1-1-2004

Application of doo subdivision for planning roughing cuts in five axis machining

Abhishek Saboo
Iowa State University

Application of doo subdivision for planning roughing cuts in five axis machining

by

Abhishek Saboo

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Industrial Engineering

Program of Study Committee:
Ranga Narayanaswami, Major Professor
Carolyn Heising
Palaniappa Molian

Iowa State University

Ames, Iowa

2004

Graduate College

Iowa State University

This is to certify that the masters thesis of

Abhishek Saboo

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to sincerely thank, Dr. Ranga Narayanaswami, for his invaluable guidance. I am grateful to him for encouraging me and providing me support throughout.

I would also like to thank my committee members Dr. Carolyn Heising and Dr. Molian for being on my committee and give me much appreciated guidance.

Finally, I would like to thank my parents and all my friends for helping me and giving moral support.

ABSTRACT

Wavelets permit multiresolution analysis of curves and surfaces. A complex curve can be decomposed using wavelet theory into lower resolution curves. The main Idea of Subdivision is "Subdivision defines a smooth surface as a limit of a sequence of successive refinements". Subdivision methods are quite useful for generating surface in computer graphics. A fine approximation of the model is obtained by successfully subdividing the vertices of the coarse approximation.

In this research we have worked on a novel idea of application of Doo subdivision in machining of surfaces using Multi axis Machine. Here we have first generated subdivision surfaces using Doo subdivision followed by the sequence of operations required for roughing.

We have data, which suggests using this approach on multi axis machine; reduce machining time in comparison with three axis NC machine. Along with that will also generate better quality-roughed surface, which will help to reduce machining time for finishing too.

CHAPTER 1. INTRODUCTION

One of the important aspects of Machining is reducing machine time and improving the quality. In the rough-cut stage, the main goal is to remove material in the most efficient manner. In finish cutting, producing the desired surface finish and accuracy is the primary driving factor. Till recently the industry relied a lot on 3-axis machine for machining of surfaces. Significant amount of research and development has been done in the field, which has led to development of better NC programming methods and machining tools. Even then machining them on 3-axis machine using special tools and multiple setups compromises the practical limits of accuracy and cost for complex parts. The main Objective of this research is to develop a process to reduce roughing time, improving the surface quality using five axis machines.

1.1) Three axis Machine

Three axis mills are available in horizontal and vertical spindles configurations. A vertical spindle machine is the choice of die-sinking and other operation requiring work on one face of the part only. Fixturing is usually rudimentary in vertical machine spindle work, since the part can be clamped to the worktable or held in suitable milling vise. Chip accumulation, however, is an inconvenience.

For production work, or work requiring operations on more than one face of the part, as is necessary in the manufacture of fluid power pumps, valves, and manifolds, am horizontal

spindle is preferred. In this chip accumulation is a lesser problem. Both horizontal and vertical spindle machine are available as positioning or contouring machines .The horizontal spindle machine configuration is well adapted to receive additional equipment to extend its versatility. Automatic tool changers with tool magazine, rotating worktables may be added.

There are three types of positioning control in Three-axis machine:

When classified according to the machine tool control system, there are three basic types of NC systems:

1.1.1. Point to Point.

1.1.2. Straight cut.

1.1.3. Contouring.

The classification is concerned with the amount of control over the relative motion between the work piece and cutting tool. The least control is exerted over the tool motion with the point-to-point systems. Contouring represents the highest level of control.

1.1.1) Point-to-Point NC positioning control

The principal function of the point-to-point positioning control is to position the tool from one point to another within a coordinate system; therefore the control is most often referred to as point-to-point NC system. In PTP the objective of the machine tool control system is to move the cutting tool to predefined location. The speed or path by which this movement is accomplished is not important in point to point NC. Once the tool reaches the desired location, the machining operation is performed at that position. NC drill presses are a good example of PTP systems.

The spindle must first be positioned at a particular location at the work piece. This is done under PTP control. Then the drilling of the holes is performed at that location, the tool is moved to the next whole location, and so forth. Since no cutting is performed between holes there is no need for controlling the relative motion of the tool and work piece between hole locations. On positioning systems the speeds and feeds used by the machine tool are often used by the machine operator rather than by the nc tape. Positioning systems are the simplest machine tool control systems and therefore the least expensive of the three types. However for certain operations such as drilling and spot welding, point-to-point is perfectly suited to task and any higher level of control is unnecessary.

1.1.2) Straight Cut NC positioning control

Straight cut positioning systems provide a limited degree of control during the positioning of the tool from one point to other. Most of the straight cut systems are fitted with manually adjusted feed control. All the programmable axes of the NC machine, which allows the system to perform milling, in addition to drilling, share this feed control. Straight cut control systems are capable of moving the cutting tool parallel to one of the major axes at a controlled rate suitable for machining. It is therefore appropriate for performing milling operations to fabricate work pieces of rectangular configurations. With this type of NC systems it is therefore appropriate for performing milling operations to fabricate work pieces of rectangular configurations. With this type of NC system it is not possible to combine movements in more than single axis direction. Therefore angular cuts on the work piece would not be possible. An NC machine tool capable of performing straight cut movements is also capable of point-to-point movements.

1.1.3) Contouring NC positioning control

The contouring system is the state of art, high technology most versatile and intricate of the CNC devices. It generates a continuously controlled tool path, by interpolating intermediate points or coordinates. Interpolating here means the capability of computing the points of the path. Contouring is the most complex flexible and the most expensive type of machine tool control. It is capable of performing both PTP and straight cut operations. In addition the distinguishing feature of the of contouring NC system is their capacity for simultaneous control of more than one axis movement of machine tool Figure 1.1 below illustrate the versatility of continuous path NC. Milling and Turning are the common examples of the use of contouring control.

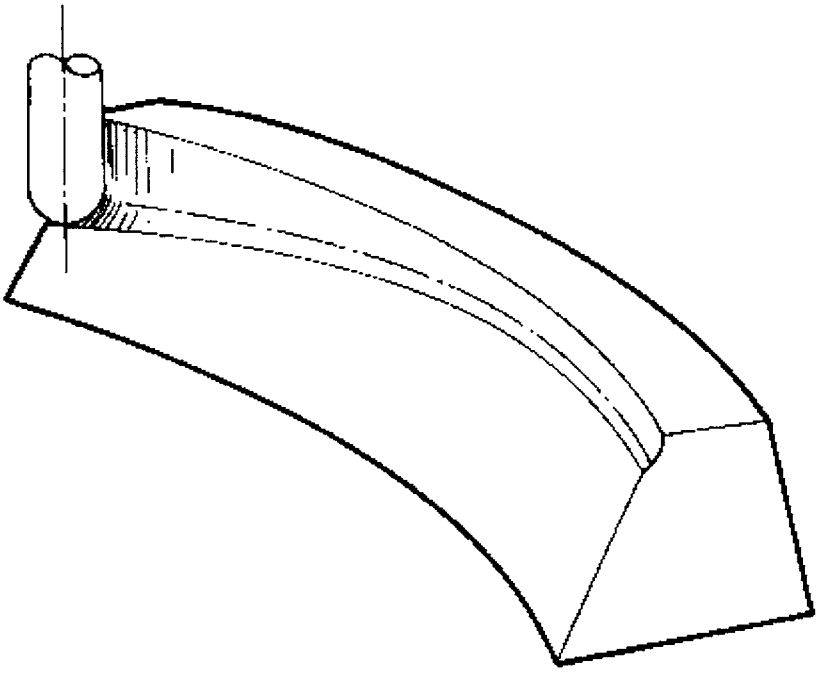


Figure 1.1 Continuous path NC

1.2) Five Axis NC Machining

The need for reducing machining time, production cost and improving machining quality led to the development of five axis machines. To achieve the intended performance of a product (e.g., an impeller or turbine blade) or capture its desired aesthetics (e.g., a car fender or hood), engineers and designers continually seek new methods and techniques for specifying part geometry. This has resulted in the development and use of entirely new classes of surfaces.

