Pattern Recognition Letters 5 (1987) 349–356
North-Holland

May 1987

3-D model building for computer vision

B. BHANU*, C.C. HO and T. HENDERSON
Department of Computer Science, University of Utah, Salt Lake City, UT 84112, USA

Received 30 March 1986
Revised 14 October 1986

Abstract: This paper presents a Computer-Aided Geometric Design (CAGD) based approach for building 3-D models. A new method is given which allows the points on the surface of the designed object to be sampled at the desired resolution. The resulting data structure includes 3-D coordinates of the points, surface normals and neighborhood information.

Key words: 3-D models, B-splines, CAD, CAGD, multiresolution 3-D data, surface filling.

1. Introduction

There has been an absence of a *systematic approach* for building 3-D models in computer vision. The emergence of CIM (Computer Integrated Manufacturing) technology 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 [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. For approaches in computer vision on 3-D model building from multiple views refer to [1, 2, 9].

The B-splines based CAGD (Computer Aided Geometric Design) model provided by the Alpha_1 system [2, 4, 5, 6, 10] contains either B-spline surface patches or subdivided polygons or both. A surface is represented by its control mesh and related parameters and a polygon is represented by its contour which is stored as a sequence of vertexes. Both surface patches and polygons may result in the design process because of the subdivision process used in finding the intersection of two B-spline surfaces [10]. There are several approaches used to generate vision models from the CAGD model. These allow multiple representations of objects from different classes and even multiple representations for the same object [3]. The important point to note is that these representations can be obtained by doing some additional work during the design process. A number of approaches we are working on currently for building 3-D models are: (a) universal representation of objects by surface points, (b) surface characterization by intrinsic properties such as the points of high curvature, (c) surface representation by edges/arcs, (d) higher level surface representation in terms of local properties such as holes, corners, surface type, etc.

In this paper we present approach (a). A new method is given which uses the computer aided geometric design and allows the points on the surface of an object to be sampled at the desired reso-

* Corresponding author; present address: Department of Computer Science, 3160 Merrill Engg. Building, University of Utah, Salt Lake City, UT 84112, USA.

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

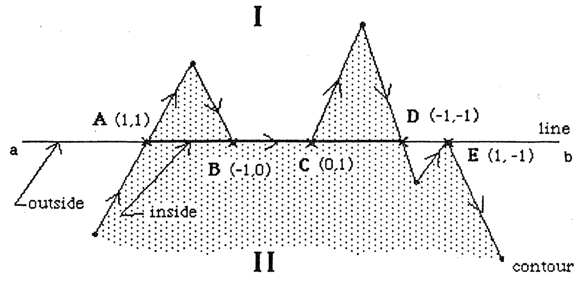


Figure 1. Intersection of a polygon by a line segment *ab*. The direction of edge going from region II to region I is 1, from region I to region II is −1 and for the edge lying on the line it is 0. At the intersection point, there are two directions: in direction and out direction. *A*, *B*, *C*, *D* and *E* are intersection points and they are labeled by the (in, out) directions. The arrows indicate the direction of the contour. The line *ab* need not be horizontal.

3.1. Algorithm for contour filling

The main element of the new contour filling algorithm in two dimensions is to find the intersection segments of a line and the polygon. They can be obtained in two steps. The first step is to find all intersection points and the next step is to decide which segments are inside the polygon using topological information associated with intersection points.

The intersection points of a line with a polygon can be divided into three classes: start point, end point and middle point. Suppose we trace a line *ab* (see Figure 1) from left to right inside a polygon. The start point can be defined as the point whose left neighbor is outside the polygon. The end point is the one whose right neighbor is outside the polygon and a middle point is the one for which both left and right neighbors are inside the polygon. In Figure 1, point *A* is a start point, points *B* and *C* are middle points and point *D* is an end point. Note that point *E* is both a start and an end point since its left and right neighbors are outside the polygon. Now what we have to do is to divide all the intersection points into these three classes and skip all middle points and finally label the start points and end points alternately.

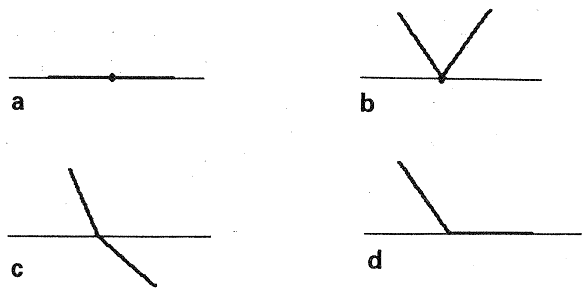


Figure 2. Four kinds of intersection points. By using the in/out direction at each intersection point, there are 9 different combinations of 4 kinds: (a) flat; (b) tangent; (c) cut; and (d) flat-cut.

