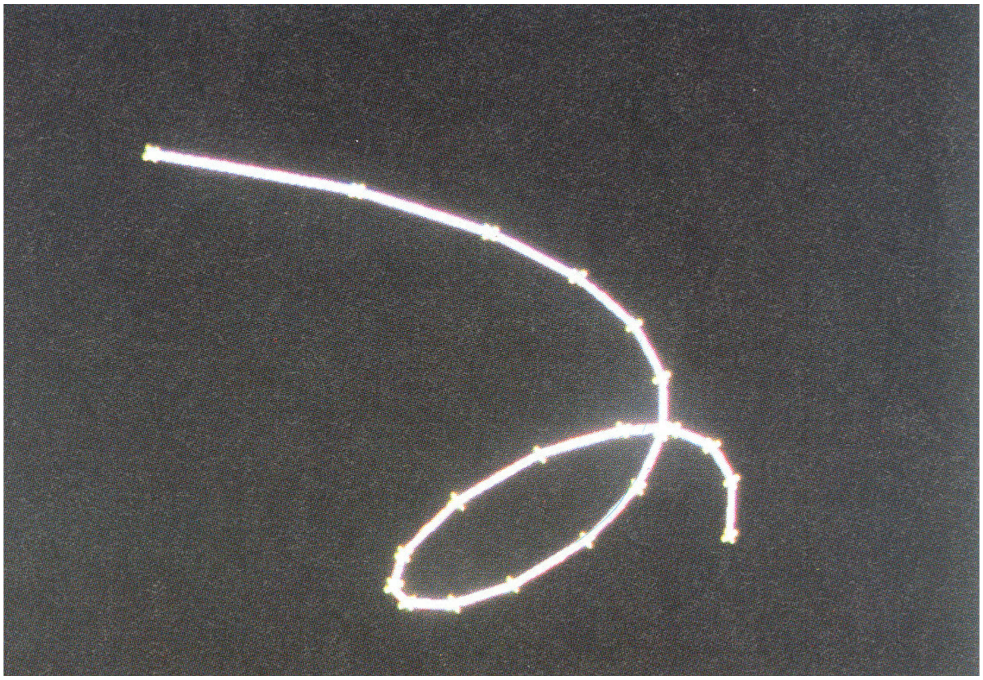

Figure 6: A nonplanar sextic curve

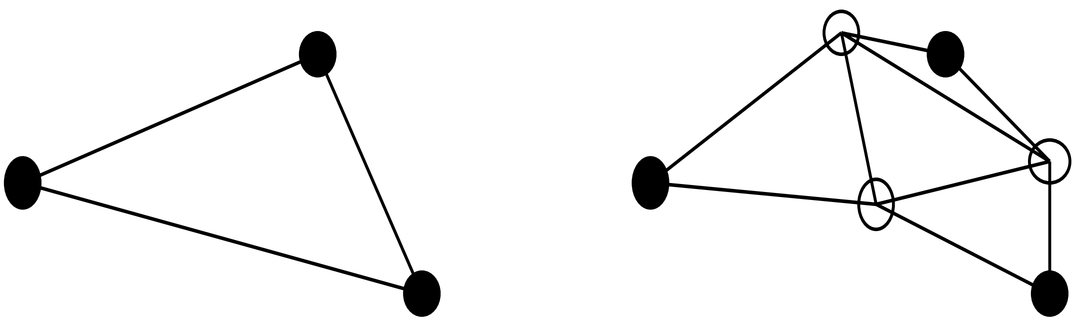

Figure 7: Recursive refinement of a triangle

the whole surface. The space curve tracing and our heuristic algorithm together can be used to produce adaptive polygonization of the necessary portions of *smooth* algebraic surface patches. For example, the following simple procedure produces adaptive polygonization of a triangular algebraic surface patch.