Traditionally, the tool paths required to machine complex surfaces have been constructed using numerical control languages or CAD systems. These languages or systems generate numerical instructions for multi-axis machine tools based on defined part geometry and specified machining parameters. The numerical instructions are generated by first reducing the dimensionality of the surface to that of planar or isoparametric curves.

Milling machines used to machine complex surfaces have three axes of simultaneous tool positioning control and zero, one or two axes of simultaneous tool orientation control. While 3-axis milling machines (those with no tool orientation capabilities) have historically been the choice to machine complex surfaced parts, five axis-milling machines have increasingly been used in industry. As compared to 3-axis machining, five axis machining (a simple example is shown in Figure 1.2) offers many advantages such as higher productivity and better machining quality. In five axis machining, the orientation of the tool can be determined by the two additional degrees of freedom so as to obtain efficient tool paths. Figure 1.3 shows the tool orientation control in five axis machining. The angle of the tool in the plane of

motion is called the tool inclination angle α, and the angle of tool out of the plane of motion is called the tool tilt angle β.

Figure 1.2 Five axis machining example

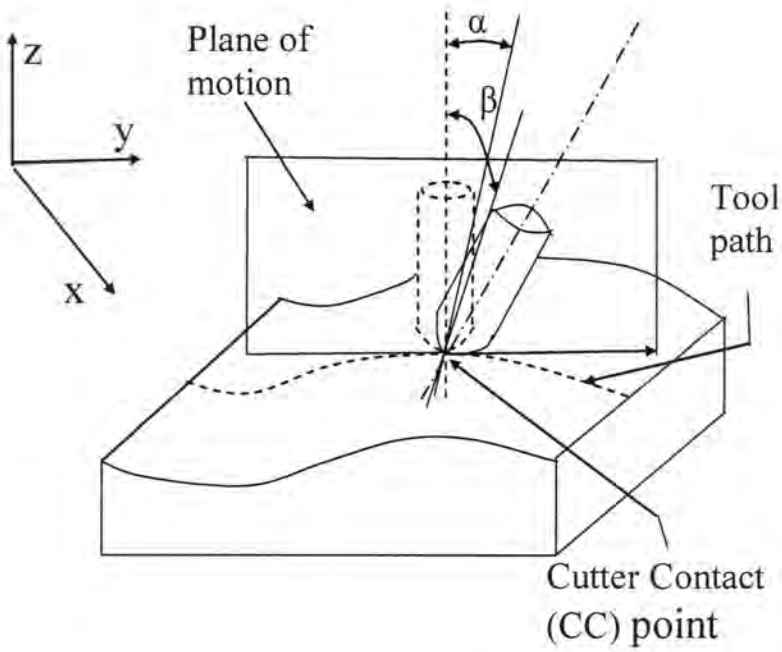


Figure 1.3 Tool orientation in five axis machining.

Wavelets can provide a single theoretical framework for performing most of the geometric reasoning needed for efficient tool path generation in five axis NC machining. The application of wavelets can also minimize the need for these geometric algorithms and improve the computational efficiency when these geometric algorithms are needed. Most important, wavelet based multiresolution representations generate a series of intermediate shape models as the references of NC programming, in which the tool cutting ability and removal volume can be automatically considered, the amount of model data can be compressed, and the resolution of the models can be changed. In addition, multiresolution models represented by polygonal meshed surfaces are becoming more popular in CAD/CAM. However, there is almost no published literature about tool path generation directly from a multiresolution meshed representation. The wavelet representation is very conducive for tool path generation based on a meshed surface.

Wavelets and multiresolution analysis may be used to hierarchically represent the original surface. Each hierarchical model successively approximates the final object shape. Thus rough-cuts can be made with the low-resolution models and finish cuts can be made with the high-resolution models. Wavelets may also be used to develop new multiresolution accessibility analysis algorithms for efficient collision detection and avoidance.

In this research we demonstrate the application of Doo Subdivision based on the theory of wavelets in machining. This is a novel application and provides added insight and a rich theoretical framework for some new process for machining. The appearance of wavelets is a relatively recent development in mathematics. They give rise to hierarchical representation

and have been successfully used for various tasks such as approximation theory, signal processing, image storage and compression, and content based retrieval. Recently the theory of wavelets and multiresolution analysis has also been applied to hierarchical editing of curves and surfaces in computer graphics.

The main Idea of Subdivision is "Subdivision defines a smooth surface as a limit of a sequence of successive refinements". Subdivision methods are quite useful for generating surface in computer graphics. A fine approximation of the model is obtained by successfully subdividing the vertices of the coarse approximation.

The rest of the report is organized as follows. The theory of multiresolution analysis and wavelets is briefly reviewed along with Chaikens and Doo Subdivision in chapter 2. In chapter 3, how did we approached to this problem has been discussed. It basically involves getting the shape of object using Doo Subdivision, Determining the sequence of operation; machining time simulation. In chapter 4, Results along with conclusion is presented, followed is the proposed future work along with references and appendix.

CHAPTER 2. WAVELETS AND MULTIRESOLUTION ANALYSIS

This chapter provides some background on wavelets and multiresolution analysis. Multiresolution analysis is a simple mathematical tool that has found a wide variety of applications in recent years, including signal analysis (Mallet, 1989), image processing (DeVore *et al.*, 1992), and numerical analysis (Beylkin *et al.*, 1991). Informally, wavelets are the basis functions for multiresolution analysis.

A signal or a function may be better understood if expressed as a linear decomposition over a basis.

$$f(t) = \sum_l a_l \phi_l(t) \tag{2.1}$$

The basis functions are usually chosen to be orthogonal. For the Fourier series these basis functions are $\sin(k\omega_0 t)$ and $\cos(k\omega_0 t)$. On the other hand, the wavelet expansion is a two-parameter system described by

$$f(t) = \sum_k \sum_j a_{j,k} \psi_{j,k}(t) \tag{2.2}$$

The coefficients $a_{j,k}$ are called as the discrete wavelet transform. The wavelet expansion gives a time frequency localization of the signal. A wavelet representation is much like a musical score where the location of the notes tells when the tones occur and what their frequencies are. Wavelet systems are generated from a single scaling function by simple scaling and translation. Therefore, if a set of functions is represented by a weighted-sum of $\psi(t-k)$ then a larger set of functions may be represented by $\psi(2t-k)$. Wavelets thus satisfy

the multiresolution conditions. The lower resolution coefficients can be calculated from the higher resolution coefficients, using a tree-structured algorithm called a *filter bank.*

Consider a discrete signal C^n, expressed as a column vector of samples $\left[c_1^n, ..., c_m^n\right]^T$. In our curve application described in Chapter 3, for example, the samples c_i^n will be the curve's control points in $\Re^2$.

Suppose we wish to create a low-resolution version C^{n-1} of C^n with a fewer number of samples m', the standard approach for creating the m' samples of C^{n-1} is to use some form of linear filtering and sub sampling on the m samples of C^n. This process can be expressed as a matrix equation

$$C^{n-1} = A^n C^n \tag{2.3}$$

Where A^n is an m' x m matrix.

