Computer-Aided Geometric Design Based 3-D Models for Machine Vision

Bir Bhanu and Chih-Cheng Ho

Department of Computer Science
The University of Utah
Salt Lake City, Utah 84112, USA

Abstract: Model based recognition is one of the key paradigms in computer vision and pattern recognition. However, at present there is an absence of a *systematic approach* for building geometrical and functional models for a large class of 3-D objects used in industrial environments. In this paper we present a Computer-Aided Geometric Design (CAGD) based approach for building 3-D models which can be used for the recognition and manipulation of 3-D objects for industrial machine vision applications. We present the details of the design on a relatively simple object named "Green Piece," and the complex automobile "Renault Piece" used in computer vision and pattern recognition community. A new algorithm is presented which uses the above geometric design and allows the points on the surface of the object to be sampled at the desired resolution, thus allowing the construction of *multiresolution* 3-D models. The resulting data structure of points includes coordinates of the points in 3-D space, surface normals and information about the neighboring points.

1. Introduction

The emergence of CIM (Computer Integrated Manufacturing) technology as a driving force in manufacturing engineering has provided opportunities and challenges to use geometric and functional models of real-world 3-D objects for the task of visual recognition and manipulation of these objects by robots [2]. CIM technology provides the database of objects as a byproduct of the design process. It allows the model-based recognition of 3-D objects to be simulated even before these objects are physically created. In this paper we present our ongoing work in defining how these designs could be used or modified in novel ways so as to be suitable for the task of recognition and manipulation.

A formal CAGD system contains an interactive designer interface, graphic display utilities, model analysis tools and automatic manufacturing interfaces. It is a suitable environment for design purpose. However, the models it generates do not contain features which are important in computer vision. A *systematic approach* to build heterogeneous vision models [2] is to construct them from the CAGD database and incorporate those features which are crucial for object recognition and manipulation. Here are some approaches used in our ongoing work: (1) Universal representation of objects by *surface points*. These points can be sampled at a desired resolution and the neighborhood and surface normal information is provided which can be used in building higher level description of the model and shape matching [1]. (2) Intrinsic surface characteristics defined by *surface curvatures*. The local surface can be characterized by curvatures alone. In B-splines based Alpha_1 CAGD system [3], derivatives are embedded inherently in its control mesh and knot vectors. Surface normals and curvatures can be obtained as a byproduct of many B-splines computations. (3) Surface representation by *edges/arcs* and *local shape features*. Edges and arcs can be extracted as boundaries of surface patches used in the design process. A more precise way is to define edges as the local extrema of curvature or to use some 3-D edge detection techniques. (4) High level *volumetric* and *sweep* representations. From the CAGD design procedure, we can construct a CSG (Constructive Solid Geometry) like representation where we can put functional information. Also, we can convert the CAGD model into a generalized cylinder model by applying an object decomposition technique followed by an axis extraction procedure. In this paper we present approach (1) for model building using the Alpha_1 CAGD system developed at the University of Utah.

[1]This work was supported in part by NSF Grants DCR-8506393, DMC-8502115, ECS-8307483 and MCS-8221750

2. Model Building Using the Alpha_1 CAGD System

Alpha_1 models the geometry of solid objects by representing their boundaries as non-uniform rational B-splines. Alpha_1 uses the Oslo algorithm [3] for knots insertion. Rational B-splines are an ideal design tool, they are simple yet powerful, all quadratic surfaces which are used as primitives in CSG can be represented exactly. Other advantages include good computational and representational properties of the spline approximation: the variation diminishing property, the convex hull property and the local interpolation property. Alpha_1 supports several modeling paradigms, including direct manipulation of the B-spline surfaces, creation and combination of primitive shapes, and high-level shape operators such as bend, twist, warp and sweep. It also allows set operations on surface patches which make the modeling task easy and complete. Here are some guidelines in using Alpha_1: (1) Analyze the object, a complex object is decomposed into simpler parts which are designed more easily. (2) Make a precise measurement of parameters. (3) Maintain validity of the model by setting correct orientation and adjacency information of each patch. (4) Perform the appropriate transformations and Boolean operations.

To design simple objects such as the "Green Piece" (Fig. 1) which has many local features, we build the complete object in a stepwise manner. First, we design the plate and all holes as in Fig. 2(a), then the dent part and scratches of Figs. 2(b) and 2(c). To design these parts, we first design curves using B-splines and then use various high level operators for surface construction, such as revolving a curve about an axis, extruding a curve in some direction and filling the surface between two curves. There are seven threads in the Green Piece each of them is designed by filling two surfaces between two twisted curves. Fig. 2(c) shows the center one, the others are similar except their radii and pitches. This design can be used in manufacturing the green piece on a numerically controlled milling machine.