In order to determine the kind of an intersection point, we need its topological information. It can be explained easily in the two-dimensional case. In Figure 1, the plane is divided by the line *ab* into two regions, I and II. Note that the line *ab* need not be horizontal in the general case. The arrows show the direction of the contour and the dotted region is the interior of the polygon. There are five intersection points (A, B, C, D and E) as shown in the figure. We find the directions, ‘in’ and ‘out’, of an edge (along the contour) when it passes through an intersection point. The ‘in’ and ‘out’ directions of an intersection point are defined as the incoming and outgoing directions at this point of the contour, respectively. 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 its direction as -1. And if the edge is lying on the line itself, we mark its direction 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 can be found easily at the time when we find the intersection points. There are nine different combinations of (in, out) directions. But they can be classified into four kinds only: flat, tangent, cut and flat-cut points as shown in Figure 2. For a flat point both of its in and out directions are characterized by 0. There is only one flat point: $(0, 0)$. For a tangent point, in $= -$ out $<> 0$. There are two possibilities: $(1, -1)$ and $(-1, 1)$. For a cut point, in $=$ out $<> 0$. Again there are two possibilities: $(1, 1)$ and $(-1, -1)$. For a flat-cut point only one of the in/out directions is 0. There are four flat-cut points: $(0, 1)$, $(0, -1)$, $(1, 0)$ and $(-1, 0)$. Now the problem is to map these four kinds of intersection points into three classes - start, middle and end points. For contour filling then we just skip all

```
Step 2. For each intersection point do:
          If the point is the first one or
          the previous one is an end point,
          then
            do Step 3.
          else
            do Step 4.
Step 3. Output this point. {start point}
        If it is a tangent point, {in = - out}
        then
          output it again. {end point}
        else
        begin
          if the in direction is 0,
            then exchange the in and out direction
            of this point. {the in direction of a
            start point should not be 0}
          Save the in direction of this start point.
        end
Step 4. If it is either a flat point or a tangent point,
        then
          skip it. {middle point}
        else
          do Step 5.
Step 5. If there is a conflict regarding the restric-
        tion as discussed in the above,
          then exchange the in and out direction of
          this point.
        If the out direction of the point is opposite
        to the in direction of the previous start
        point after the adjustment,
        then
          output this point. {end point}
        else
          skip it. {middle point}
```

Figure 4 shows a two-dimensional example. Figure 4(a) is the input contour. Each cross represents a vertex of the contour in the output of the combiner of Alpha_1. Some of these vertexes which appear to be redundant are also parts of other polygons of the same object. When considering a polygon they can be removed easily by finding the angle between two line segments joining the two neighboring vertexes. The direction of this contour is not significant for the use of the above algorithm. Figure 4(b) is the result of the above algorithm applied to Figure 4(a). Finally, in Figure 4(c), we extract all the surface points from the line segments by a user-defined resolution. The algorithm as outlined above is quite general for 2-D case. It can be implemented very efficiently.

The complexity of this algorithm is

$$O[(n + m * \log(m)],$$

where n is the number of vertexes in the input contour and m is the number of intersection points of one cutting line. Usually the number of intersection points is much smaller than the number of vertexes, and therefore, the complexity of the algorithm is linear. In the worst case, m is equal to $(n-1)$ and the complexity becomes $O[m * \log(m)]$. This is due to the sorting of the intersection points.

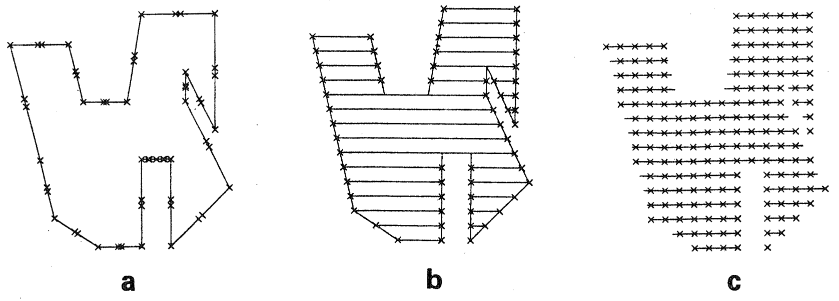


Figure 4. A two-dimensional example. (a) Input contour. The points marked as x's are the vertexes of this contour. (b) Interior line segments. The required points are the start and end points of these segments. (c) Surface points on the polygon. Points are extracted from the line segments in Figure 4(b).