Since C^{n-1} contains fewer samples than C^n, it is intuitively clear that some amount of details is lost in this filtering process. If A^n is chosen appropriately, it is possible to capture the lost detail as another signal D^{n-1} with m - m' samples, computed by

$$D^{n-1} = B^n C^n \tag{2.4}$$

Where B^n is an (m - m') x m matrix, which is related to matrix A^n. The pair of matrices A^n and B^n are called *analysis filters.* The process of splitting a signal C^n into a low-resolution

version C^{n-1} and detail D^{n-1} is called *decomposition*. Usually m is roughly half of m, so that C^{n-1} and D^{n-1} are roughly equal in size.

Note that C^{n-1} and D^{n-1} together have the same amount of information as C^n. If A^n and B^n are chosen correctly, then the original signal C^n can be recovered from C^{n-1} and D^{n-1} by using another pair of matrices P^n and Q^n, called *synthesis filters*, as follows:

$$C^n = P^n C^{n-1} + Q^n D^{n-1} \tag{2.5}$$

Recovering C^n from C^{n-1} and D^{n-1} is called *reconstruction.*

The procedure for splitting C^n into a low-resolution part C^{n-1} and a detail part D^{n-1} can be applied recursively to the new signal C^{n-1}. Thus, the original signal can be expressed as a hierarchy of low-resolution signals $C^0, \ldots, C^{n-1}$ and details $D^0, \ldots, D^{n-1}$, as shown in Figure 2.1. This recursive process is known as a *filter bank.*

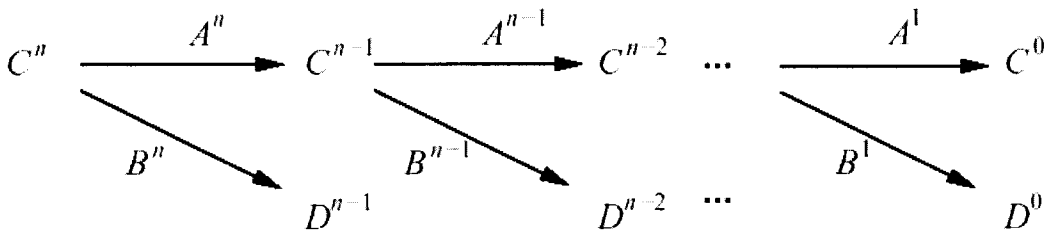


Figure 2.1. The filter bank

Since the original signal C^n can be recovered from the sequence $C^0, D^0, D^1, \ldots, D^{n-1}$, this sequence can be thought of as a transform of the original signal known as a *wavelet transform*. Note that the total size of the transform $C^0, D^0, \ldots, D^{n-1}$ is the same as that of the original signal C^n, so no extra storage is required.

2.1) Chaikin's Subdivision

Since Doo subdivision may be explained as Chaikin's subdivision rule for curves extended to surfaces, here first explained is geometrically how to determine the Chaikin local reverse subdivision rule. Chaikin's subdivision is a "corner-cutting" strategy. Figure 2.2 shows one step of this subdivision as a two-step process (for simplicity, we use *vi* to stand for *vi k* and *wj* for *v j k)*

Two new points are first determined on every line segment, and then the second new point on each successive line segment is connected with the first new point of the next segment. It can be shown that the points generated by repeatedly applying Chaikin's subdivision rule

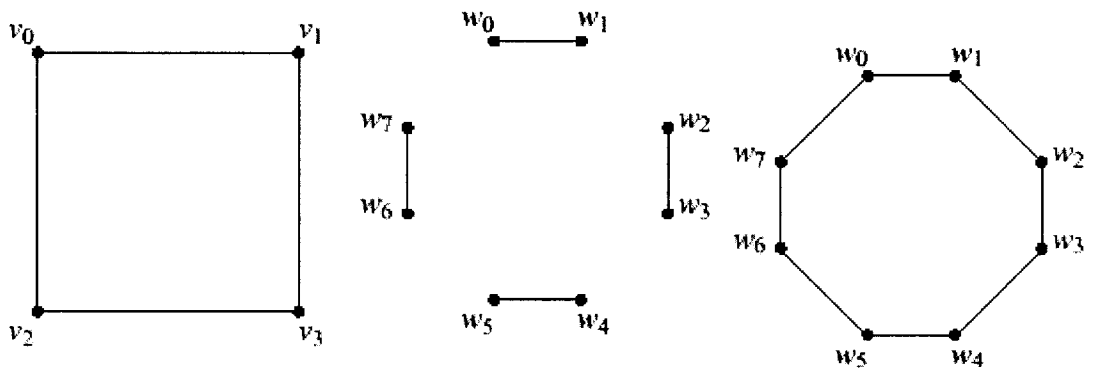


Figure 2.2 A two stage representation of Chaikin's Rule

2.2) Doo Subdivision

Doo subdivision is a "corner-cutting" method, for the representation of three-dimensional surfaces. In this method, the shapes of the faces of the surface are assumed to be arbitrary (consequently, if a face is not triangular, it may not be planar). It can be shown that the Doo subdivision scheme is an extension of Chaikin's method. To show this, consider Chaikin's method in two stages as shown in Figure 2.2.

In the first stage, contract each line segment of the polygon towards its center by a factor α .for example in Figure.2.2, (v_0, v1) is contracted into (w_0, w1), where $w_0=\alpha d+(1-\alpha)v_0$, $d=(v_0+v_1)/2$, and so on for the other points. In the second stage for those line segments having been joined together before the subdivision, introduce a new line segment by joining the corresponding end points of their contracted line segments. Again in figure 2.2,the line segment (w_0, w_1) is joined with (w_2, w_3) through the new line segment (w_0, w_1). The multiplying factor α should be chosen such that the limiting curve should be C^1 curve.

For the Doo Subdivision of surfaces, the operations above on the line segment of a polygon are extended to the faces of the polyhedron: the faces are contracted towards their centroids and the contracted versions of adjoining faces are joined by introducing additional new faces for example in the figure 2.3

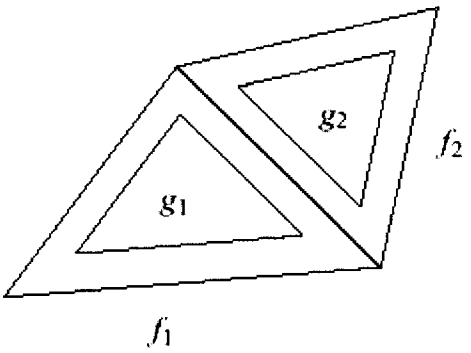


Figure 2.3 A simple doo subdivision

The original surface is composed of the two adjoining faces f_1 and f_2. In the next stage, g1 and g_2 are joined by g_3 to observe the effect on a three dimensional figure (cube), where Doo subdivision is applied on a cube.