Let $f(x, y, z) = 0$ be a primary surface whose triangular portion clipped by three planes $h_i(x, y, z) = 0$, $i = 1, 2, 3$ is to be polygonized. (See Figure 7.) Initially, the triangle $T_0 = (P_0, P_1, P_2)$ is a rough approximation of the surface patch. Each boundary curve decided by f and h_i is traced to produce a digitized space curve, then its LA of order 2^d for some given d is computed. Then T_0 is refined into four triangles by introducing the 3 points Q_0, Q_1, and Q_2 where Q_i, $i = 0, 1, 2$ is the center point of each LA of order 2^d. The clipping planes of subdivided triangles can be computed by averaging the normals of the two triangles incident to the edge. Then, each new edge is traced, and then its LA of order 2^{d-1} is produced. In this way, this new approximation is further refined by recursively subdividing each triangle

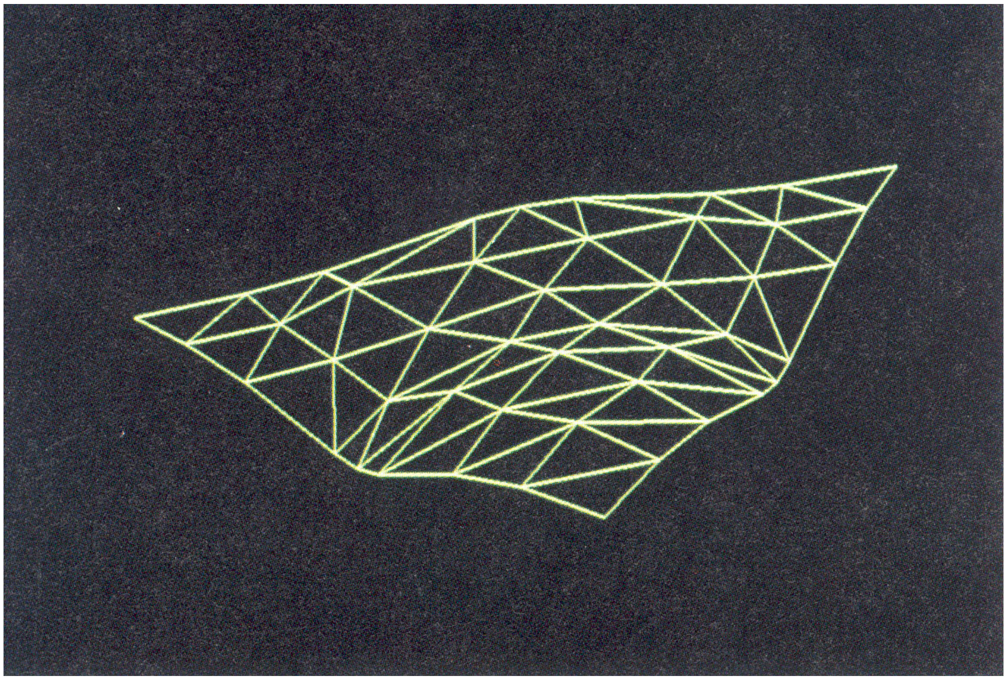

Figure 8: A quartic surface patch

until some criterion is met.

Figure 8 shows an example of the resulting adaptive polygonizations when $d = 3$. A goal here is to put more triangles on the highly curved portion.

While the above method produces a regular (but adaptive) network of polygons, it could be modified to generate more adaptive polygonization. Rather than subdivide all the triangles up to the same level, each triangle is examined to see if it is already a good approximation to the surface portion it is approximating. It is refined only when the answer is no. Some criterions for such local refinement are suggested in [Blo88, AG89]. However, to design an irregular adaptive polygonization algorithm with robust local refinement criterions, is an open problem.

6 APPLICATION II : CONSTRUCTION OF BINARY SPACE PARTITIONING TREE

Binary space partitioning tree (*bsp tree*) has been shown to provide an effective representation of polyhedra through the use of spatial subdivision, and are an alternative to the topologically based b-reps. It represents a recursive, hierarchical partitioning, or subdivision, of d-dimensional space. It is most easily understood as a process which takes a subspace and partitions it by any hyperplane that intersects the subspace's interior. This produces two new subspaces that can be partitioned further.

An examples of a bsp tree in 2D can be formed by using lines to recursively partition the 2D space. Figure 9(a) shows a bsp tree induced partitioning of the plane and (b) shows the corresponding binary tree. The root node represents the entire plane. A binary partitioning of the plane is formed by the line labeled u, resulting in a negative halfspace and a positive halfspace. These two halfspaces are represented respectively by the left and right children of the root. A binary partitioning of each of these two halfspaces may then be performed, as in the figure, and so on recursively. When, along any path of the tree, subdivision is terminated, the leaf node will correspond to an unpartitioned region, called a cell.