For objects like the Renault Piece (Fig. 3) which contains sculptured freeform surfaces, we divide it into a set of simpler parts though the partitioning is not so straightforward. Here we divide it into five sub-parts: left head, right head, neck, base plate, and back bump (Fig. 4). For the right head, left head and back bump, we find all sharp edges and then construct the surfaces from them as before. For the base plate and the neck part, we need some pseudo sharp edges which are the intersection of the surface planes. Then we construct these surfaces but leave small gaps between them where we use cubic B-splines patches to produce the rounded edges. Fig. 5 shows rendered images of these parts and Fig. 6 shows intersection curves of them, which are computed to obtain the complete object using set operations performed by the combiner in Alpha_1. Figs. 7 and 8 are the final CAGD model for Renault Piece and Green Piece respectively.

3. Multiresolution Surface Point Extraction

The set operations over B-spline surfaces are not closed, parts of the final CAGD model are represented as polygons. Our strategy is to subdivide all the B-spline surfaces into polygons so as to make the problem uniform. By applying a contour filling algorithm, we get interior line segments of the polygons, and extract points along these segments at a desired resolution.

An edge based contour filling algorithm is described in [4] which requires expensive preprocessing on the contours. It is used in applications, such as surface shading, where the same contour is used repeatedly. In our case, the number of polygons in a model is very large and we extract only a small number of points from each one of them. Instead of sorting and marking the edges, we use topological information of the intersection points. Since the number of vertices is usually much larger then the number of intersection points for one line segment, its time complexity is linear in the average case.

3.1. Contour Filling Algorithm

The main element of contour filling is to find the intersection segments of a line (scanline) and the polygon. They can be obtained by first finding all intersection points and then deciding which parts are inside the polygon. There are three kinds of intersection points: start, end, and middle point. Suppose we travel along a line from left to right, the start point is defined as a point whose left neighborhood is outside the polygon. Similarly, the end point is the one whose right neighborhood is outside the polygon, and a middle point is the one for which both left and right neighborhoods are inside the polygon. In Fig. 9, point A is a start point, B and C are middle points and D is an end point. However, point E is both a start and an end point. Now we want to characterize the intersection points into these three classes. Then the segments between all start/end point pairs are inside the polygon.

To determine the kind of an intersection point, we need its topological information. In Fig. 9, the space is divided by line ab into two regions, I and II. There are five intersection points: A, B, C, D, and E. We find the incoming and outgoing directions (in,out) of an edge (along the contour) when it passes through an intersection point. For the edge going from region II to region I we mark its direction as 1 and if it goes from region I to region II we mark it as -1. And if the edge is lying on the line itself, we mark it as 0. For example, point A will be marked as (1,1), point C as (0,1) and point E as (1,-1). This information is obtained when we find the intersection points. There are nine different combinations of the (in,out) directions. They can be classified into four kinds only: flat (in = out = 0), tangent (in = -out ≠ 0), cut (in = out ≠ 0), and flat-cut (only one of them is 0), see Fig. 10. Now the problem is to map these four kinds of intersection points into three classes - start, middle, and end points.

Obviously, the flat point must be a middle point. The tangent point must be an end point if it is a start point otherwise it is a middle point. The cut point can be either a start or an end point. But the flat-cut point can be anyone of them. There is an important property of the edge directions: the out direction of an end point must be opposite to the in direction of its start point. As an example in Fig. 9, the start point A has 1 as its in direction and the end point D has -1 as its out direction. But using this property alone may not result the correct mapping when the contour is in the opposite direction of the cutting line. As shown in Fig. 11, the direction of the contour between D and C is from right to left but all intersection points are labeled assuming the cutting line is from left to right. The end point should be D instead of C which is a middle point. It can be resolved by exchanging the in and out directions of these points. Thus, there is a restriction on the in/out directions which is used to tell if such a correction is required. Assuming that the contour does not intersect itself, where the in/out directions cannot be defined uniquely, for any two adjacent intersection points, the out direction of the first one and the in direction of the second one must be both zero or both non-zero if they are marked correctly. In Fig. 11, there is a conflict between B and C, the out direction of B is not 0 but the in direction of C is 0. The in/out direction of C is therefore changed to (-1,0) and point D is changed to (0,-1), since the conflict occurs on point D after we change point C. After these corrections we can find that point D is the end point corresponding to point A. The final algorithm is:

Inputs: List of intersection points, each point contains both geometric (coordinates) and topological (in/out directions) information.