To explain more completely, assume that (F^k, V^k) is a surface of arbitrary topological type, being represented by Faces F^k and vertices V^k. Each vertex $v \in V^k$ is identified by its coordinates in space and each face by the sequence of its associated vertices. Let E be the set of existing edges (F^k, V^k). Now is the explanation, how to identify a subdivided surface (F^{k+1}, V^{k+1}) as a result of application of Doo subdivision scheme on (F^k, V^k). First, for each $f_i \in F^k$ introduce a new face $g_i \in F^{k+1}$ by contraction, as in figure 2.3 in the following way: If $f_i=(v_1,\ldots\ldots,v_n)$ is the faces in original faces and $g_i=(w_1,\ldots\ldots,w_n)$ is the contracted faces, the w_μ come from contracting v_λ. With respect to the centroid of f_i as follows (for any reasonable, systematic indexing system for the f,g,v, and w):

$$w_\mu = (1-\alpha)d + \alpha v_\lambda.$$

$$d = 1/n(v_1+v_2+\ldots\ldots+v_n)$$

In the above equation the vale of α, $0<\alpha<1$, can be chosen so that the limiting surface will be C^1. After this operation, the set of w_μ obtained by the contraction of all faces, forms the new set V^{K+1}.Note that each $v \in V^k$ corresponds to deg (v)(the degree of v: that is, the number of faces joined at v) new vertices introduced in V^{k+1}(Fig 2.4).

The set of new faces in F^{K+1} is composed of the following three types.

1) Face-to-face: The new set of faces, g_i is obtained from F^k, denoted by F_F.
2) Face-to-edge: If f_i is joined with fj through an edge e in the graph (F^k, V^k), then we have face g_e joining g_i and g_j. Therefore for each edge not lying on the boundary, a new face is introduced denoted by F_E. Each face in F_E is rectangular.
3) Face-to-vertex: For each internal vertex v in (F^K, V^K) a new face g_v is introduced, so that it joins the corresponding new vertices w_j of v. Denoted by Fv. Number of sides for each force in F_v is equal to the degree of its corresponding vertex in V^k.

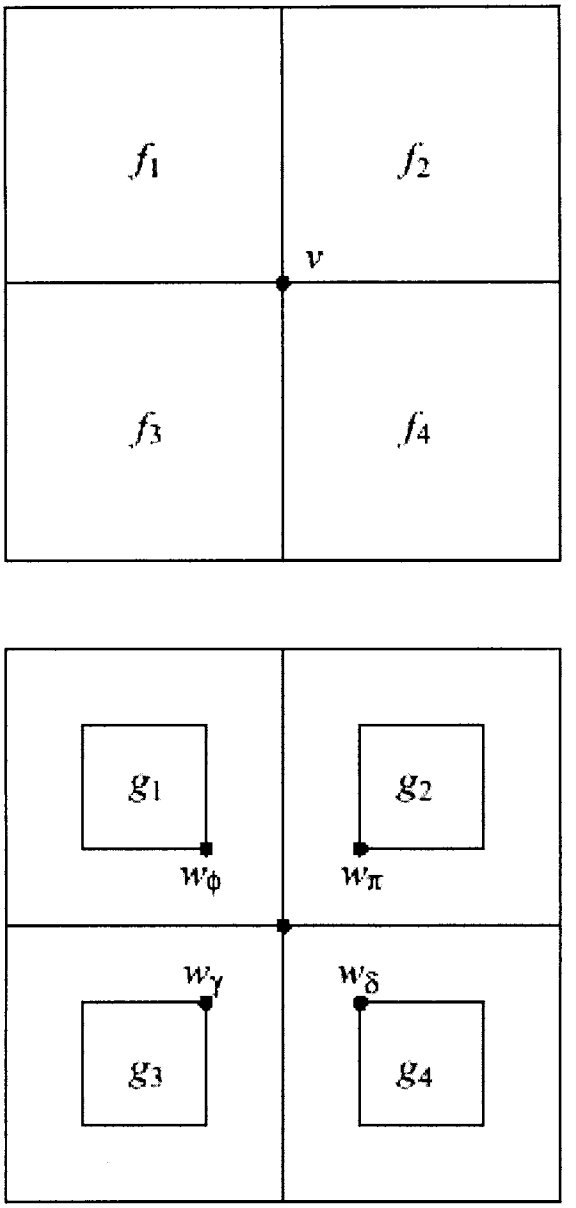


Figure 2.4 Conversion of v to w_Φ, w_Π, w_γ, w_δ

It can be easily verified that

- Each face in F_F has the same number of edges as the face from which it is contracted.
- Each face in F_E is rectangular and
- The number of sides for each face in F_v is equal to the degree of its corresponding vertex in V^k.

Determining these three types of faces, the subdivision is complete. Therefore we have

$$F^{k+1}=F_F+F_E+F_V.$$

CHAPTER 3. APPLICATION OF DOO SUBDIVISION IN COMPUTER AIDED DESIGN AND MANUFACTURING

Lot of work has been done till date using idea of subdivision in curves and surfaces in the field of computer graphics, here we are trying to develop an innovative idea of application of subdivision in Manufacturing. In this section application of Doo subdivision in computer aided design and manufacturing has been explained. It's a new idea so no literature was available for this.

The main Idea of Subdivision is "Subdivision defines a smooth surface as a limit of a sequence of successive refinements". We are using the idea of subdivision such that, we start from initial stock and then by using doo subdivision once, we get the shape we should have using multi axis machine and similar repetitions leads us to the final product.

Subdivision methods are quite useful for generating surface in computer graphics. A fine approximation of the model is obtained by successfully subdividing the vertices of the coarse approximation.

In this research we have approached this application basically in three steps:

3.1) Getting the shape of object using Doo Subdivision

3.2) Determining the sequence of operation using mat lab.

3.3) Machining time simulation.

3.1) Getting the shape of object using Doo Subdivision:

In this step we apply doo subdivision and get the shape of the figure we are going to manufacture. Working on mat lab does this; basically we apply doo subdivision to look the shape of the figure. Here we have some figures, which we get after applying Doo Subdivision. Initially some coordinate points have been given for the cube as input parameters. The codes used to get these shapes have been attached in the appendix.

Followed are the different surfaces after application of Doo Subdivision:

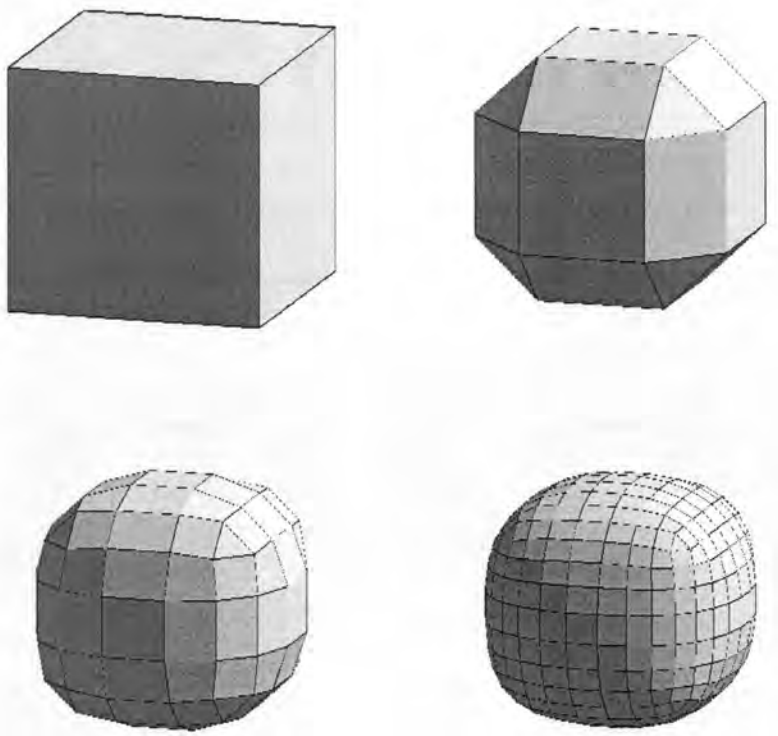


Figure 3.1 Doo subdivision on cube

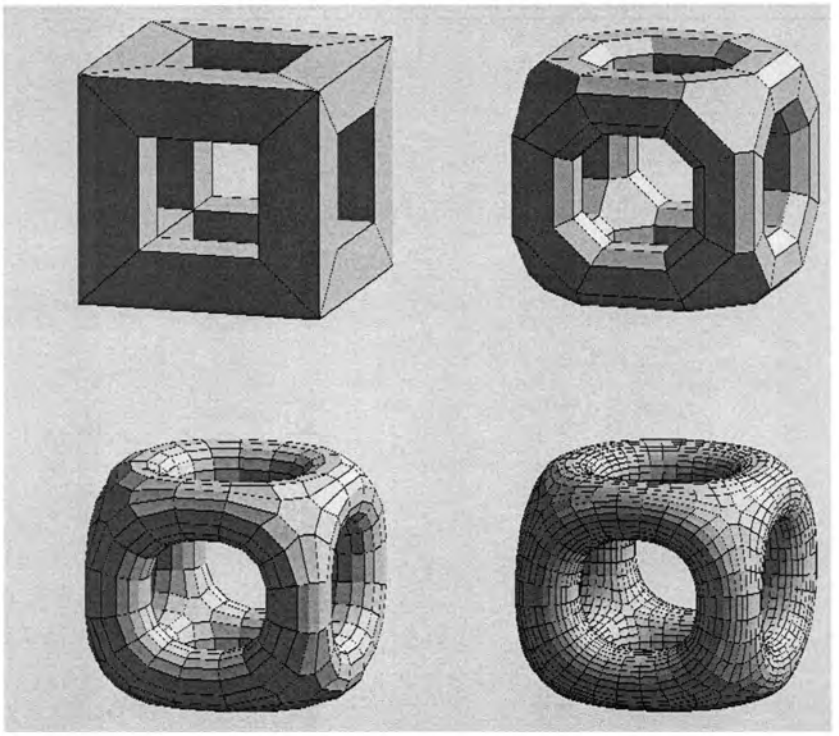


Figure 3.2 Doo subdivision on menger

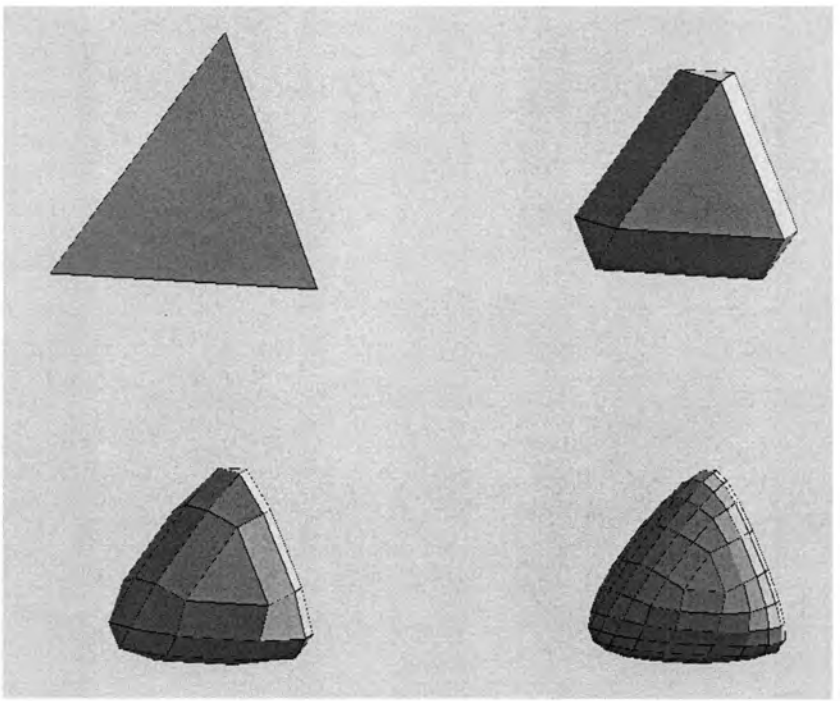


Figure 3.3 Doo subdivision on tetra

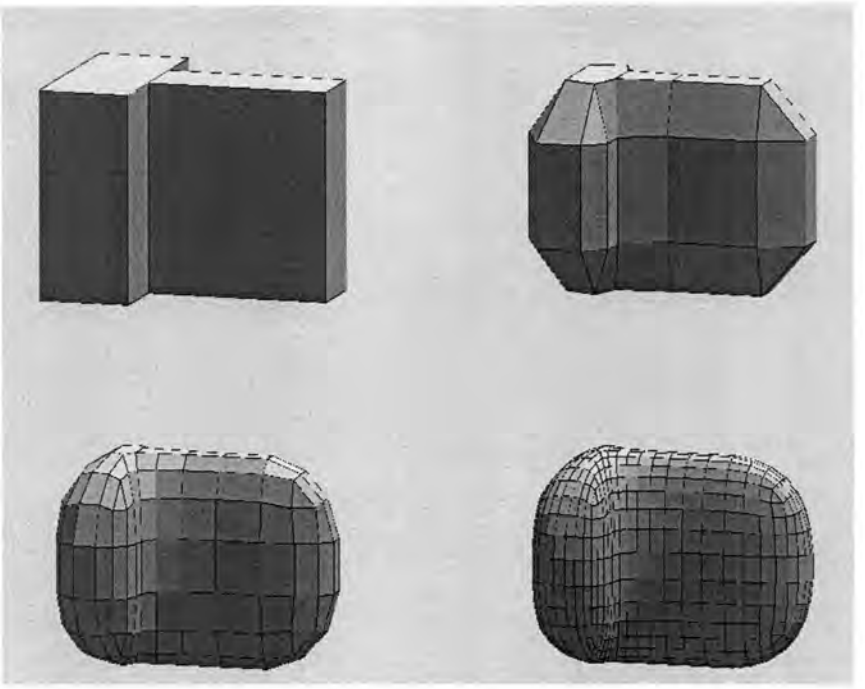


Figure 3.4 Doo subdivision on tetris

3.2) Determining the sequence of operation using mat lab:

Doo Subdivision leads to different steps, after applying subdivision for the first time the result lead us to the surface which is needed after first roughing operations. Here for example in the case for convex figure like cube we start with a cube and after first roughing operation using multi axis machine we want to have it same as cube will look after application of doo subdivision once. So here we can see basically doo subdivision is leading us to the shape, which we want after each roughing. But in order to determine the right sequence to get the desired shape we have generated Code to look at shape of the original cube exactly after each cut. The main motive behind this is to give rough cuts in a sequence of operations to get the required convex figure with least number of set ups. Similarly second roughing operation is

done to get the following figure. We have done this work by analyzing movement of cutting plane and original plane in mat lab, which can be seen in the figure (3.5 and 3.6)