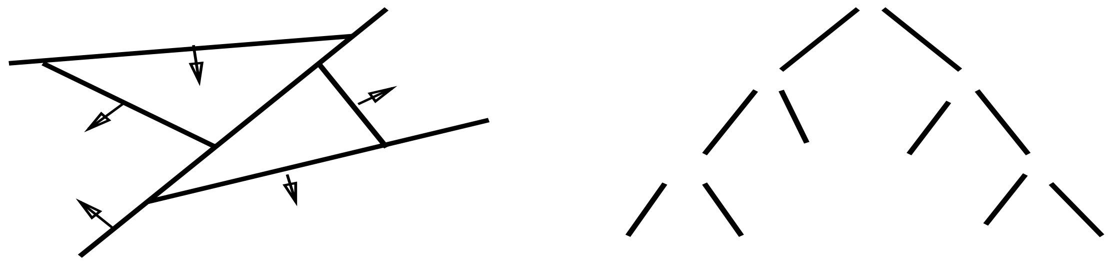

Figure 9: Partitioning of a 2D bsp tree (a), and its binary tree (b)

The primary use of bsp trees to date has been to represent polytopes. this is accomplished by simply associating with each cell of the tree a single boolean attribute $classification ::= \{\ in,\ out\}$. If, in figure 9, we choose cells 1 and 5 to be *in* cells, and to the rest *out* cells, we will have determined a concave polygon of six sides. This method, while conceptually very simply, is capable of representing the entire domain of polytopes, including unbounded and non-manifold varieties. Moreover, the algorithms that use the bsp tree representation of space are simple and uniform over the entire domain. This is because the algorithms only operate on the tree one node at a time and so are insensitive to the complexity of the tree.

A number of bsp tree algorithms are known, including affine transformations, set operations, and rendering (see e.g. [NAT90]). The computational complexity of these algorithms depends upon the shape and size of each tree. Consider point classification for example. The point is inserted into the tree and at each node the location of the point with respect to the node's hyperplane determines whether to take the left or right branch; this continues until a leaf is reached. The cost of this is the length of the path taken. Now if this point is chosen from a uniform distribution of points over some sample space of volume v, then for any cell c with volume v_c at tree depth d_c, the probability p_c of reaching c is simply $\frac{v_c}{v}$ and the cost is d_c. So an optimal expected case bsp tree for point classification would be a tree for which the sum of $p_c d_c$ over all c is minimized. If the embedding space is one dimensional, then this is the classic problem of constructing an optimal binary search tree; a problem solved by dynamic programming.

The essential idea here is that the largest cells should have the shortest paths and smallest cells the longest. For example, satisfying this objective function globally generates bounding volumes as a by-product: if a polytope's volume is somewhat smaller than the sample space's volume, constructing a bounding volume with the first hyperplanes of the tree results is large "out" cells with very small depths. Now, in the general case in which the "query" object q has extent, i.e. is not a point, then q will lie in more than one cell, and a subgraph of the tree will be visited. Thus the cost of the query is the number of nodes in this subgraph. This leads to a more complicated objective function, which we do not intend to examine here, but the intuition taken from point classification remains valid.

We use these ideas in conjunction with the linear approximation methods, described before, to build "good" expected case trees for solids defined as surfaces of revolution (or should we say, that we expect these trees to be good). First, we orthogonally project the curve to be revolved onto the axis of rotation, which we take to be a vertical z-axis. We then partition space with horizontal planes where each plane contains one of the linearly approximated curve points. The bsp tree representing this is a nearly balanced tree, and each cell will contain the surface resulting from the revolution of a single curve segment.

Now the revolution of the curve need not be along a circle, but can be any convex path for which we have constructed a linear approximation. Thus each face of the solid will be a quadrilateral in which the "upper" and "lower" edges lie in consecutive horizontal partitioning planes, are parallel, and are instances of a single path edge at some distance from the axis of revolution that is determined by the revolved curve. Now the bsp subtree for the surface between horizontal planes is obtained by recursively partitioning the path of revolution to form a nearly balanced tree.

The method we use is one that in 2D generates for any n-sided convex polygon a corresponding nearly balanced bsp tree of size $O(n)$ and height $O(\log n)$. The path curve is first divided into four sub-curves, one for each quadrant, and a hyperplane containing the first and last points is constructed. By convexity, a sub-curve lies entirely in one halfspace of its corresponding hyperplane, and we call that halfspace the "outside" halfspace and the opposite halfspace the "inside" halfspace. The intersection of the four inside halfspaces is entirely inside the polygon, and so forms an in-cell of the bsp tree. We then construct independently a tree for each sub-curve recursively.