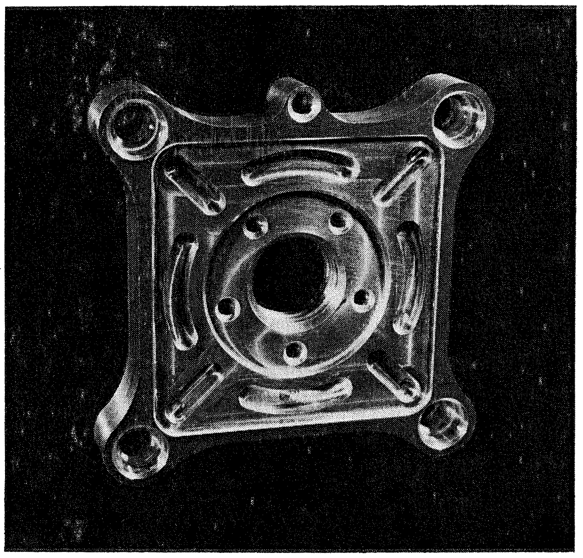
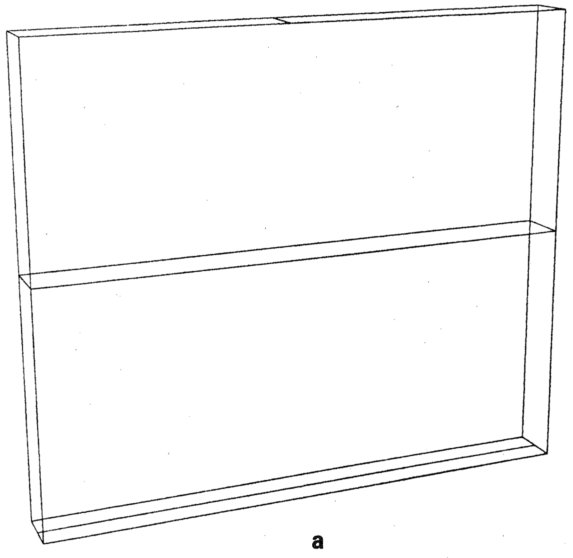


Figure 5. A picture of the workpiece to be designed.