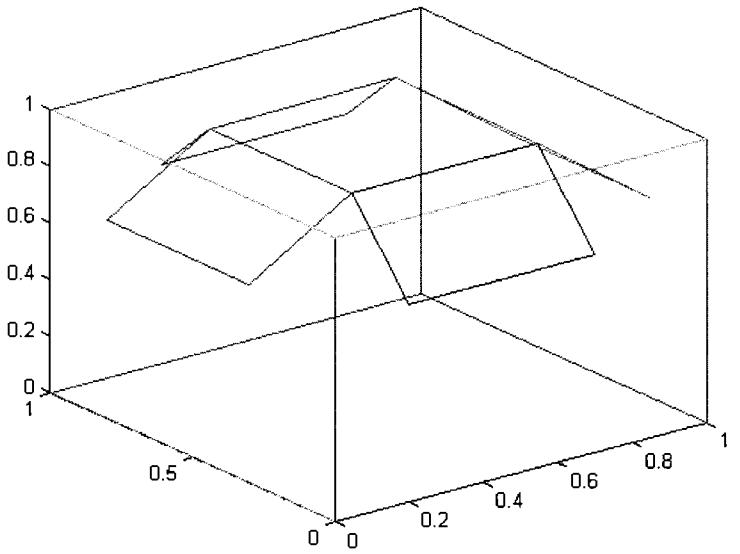


Figure 3.5 Sequence of operations

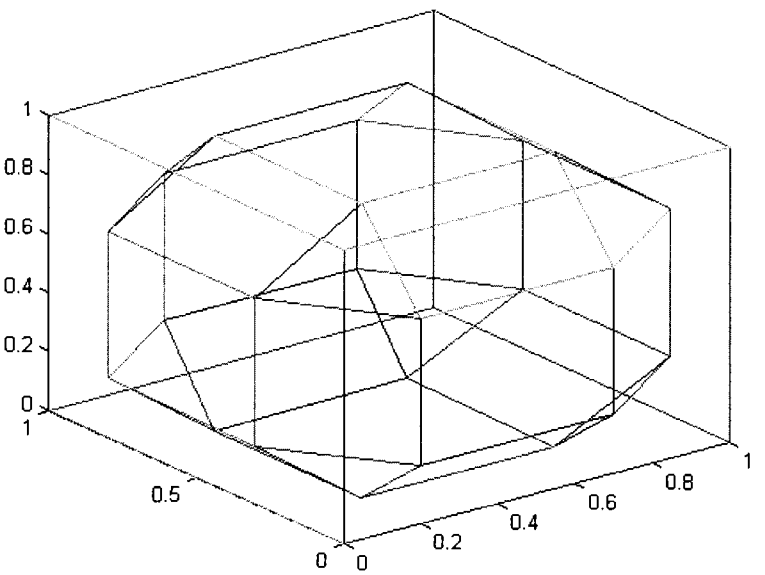


Figure 3.6 Step I after machining

Here is the code used for generating the step1 and by using this code with different plot commands one can see how the figure look as shown in fig

```
x = [0 1 1 0 0 0 0 ;1 1 0 0 1 1 1 ; 1 1 0 0 1 1 1 ; 0 1 1 0 0 0 0;0 1 1 0
                                    0 0 0];
y = [0 0 1 1 0 0 0; 0 1 1 0 0 0 0; 0 1 1 0 1 1 0; 0 0 1 1 1 1 0;0 0 1 1 0
                                      0 0];
z = [ 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 1 0; 1 1 1 1 0 1 1; 1 1 1 1 0 1 1;0 0 0 0 0
                                      1 0];

                          for i = 1:1:7
                         temp(1) = 0.0;
                         temp(3) = 0.0;
                         temp(2) = 0.0;
                          for j = 1:1:5
                  temp(1) = (temp(1) + x(j,i));
                  temp(2) = (temp(2) + y(j,i));
                  temp(3) = (temp(3) + z(j,i));
                             end
                       d(i,1) = temp(1)/5;
                       d(i,2) = temp(2)/5;
                       d(i,3) = temp(3)/5;

                             end
                         for i = 1:7
                         for j = 1:5
                  x(j,i) = d(i,1)/2 + x(j,i)/2;
                  y(j,i) = d(i,2)/2 + y(j,i)/2;
                  z(j,i) = d(i,3)/2 + z(j,i)/2;

                             end
                             end

                 X=[x(2,1),x(2,5),x(1,2),x(2,1)];
                 Y=[y(2,1),y(2,5),y(1,2),y(2,1)];
                 Z=[z(2,1),z(2,5),z(1,2),z(2,1)];
                X1=[x(4,5),x(2,3),x(1,4),x(4,5)];
                Y1=[y(4,5),y(2,3),y(1,4),y(4,5)];
                Z1=[z(4,5),z(2,3),z(1,4),z(4,5)];
                X2=[x(1,5),x(2,4),x(1,1),x(1,5)];
                Y2=[y(1,5),y(2,4),y(1,1),y(1,5)];
                Z2=[z(1,5),z(2,4),z(1,1),z(1,5)];
                X3=[x(3,2),x(3,6),x(4,3),x(3,2)];
                Y3=[y(3,2),y(3,6),y(4,3),y(3,2)];
                Z3=[z(3,2),z(3,6),z(4,3),z(3,2)];
                X4=[x(4,2),x(2,6),x(3,1),x(4,2)];
                Y4=[y(4,2),y(2,6),y(3,1),y(4,2)];
                Z4=[z(4,2),z(2,6),z(3,1),z(4,2)];
                X5=[x(4,4),x(4,6),x(3,3),x(4,4)];
                Y5=[y(4,4),y(4,6),y(3,3),y(4,4)];
                Z5=[z(4,4),z(4,6),z(3,3),z(4,4)];
                X6=[x(3,4),x(1,6),x(4,1),x(3,4)];
```

```
Y6=[y(3,4),y(1,6),y(4,1),y(3,4)];
Z6=[z(3,4),z(1,6),z(4,1),z(3,4)];
X7=[x(3,5),x(2,2),x(1,3),x(3,5)];
Y7=[y(3,5),y(2,2),y(1,3),y(3,5)];
Z7=[z(3,5),z(2,2),z(1,3),z(3,5)];
x1=[x(3,5),x(1,3),x(2,3),x(4,5),x(3,5)];
y1=[y(3,5),y(1,3),y(2,3),y(4,5),y(3,5)];
z1=[z(3,5),z(1,3),z(2,3),z(4,5),z(3,5)];
x2=[x(1,2),x(4,2),x(3,1),x(2,1),x(1,2)];
y2=[y(1,2),y(4,2),y(3,1),y(2,1),y(1,2)];
z2=[z(1,2),z(4,2),z(3,1),z(2,1),z(1,2)];
x3=[x(2,5),x(1,5),x(1,1),x(2,1),x(2,5)];
y3=[y(2,5),y(1,5),y(1,1),y(2,1),y(2,5)];
z3=[z(2,5),z(1,5),z(1,1),z(2,1),z(2,5)];
x4=[x(2,3),x(3,3),x(4,4),x(1,4),x(2,3)];
y4=[y(2,3),y(3,3),y(4,4),y(1,4),y(2,3)];
z4=[z(2,3),z(3,3),z(4,4),z(1,4),z(2,3)];
x5=[x(2,4),x(3,4),x(4,1),x(1,1),x(2,4)];
y5=[y(2,4),y(3,4),y(4,1),y(1,1),y(2,4)];
z5=[z(2,4),z(3,4),z(4,1),z(1,1),z(2,4)];
x6=[x(4,1),x(3,1),x(2,6),x(1,6),x(4,1)];
y6=[y(4,1),y(3,1),y(2,6),y(1,6),y(4,1)];
z6=[z(4,1),z(3,1),z(2,6),z(1,6),z(4,1)];
x7=[x(4,6),x(3,6),x(4,3),x(3,3),x(4,6)];
y7=[y(4,6),y(3,6),y(4,3),y(3,3),y(4,6)];
z7=[z(4,6),z(3,6),z(4,3),z(3,3),z(4,6)];
x8=[x(3,4),x(4,4),x(4,6),x(1,6),x(3,4)];
y8=[y(3,4),y(4,4),y(4,6),y(1,6),y(3,4)];
z8=[z(3,4),z(4,4),z(4,6),z(1,6),z(3,4)];
x9=[x(2,6),x(3,6),x(3,2),x(4,2),x(2,6)];
y9=[y(2,6),y(3,6),y(3,2),y(4,2),y(2,6)];
z9=[z(2,6),z(3,6),z(3,2),z(4,2),z(2,6)];
x10=[x(1,3),x(4,3),x(3,2),x(2,2),x(1,3)];
y10=[y(1,3),y(4,3),y(3,2),y(2,2),y(1,3)];
z10=[z(1,3),z(4,3),z(3,2),z(2,2),z(1,3)];
x11=[x(2,4),x(1,4),x(4,5),x(1,5),x(2,4)];
y11=[y(2,4),y(1,4),y(4,5),y(1,5),y(2,4)];
z11=[z(2,4),z(1,4),z(4,5),z(1,5),z(2,4)];
x12=[x(2,5),x(3,5),x(2,2),x(1,2),x(2,5)];
y12=[y(2,5),y(3,5),y(2,2),y(1,2),y(2,5)];
z12=[z(2,5),z(3,5),z(2,2),z(1,2),z(2,5)];

plot3(x,y,z,X,Y,Z,X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3,X3,Y3,Z3,X4,Y4,Z4,X5,Y5,Z5,X6
,Y6,Z6,X7,Y7,Z7)
hold on
plot3(x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,z5,x6,y6,z6,x7,y7,z7,x8,y8
,z8,x9,y9,z9,x10,y10,z10,x11,y11,z11,x12,y12,z12)
```