We first choose the median segment of the sub-curve and partition by the plane of the corresponding face. Since the path curve is convex, all of the faces will be in the inside halfspace of this plane and an out-cell can be created in its outside halfspace. Now each non-horizontal edge of the median face is used to define a partitioning plane which also contains the first/last point of the sub-curve. All of the faces corresponding to this sub-curves' edges are in the outside halfspace, and so an in-cell can be created in its inside halfspace. We have now bisected the sub-curve by these planes which contain no faces and can recurse with them. The recursion continues until only a small number of faces/segments remain, say 6, at which point only face planes are used for partitioning, since the cost of the non-face partitioning planes out-weights their contribution to balancing the tree. The result for a path curve of n edges is a nearly balanced tree of size $< 3n$ and height $O(logn)$.

In some sense, we have constructed a tree that is the cross product of the path curve and the revolved curve; we build a tree of horizontal planes that partitions the revolved curve, and then we form "slices" of the object by constructing a tree for each segment of the path curve. If the revolved curve has m segments, then the number of faces is nm and the bsp tree is of size $O(nm)$ and height $O(\log nm) = O(\log n + \log m)$.

The object in figure 10 was made by rotating the curve in figure 3 around an ellipse. Its bsp tree is quite well balanced. The goblet in figure 11 was made by constructing two objects