Outputs: A list of start/end point pairs, each represents one interior line segment.

Assumptions: The contour is closed and does not intersect itself.

Procedure: It includes the following steps:

1. Sort the intersection points along the cutting line.
2. For each intersection point, **if** it is the first one or the previous one is an end point, **then** it is a start point and **do** step 3 **else do** step 4.
3. **If** it is a tangent point, **then** it is also an end point **else if** its in direction is 0, **then** exchange the in and out direction of this point. Save the in direction of this start point.
4. **If** it is either a flat point or a tangent point, **then** it is a middle point **else do** step 5.
5. **If** there is a conflict regarding the restriction as described above, **then** exchange its in and out direction. Finally, **if** the out direction of this point is opposite to the saved in direction from step 3 **then** it is the end point **else** it is a middle point.

Fig. 12 shows a two dimensional example. Fig. 12(a) is the input polygon which is taken from the output of the combiner. Each cross represents a vertex of the contour. Some of them appear to be redundant were kept to maintain the adjacency information between neighboring polygons. When considering a single polygon they can be removed. Fig. 12(b) is the result of the above algorithm applied to Fig. 12(a). Finally, in Fig. 12(c), we extract the surface points from these line segments by a user defined resolution.

The time complexity of this algorithm is $O(n + m*\log(m))$, where n is the number of vertices of the input contour and m is the number of intersection points of one cutting line. Usually the number of intersection points is much less then the number of vertices, and the complexity is linear. In the worst case, m is equal to (n-1), number of edges minus 1, and it becomes $O(n*\log(n))$, the same as the edge based algorithm.

3.2. Surface Point Extraction Using Contour Filling Algorithm

We use the contour filling algorithm described above to find the surface points and their normals. Since the algorithm uses topological information only, it can be applied to 3-D polygons as long as we change the concept of cutting "lines" to cutting "planes". The resolution in one sampling direction is defined as the maximum distance between any two adjacent points. The given resolution is ensured by finding the maximum projection plane of each polygon among either x-y, y-z or z-x planes from its normal vector and using cutting planes orthogonal to it. Thus the distance between any two adjacent points is less than the grids spacing times $3^{1/2}$.

Figs. 13 and 15 show the surface points extracted from the models of Green Piece and Renault Piece at 0.2 inch resolution. In Figs. 14 and 16, we show the surface normals at 0.4 inch resolution. These were computed using bi-linear interpolation of normals at vertices. Fig. 17 exhibits three samplings of surface points of Renault Piece at various resolutions. They are taken from only one view and can be used to simulate the real range data.

4. Conclusions

In this paper we have presented a new technique for the representation of 3-D objects by surface points which can be sampled at any desired resolution. Other approaches toward the multi-representation vision system and matching based on these representations are under investigation. These results will be presented in the future.

References

[1] B. Bhanu. Representation and Shape Matching of 3-D Objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence* PAMI-6(3):340-351, May, 1984.

[2] B. Bhanu and T. Henderson. CAGD Based 3-D Vision. In *International Conference on Robotics and Automation*. IEEE, March, 1985.

[3] E. Cohen, T. Lyche and R.F. Riesenfeld. Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics. *Computer Graphics and Image Processing* 14(2):87-111, October, 1980.

[4] T. Pavlidis. *Algorithms for Graphics & Image Processing*. Computer Science Press, 1982.

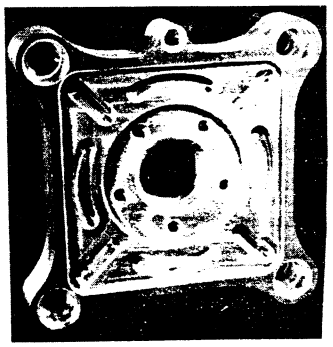
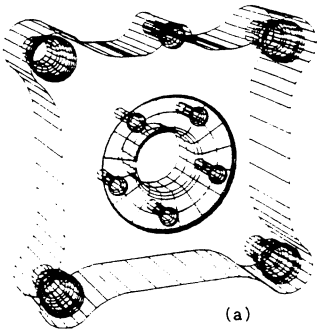