Also the shape that we get after following the step method is below.

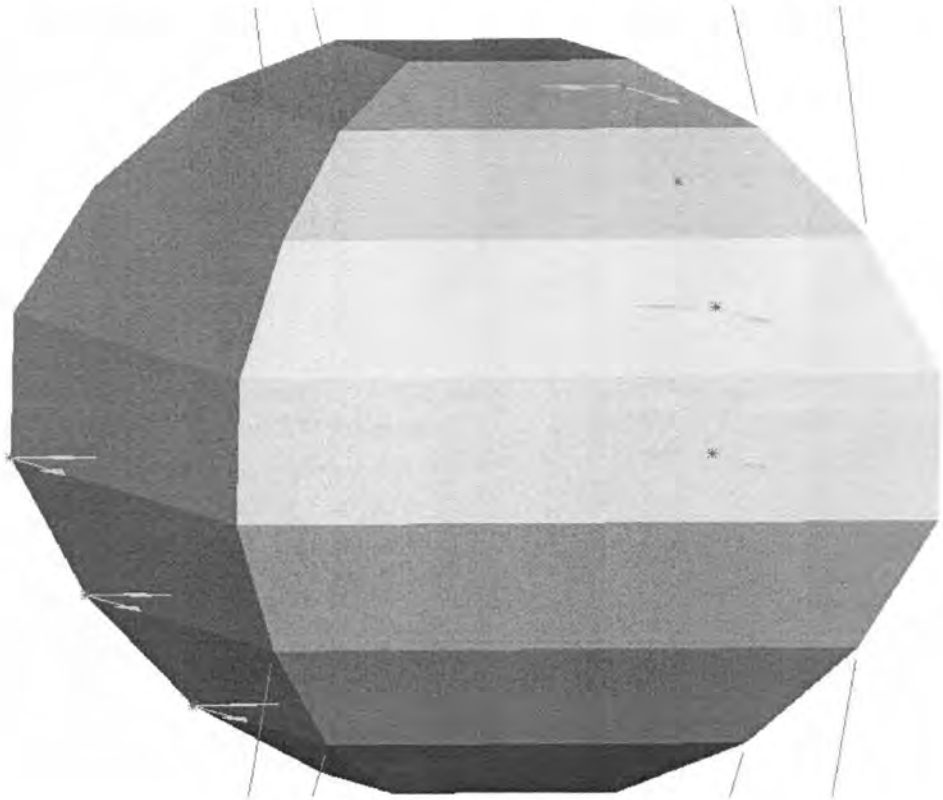


Figure 3.7 Shape after step procedure

3.3) Machining time simulation

Machining time is a function of the length of the machining segment and the shape complexity of the machining segment for a given feed rate. To calculate the machine time we collected all the data, which are necessary to calculate machining time which includes power of machine, spindle speed, cutter data, feed rate, depth of cut, length of cut etc. After determining all the data required we calculated the machining time required getting to doo subdivision step 2.

CHAPTER 4. RESULTS AND CONCLUSION

Doo subdivision along with multi axis machining has given us much better results than using the traditional contouring method used on NC machine. After running simulations for both the methods in case of the cube we got to the following results.

Table 4.1 Comparison of results

Machine and method	Spindle speed	Depth of cut	Total time (Seconds)	Improvement %
five axis machine + Subdivision approach	5000 rpm	.25 inch	44.04	21.2
five axis machine + Subdivision approach	5000 rpm	.35 inch	39.53	29.27
five axis machine + Subdivision approach	5000 rpm	.43 inch	35.62	36.27
3 axis machine + step cut approach	5000 rpm	.25 inch	55.89	
five axis machine + Subdivision approach	4000 rpm	.25 inch	54.18	21.64
five axis machine + Subdivision approach	4000 rpm	.35 inch	48.61	29.69
five axis machine + Subdivision approach	4000 rpm	.43inch	43.81	36.64
3 axis machine + step cut approach	4000 rpm	.25 inch	69.14	
five axis machine + Subdivision approach	3000 rpm	.25 inch	71.06	22.1
five axis machine + Subdivision approach	3000 rpm	.35 inch	63.76	30.1
five axis machine + Subdivision approach	3000 rpm	.43 inch	57.47	37
3 axis machine + step cut approach	3000 rpm	.25 inch	91.22	
five axis machine + Subdivision approach	2000 rpm	.25 inch	104.84	22.57
five axis machine + Subdivision approach	2000 rpm	.35 inch	94.05	30.54
five axis machine + Subdivision approach	2000 rpm	.43 inch	84.73	37.42
3 axis machine + step cut approach	2000 rpm	.25 inch	135.4	

Here we can see that by using Subdivision approach there is an improvement of nearly 20% for the machining time. Also here we have to consider that along with the improvement in machining time there is a significant difference in the quality too as we can see in the figure 3.7 along with figure 3.1, 3.2, 3.3, 3.4 respectively. Here we can also note that by using multi axis machine we can increase spindle speed much higher than the normal NC machine, which will further reducing the machining time. After using Doo Subdivision method for Roughing it is evident by looking at the figures that the time required for finishing will also reduce. So we can conclude that using Doo subdivision method for machining using multi axis machining has not only improved the machining time for rough cuts but also will generate better quality roughed surface which will help to reduce machining time and quality for finishing too.

CHAPTER 5. FUTURE WORK

In this research we have discussed and worked on convex surfaces. Application of wavelets and Subdivision can be extended to concave surfaces too. Multiresolution subdivision surface offset technique is a key for 3D machining. Visibility Maps can determine the visibility or accessibility information at any resolution from lower resolution and wavelet coefficients. The multiresolution analysis based NC tool path planning is expected to be very efficient because of the attractive linear time complexity of wavelets. It can significantly improve the productivity and the accuracy of machined parts. Here in this section we have discussed the possible problems, which are associated with concave surfaces along with recommended approach.