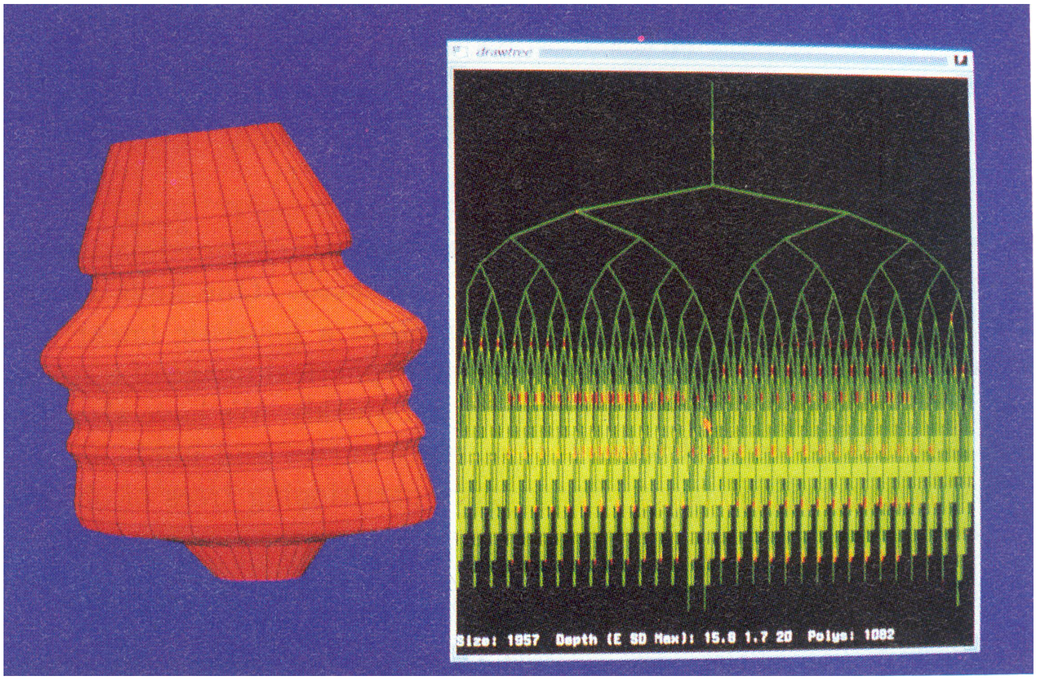

Figure 10: A Human profile rotated

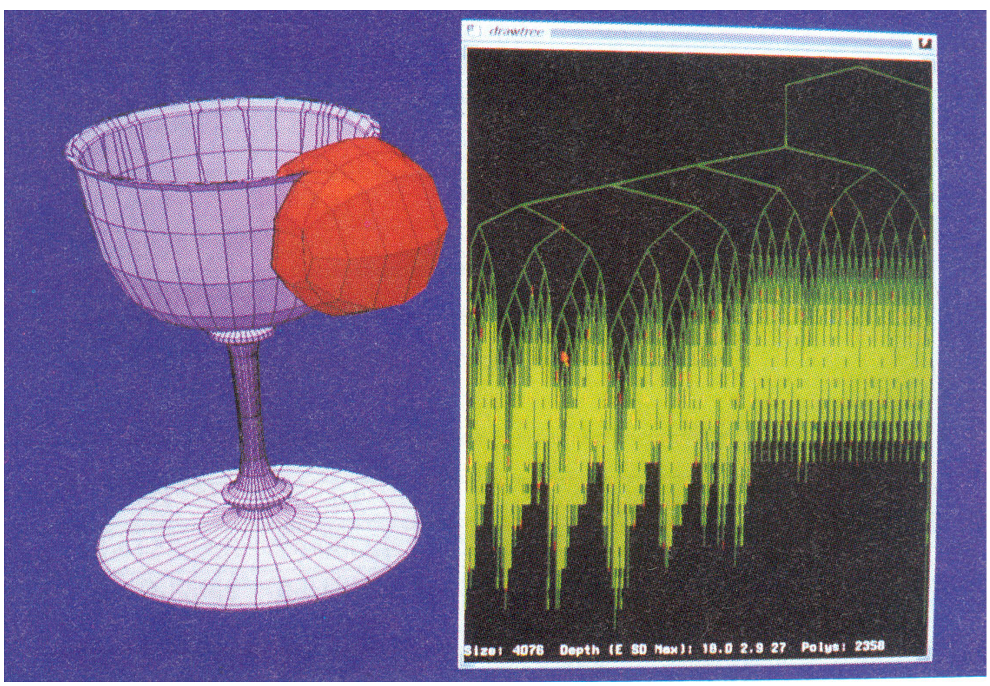

Figure 11: A goblet

using the curve in figure 3, and then applying a difference operation to carve a hole on the goblet. The bsp tree in figure 11 was obtained after applying the difference operation, and then a union operation for the red ball. It is observed that set operations on well-balanced bsp trees result in well-balanced trees. The set operation and display were done in **Sculpt** [Nay90] which is an interactive modeling system based on bsp trees.

7 CONCLUSION

In this paper, we have discussed the problem of piecewise linear approximation of an arbitrary digitized 3-D curve. Two algorithms have been presented. One finds an optimal linear approximation at a high expense. The other computes a heuristic linear approximation, based on the fundamental notions of curve length and spherical image of a space curve. This heuristic algorithm finds a good linear approximation quickly. We have also shown that our heuristic algorithm can be applied to display of space curves and implicit surfaces, and to adaptively constructing well-balanced binary space partitioning trees of objects defined by revolution.

Acknowledgment The authors would like to thank the anonymous referees for their helpful comments. Insung Ihm wishes to thank AT&T Bell Laboratories in Murray Hill, New Jersey, for providing nice environment where most of this work was done during the summer in 1990.

Insung Ihm is currently a Ph.D. student and member of the Computing about Physical Objects Lab. in the Department of Computer Sciences at Purdue University. His research interests are in Computer Aided Geometric Design and Computer Graphics. He received his B.S. and M.S. degrees in Computer Science from Seoul National University in 1985, and Rutgers University in 1987, respectively, and worked at AT&T Bell Labs. in Murray Hill, NJ during the summer in 1990.
Address: Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA.

Bruce Naylor has been a Member of Technical Staff at AT&T Bell Labs. in Murray Hill, NJ since 1986. Previously, he was on the faculty of Information and Computer Science at Georgia Institute of Technology in Atlanta, GA from 1981-1986. He did his graduate studies at the University of Texas at Dallas, receiving a Master and Ph.D. in Computer Science in 1979 and 1981, respectively. His B.A., received in 1976, is from the University of Texas at Austin where he majored in Philosophy.
Address: AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974, USA.