Fig. 1 Green Piece Object



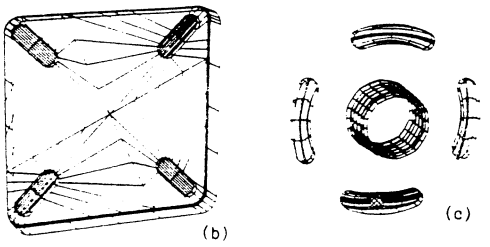


Fig. 2 Sub-parts of Green Piece CAGD Model

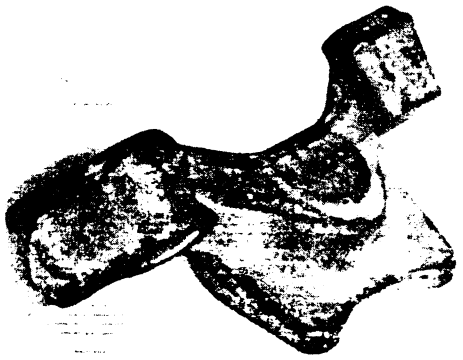


Fig. 3 Renault Piece Object

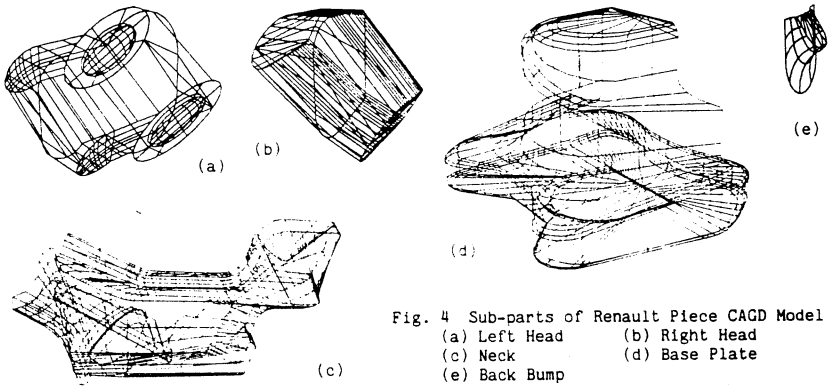


Fig. 4 Sub-parts of Renault Piece CAGD Model
(a) Left Head (b) Right Head
(c) Neck (d) Base Plate
(e) Back Bump

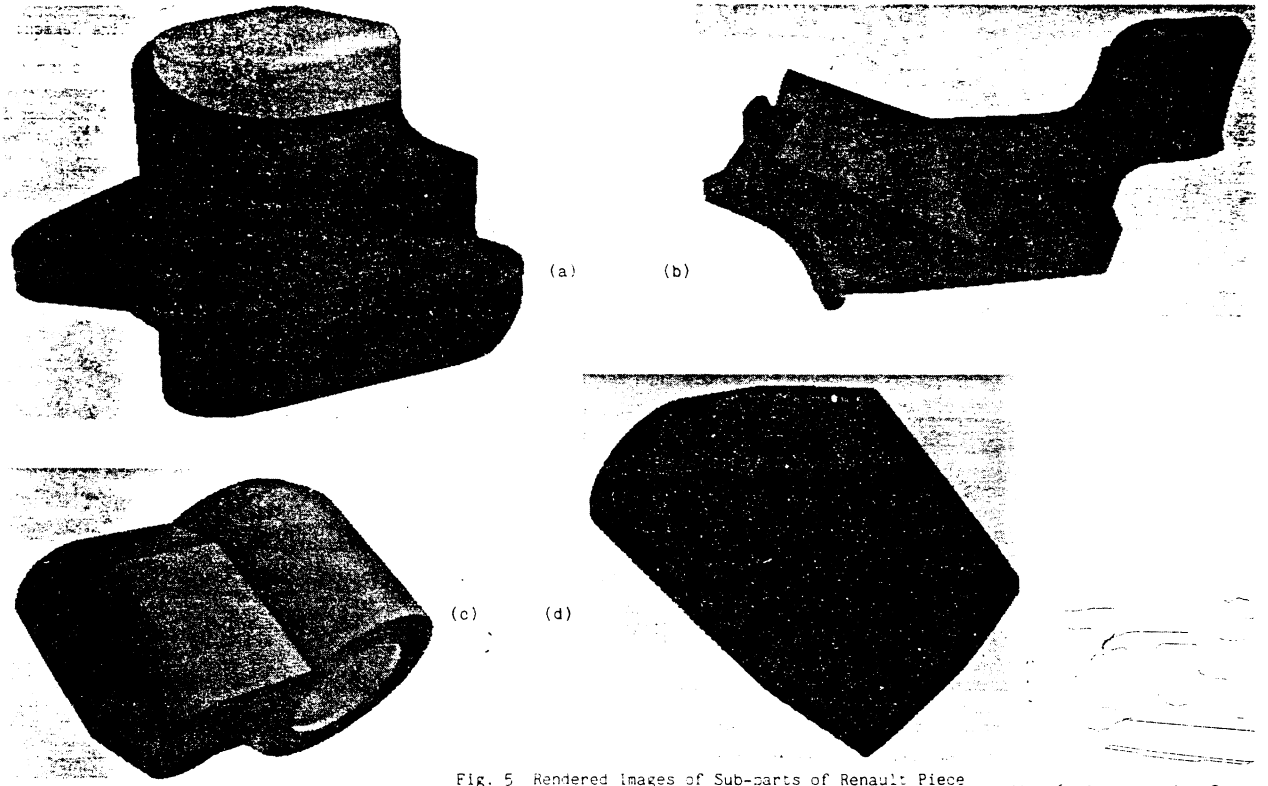
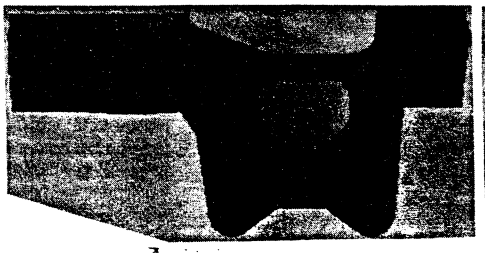