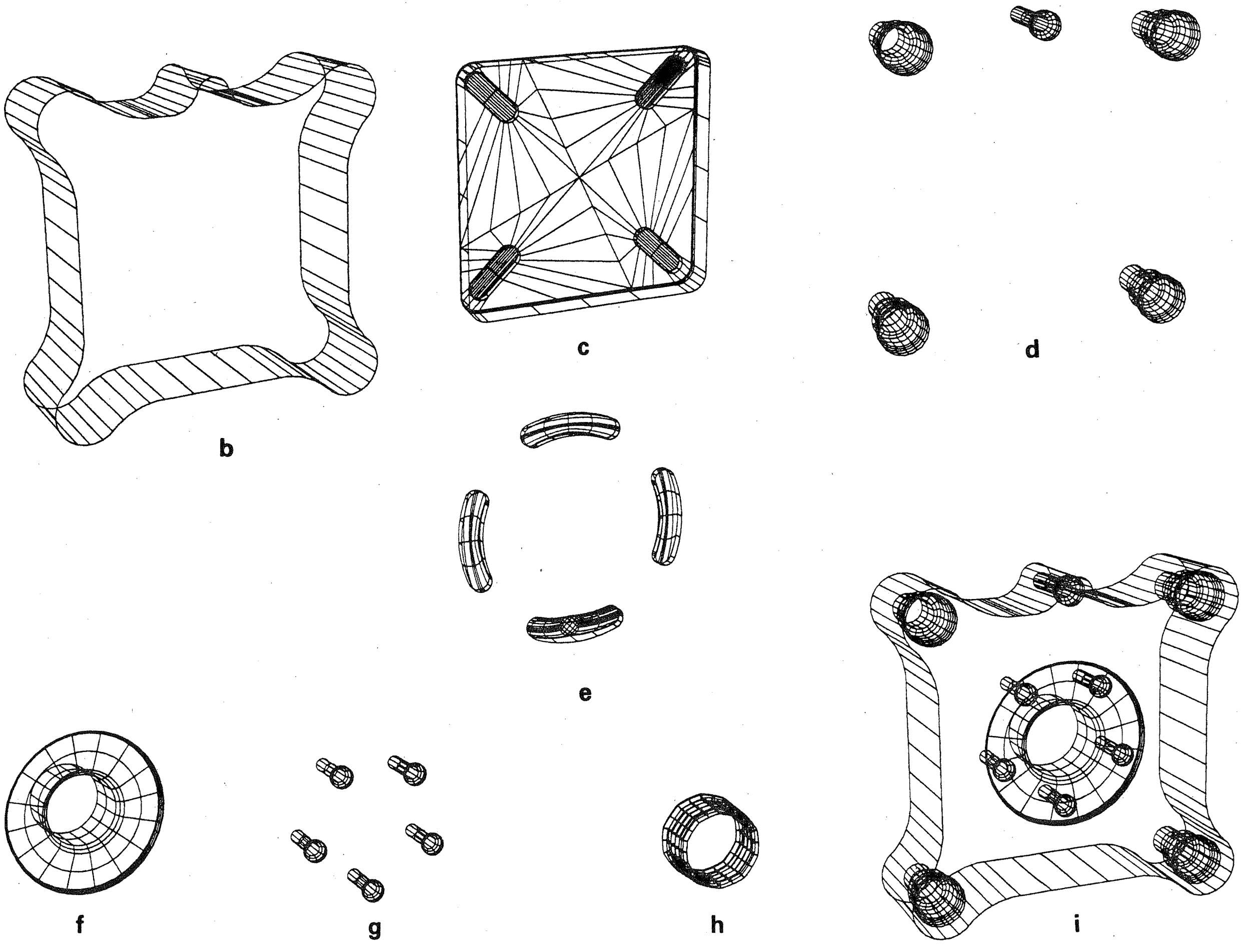


Figure 6. Subparts for designing the work piece. (a) Plate; (b) Outline cutter; (c) Outer dent part and the corner scratches; (d) Head hole and corner holes; (e) Four arc scratches; (f) Inner dent part and the center hole; (g) Five small holes; (h) Center thread; (i) The position of the subparts in the model of the work piece.

surface is different from other scratches. This is because they are actually different in the object (see Figure 5). Figure 8 shows a shaded image of the designed work piece. Next we use the contour filling algorithm described in Section 3 to find the surface points and their normals. First we convert parts of the representation which are still surfaces, in our CAGD model after the set operations, into polygons by subdivision [6]. Now we have three-dimensional polygons only. In extending the algorithm as described in the last section to 3-D planar polygons, several modifications are needed. The cutting line becomes a plane and according to our conventions, the direction of the contour is clockwise if the normal to the surface is directed towards the viewer. It is important to choose the appropriate cutting plane and to choose points which are within the desired resolution. Note that we cannot always cut in the $x-y$ plane because we will not get any point on the surface which is parallel to the z axis. The approach used here is to first find the bounding box for the polygon. A bounding box of a polygon is specified by the minimum and maximum x, y, z coordinates of all its vertexes. Next we choose the two directions in which the bounding box has large spreads. These two directions specify the cutting plane and are taken to be the directions along which the lines and points are found respectively. Assuming uniform spacing d in x, y, and z spatial coordinates, for a given resolution r such that there is at least one point in the sphere of radius r, we find the spacing d by dividing r by $3^{1/2}$. Figure 9 shows the surface points with 0.2 inch resolution. In Figure 10, we show the surface point normals with 0.4 inch resolution. The output polygons of the combiner contain normals of every vertex which are computed from the surface. We interpolate vertexes to find intersection points. The normals at intersection points are found by linear interpolation of normals at adjacent vertexes. Linear interpolation procedure is again used to find normals at surface points along a line segment of the polygon. Points and their normals are computed at the same time.

5. Conclusions

In this paper we have presented a technique for the representation of 3-D objects by surface points. The resulting data structure of points including coordinates of the points in 3-D space, surface normals and information about the neighboring points. This representation is the lowest-level one. However, many higher-level features to be used in matching can be extracted from it by using the same procedures as used on the sensed range data [1, 7].

References

[1] Bhanu, B. Representation and shape matching of 3-D objects. *IEEE Trans. Patt. Anal. Mach. Intell.* 6(3) (1984) 340–351.

[2] Bhanu B. and T. Henderson. CAGD-Based 3-D vision. In: *IEEE International Conference on Robotics and Automation*, 1985, pp. 411–417.

[3] Bhanu, B., C.C. Ho and T. Henderson. 3-D model building for computer vision. Technical Report UUCS85-112, Department of Computer Science, University of Utah, September, 1985.

[4] Cobb, E.S., Design of sculptured surfaces using the B-spline representation. Ph.D. thesis, University of Utah, June, 1984.

[5] Cohen, E. Some mathematical tools for a modeler's workbench. *IEEE Comput. Graphics Appl.* October (1983) 63–66.

[6] Cohen, E., 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) (1980) 87–111.

[7] Henderson, T.C., Efficient 3-D object representations for industrial vision systems. *IEEE Trans. Patt. Anal. Mach. Intell.* 5(6) (1983) 609–618.

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

[9] Potmesil, M. Generating models of solid objects by matching 3-D surface segments. In: *Proc. 8th IJCAI*. Karlsruhe, August, 1983, pp. 1089–1093.

[10] Thomas, S.W. Modelling volumes bounded by B-spline surfaces. Ph.D. thesis, University of Utah, June, 1984.