A The Performances

n	109				
m	4	8	16	32	64
LA_0	5.25619e-1	2.21325e-1	8.09768e-2	2.66073e-2	8.98677e-3
LA_k	4.02838e-1	1.12507e-1	3.08774e-2	1.23695e-2	3.76437e-3
(k)	(5)	(9)	(17)	(16)	(14)
LA^*	4.02838e-1	1.12507e-1	3.04525e-2	8.94592e-3	2.76188e-3
LA_k/LA^*	1.000	1.000	1.014	1.393	1.363

Table 1: The curve in Figure 3

n	408					
m	2	4	8	16	32	64
LA_0	3.41045e-1	2.24775e-1	5.47018e-2	1.42360e-2	3.80354e-3	1.19389e-3
LA_k	3.41045e-1	9.53494e-2	2.75688e-2	8.92481e-3	2.95337e-3	8.66971e-4
(k)	(0)	(86)	(50)	(80)	(66)	(381)
LA^*	2.73563e-1	8.95572e-2	2.63278e-2	6.62991e-3	2.02041e-3	5.45924e-4
LA_k/LA^*	1.247	1.065	1.047	1.346	1.462	1.588

Table 2: The curve in Figure 4

n	237			
m	8	16	32	64
LA_0	1.03375e-1	6.11455e-2	2.95784e-2	8.24535e-3
LA_k	6.07749e-2	2.90067e-2	7.76577e-3	5.63440e-3
(k)	(12)	(17)	(20)	(5)
LA^*	5.87190e-2	2.06813e-2	5.12219e-3	1.75973e-3
LA_k/LA^*	1.035	1.403	1.516	3.202

Table 3: The curve in Figure 5 (goblet)

n	404				
m	4	8	16	32	64
LA_0	2.01399e-0	3.63921e-1	9.66444e-2	2.67485e-2	8.39998e-3
LA_k	1.86220e-0	2.26435e-1	7.78874e-2	2.24510e-2	6.98709e-3
(k)	(4)	(32)	(16)	(10)	(8)
LA^*	1.85530e-0	2.26435e-1	7.60669e-2	2.05613e-2	5.69636e-3
LA_k/LA^*	1.004	1.000	1.024	1.092	1.227

Table 4: The curve in Figure 7

n	234				
m	4	8	16	32	64
LA_0	7.24214e-1	1.72242e-1	5.32029e-2	1.64083e-2	1.60168e-2
LA_k	4.81844e-1	1.34728e-1	3.69400e-2	1.53000e-2	3.80315e-3
(k)	(15)	(15)	(13)	(2)	(11)
LA^*	4.81844e-1	1.34728e-1	3.65433e-2	1.05332e-2	3.16273e-3
LA_k/LA^*	1.000	1.000	1.011	1.453	1.202

Table 5: The curve in Figure 8

B List of Figures

(b) $n = 404$, $m = 32$

6. Figure 6 : A nonplanar sextic curve

   (a) equation : $(f(x, y, z) = y^2 - x^2 - x^3,\ g(x, y, z) = z - x^2 + x - 2)$

   (b) $n = 234$, $m = 20$

7. Figure 8 : A quartic surface patch

   (a) equation : $f(x, y, z) = 0.01853292z^4 - 1.14809166y^2z^2 - 1.14809166x^2z^2 + 0.99982830z^2 - 1.16662458y^4 - 1.14809166x^2y^2 + 2.1849858y^2 + 0.01853292x^4 + 0.99982830x^2 - 0.72183450$

   (b) $d = 3$

8. Figure 10 A human profile rotated

   (a) The primary curve : the curve in Figure 3 with 32 segments.

   (b) The auxiliary curve : an ellipse with 32 segments.

9. Figure 11 A goblet

   (a) The primary curve : the curve in Figure 3 with 20 segments.

   (b) The auxiliary curve : a circle with 32 segments.

References

[AG89] E.L. Allgower and S. Gnutzmann. Polygonal meshes for implicitly defined surfaces. manuscript, Nov. 1989.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.

[BHHL88] C. Bajaj, C. Hoffmann, J. Hopcroft, and R. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5:285–307, 1988.

[BI88] C. Bajaj and I. Ihm. Hermite interpolation using real algebraic surfaces. In *Proceedings of the Fifth Annual ACM Symposium on Computational Geometry*, pages 94–103, Germany, 1988.

[Blo88] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[Can71] A. Cantoni. Optimal curve-fitting with piecewise linear functions. *IEEE Transactions on Computers*, c-20:59–67, 1971.

[Cd80] S.D. Conte and C. deBoor. *Elementary Numerical Analysis: An Algorithmic Approach*. McGraw-Hill, New York, third edition, 1980.