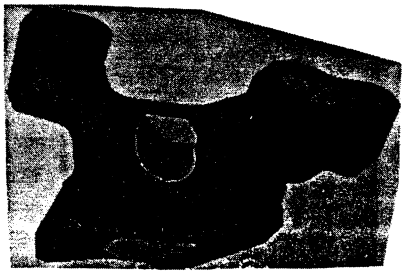
Fig. 5 Rendered Images of Sub-parts of Renault Piece
(a) Base Plate (b) Neck
(c) Left Head (d) Right Head

Fig. 6 Intersection Curves of Sub-parts of Renault Piece

(a) Front View



(b) Top View



(c) Back View

Fig. 7 Various Views of Designed CAGD Model for Renault Piece

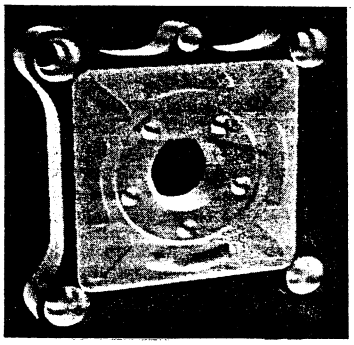


Fig. 8 Designed CAGD Model for Green Piece

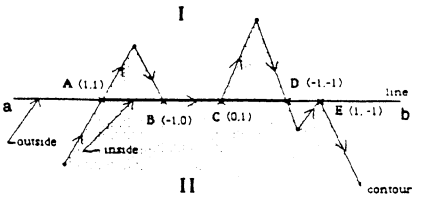


Fig. 9 Direction of Edges

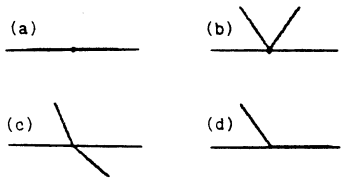


Fig. 10 Four Kinds of Junction Points
(a) Flat Point (b) Tangent Point
(c) Cut Point (d) Flat-Cut Point

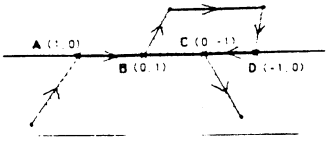


Fig. 11 The Restriction on In/Out Directions

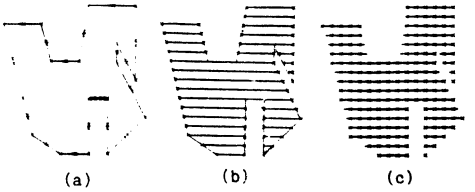


Fig. 12 A Two-Dimensional Example
(a) Input Polygon (b) Interior Line Segments
(c) Sampled Surface Points