[deB78] C. deBoor. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.

[Dun86] J.G. Dunham. Optimum uniform piecewise linear approximation of planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:67–75, Jan. 1986.

[FWL89] C. Fahn, J. Wang, and J. Lee. An adaptive reduction procedure for the piecewise linear approximation of digitized curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(9):967–973, Sep. 1989.

[Han83] P. Hanrahan. Ray tracing algebraic surfaces. *Computer Graphics*, 17(3):83–90, 1983.

[HH86] C. Hoffmann and J. Hopcroft. Quadratic blending surfaces. *Computer Aided Design*, 18(6):301–306, 1986.

[II86] H. Imai and M. Iri. Computational geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics and Image Processing*, 36:31–41, 1986.

[Kd88] N. Kehtarnavaz and R.J.P. deFigueiredo. A 3-D contour segmentation scheme based on curvature and torsion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):707–713, Sep. 1988.

[Kre59] E. Kreyszig. *Differential Geometry*. University of Toronto Press, 1959.

[MO88] A. A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*, pages 87–95. North-Holland, 1988.

[Mon70] U. Montanari. A note on minimal length polygonal approximation to a digitized contour. *Comm. ACM*, 13:41–47, 1970.

[MS85] A. Middleditch and K. Sears. Blend surfaces for set theoretic volume modeling system. *Computer Graphics*, 19(3):161–170, 1985.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. *Computer Graphics*, 24(4), Aug. 1990.

[Nay90] B. Naylor. Sculpt : An interactive solid modeling tool. In *Proc. of Graphics Interface*, Halifax, Nova Scotia, May 1990. Graphics Interface '90.

[O'N66] B. O'Neill. *Elementary Differential Geometry*. Academic Press, 1966.

[Pav82] T. Pavlidis. *Algorithms for Graphics and Image Processing*, pages 281–297. Computer Sciences, New York, 1982.

[PH74] T. Pavlidis and S.L. Horowitz. Segmentation of plane curves. *IEEE Transactions on Computers*, c-23(8):860–870, Aug. 1974.

[Pie87] L. Piegl. Interactive data interpolation by rational Bezier curves. *IEEE Computer Graphics and Applications*, pages 45–58, July 1987.

[PK89] N.M. Patrikalakis and G.A. Kriezis. Representation of piecewise continuous algebraic surfaces in terms of B-splines. *The Visual Computer*, 5(6):360–374, Dec. 1989.

[Ram72] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:244–256, 1972.

[Ric64] J.R. Rice. *The Approximation of Functions*, volume 1. Addison-Wesley, Reading, Mass., 1964.

[RO87] A.P. Rockwood and J.C. Owen. Blending surfaces in solid modeling. In G. Farin, editor, *Geometric Modeling : Algorithms and New Trends*, pages 367–383. SIAM, Philadelphia, 1987.

[Rob85] J. Roberge. A data reduction algorithm for planar curves. *Computer Vision, Graphics and Image Processing*, 29:168–195, 1985.

[RW74] K. Reumann and A.P.M. Witkam. Optimizing curve segmentation in computer graphics. In A. Gunther, B. Levrat, and H. Lipps, editors, *International Computer Symposium*, pages 467–472. American Elsevier, New York, 1974.

[Sed85] T.W. Sederberg. Piecewise algebraic surface patches. *Computer Aided Geometric Design*, 2:53–59, 1985.

[SG80] J. Sklansky and V. Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12:327–331, 1980.

[SZ89] T.W. Sederberg and A.K. Zundel. Scan line display of algebraic surfaces. *Computer Graphics*, 23(3):147–156, 1989.

[Tou85] G. T. Toussaint. On the complexity of approximating polygonal curves in the plane. In *Proceedings of the IASTED International Symposium on Robotics and Automation*, pages 59–62, Lugano, Switzerland, 1985.

[War86] J. Warren. *On algebraic surfaces meeting with geometric continuity*. PhD thesis, Cornell University, 1986.

[WD84] K. Wall and P.E. Danielsson. A fast sequential method for polygonal approximation of digitized curves. *Computer Vision, Graphics and Image Processing*, 28:220–227, 1984.

[Wil78] C.M. Williams. An efficient algorithm for the piecewise linear approximation of planar curves. *Computer Graphics and Image Processing*, 8:286–293, 1978.