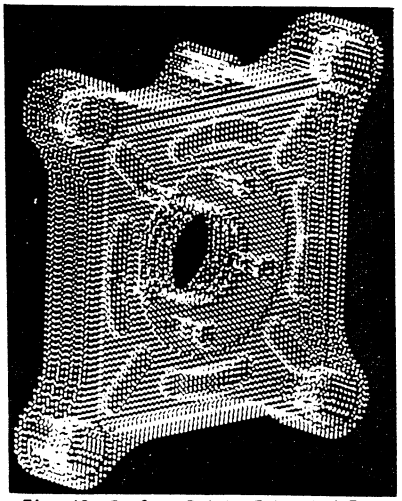


Fig. 13 Surface Points Extracted From the CAGD Model of Green Piece, 0.2 Inch Resolution

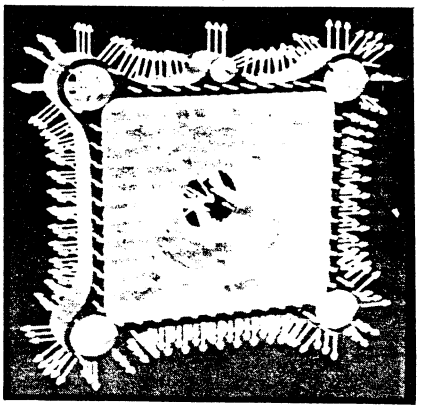


Fig. 14 Normals at the Surface Points for Green Piece, 0.4 Inch Resolution

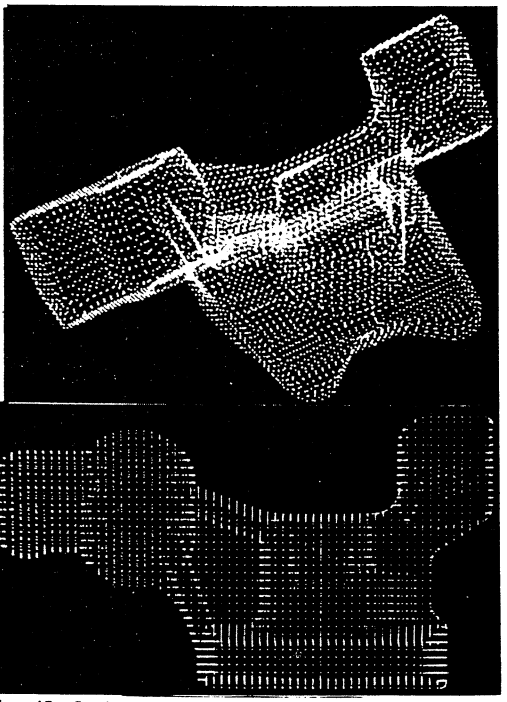


Fig. 15 Surface Points Extracted From the CAGD Model of Renault Piece, 0.2 Inch Resolution

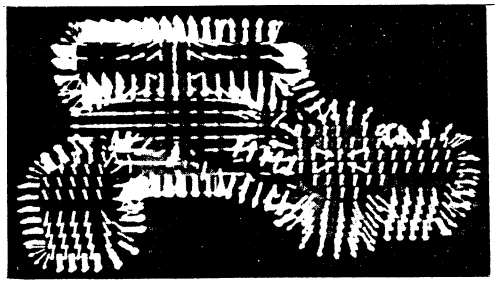


Fig. 16 Normals at the Surface Points for Renault Piece, 0.4 Inch Resolution

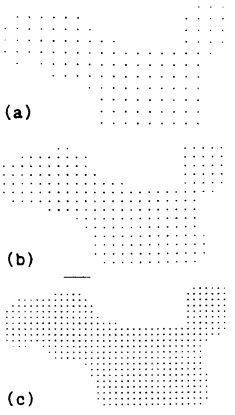


Fig. 17 Sampled Surface Points of the Renault Piece in Various Resolutions
(a) 0.4 Inch Spacing (b) 0.3 Inch Spacing
(c) 0.2 Inch Spacing

IAPR afcet

EIGHTH INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION

PARIS, FRANCE
OCTOBER 27-31, 1986

PROCEEDINGS

Sponsor : IAPR, International Association for Pattern Recognition
Organizer : AFCET, Association Francaise pour la Cybernetique Economique et Technique

Volume 1

IEEE Computer Society Order Number 742
Library of Congress Number 86-81419
IEEE Catalog Number 86CH2342-4
ISBN 0-8186-0742-4

IEEE COMPUTER SOCIETY PRESS

IAPR

afcet

EIGHTH INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION

PARIS, FRANCE OCTOBER 27-31, 1986

PROCEEDINGS

Sponsor : IAPR, International Association for Pattern Recognition

Organizer : AFCET, Association Française pour la Cybernétique Economique et Technique