5.1) Wavelet Decomposition

Wavelet based multiresolution analysis will be used to convert the original meshed surface with arbitrary topology to its multiresolution representation (Figure 5.1). Figure 5.1(b) shows the low-resolution representation of the surface in Figure 5.1(a) by implementing a low pass filter A. A high pass filter B obtains the detail part, consisting of wavelet coefficients. The decomposition process, called analysis, further splits Figure 5.1(b) into an even lower resolution version and corresponding wavelet coefficients (Figure 5.1(c)). The filter bank algorithm culminates with the coarsest level representation in Figure 5.1(d), together with wavelet coefficients at each level.

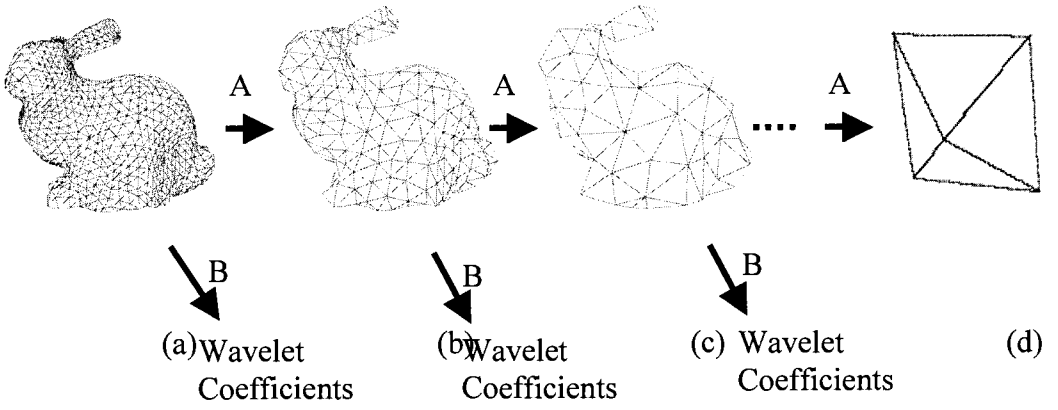


Figure 5.1. Decomposition of a mesh

Tool orientation determination

For a simple convex mesh, the normal direction of every small triangular plane in the mesh, intuitively, can be used to decide the tool orientation for five axis machining. For a complex surface of general topology, the task of tool orientation determination is a local search in the neighborhood of the direction first suggested by visibility analysis. The search criterion is to find the point within each region that results in minimum cusp height. For every sample point in the surface, a set of valid tool orientations will be output after this stage.

5.2) Surface Offset

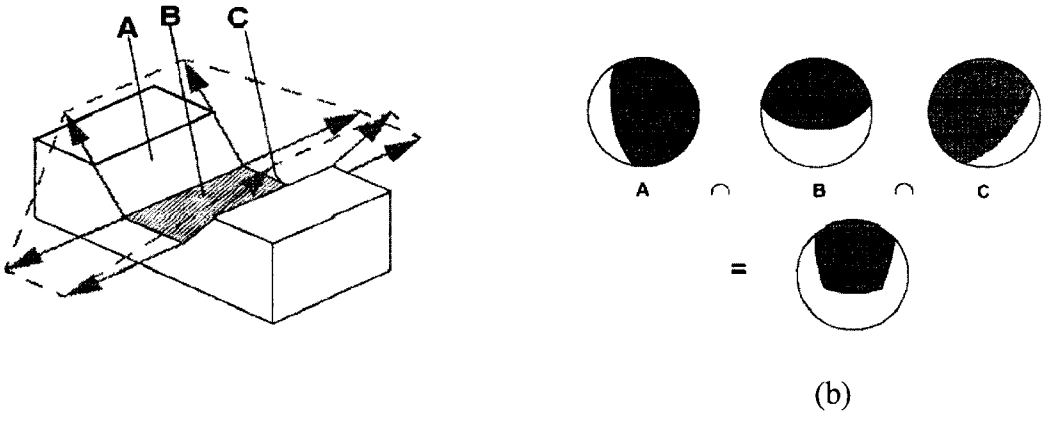
Obtain intermediate surfaces for machining at different levels of detail. The generation of an intermediate surface is a very important component since the accuracy of CNC tool paths is directly related to the accuracy of the intermediate surface. An issue arises when using these simplified surfaces for machining. They are probably intersecting with the original mesh. To avoid gouging in machining, surface offsetting is necessary. Moreover, to apply the

multiresolution surfaces in machining, the surface of a higher resolution model must be covered with that of a lower resolution model in order to prevent over cut.

In this part, an effective and practical offset surface generation method for multiresolution meshed surfaces will be developed. An offset error analysis method based on wavelets will be developed first, in which the multiresolution control method based on the lifting scheme will be used to control offset error to prevent over cut. The value of the wavelet coefficient indicates the difference between a data value that corresponds to the wavelet coefficient and the average of its neighboring data values. Lifting scheme can be used to decide the position of the low-resolution data that are the neighboring points of a higher resolution point corresponding to the wavelet coefficient.

5.3) Multiresolution accessibility analysis

Perform accessibility analysis hierarchically. In this stage, we use the concept of visibility to determine from which directions a point in the offset surface is likely to be accessible to a tool located outside the convex envelop of the object. In our research, visibility information is represented using visibility maps (Chen and Woo, 1992). Visibility cones, which are clusters of visibility directions for points on a work-piece, can be mapped on to the unit sphere to create a visibility map, as shown in Figure 5.4. In Figure 5.4(a), visibility cone for the shaded region is shown; (b) shows the visibility information for the concave region, formed by surfaces A, B and C, is given by the intersection of the individual visibility hemispheres of its neighboring faces.



(a) (b)

Figure 5.4. Visibility cone and visibility maps.

We will develop visibility maps calculation software for triangular meshed surface.

To determine visibility at any resolution, we propose to use visibility information from the lower resolution representations. Wavelet coefficients are used in the procedure. After visibility analysis, the diameter of the tool and the tool holder has to be considered to ensure that the tool can access a required point without any interference

REFERENCES

Austin, S.P., Jerard, P.B. and Drysdale, R.L., 1997, Comparison of discretization algorithms for NURBS surfaces with application to numerically controlled machining, V29, n1, pp71-83.

Bedworth, D.D., Henderson, M.R., and Wolfe, P.M., 1991, Computer-integrated design and manufacturing, Industrial Engineering and Management Science, McGraw-Hill, New York.

Beylkin, G., Coifman, R. and Rokhlin, V., 1991, Fast wavelet transforms and numerical algorithms I, Communications on Pure and Applied Mathematics, V44, pp.141–183.

Bobrow, J.E., 1985, Solid modelers improve NC machine tool path generation techniques, Computers in Engineering, pp. 439-444.

Chen, L.L. and Woo, T.C., 1992, Computational geometry on the sphere with application to automated machining, ASME Trans. J. Mech. Des. 114, pp. 285-95.

Choi, B.K. and Jerard. R.B., 1998, Sculptured surface machining – theory and applications, ISBN 0-412-78020-8, Dordrecht, The Netherlands: Kluwer.

Choi, B.K. and Kim, B.H., 1997 Die-cavity pocketing via cutting simulation Computer Aided Design, Vol. 29, No. 12, pp 837-846

Choi, B.K., Lee, C.S. Hwang, J.S. and Jun, C.S., 1988, Compound surface modeling and machining, Computer Aided Design, V20, n3, pp. 127-136.

Chui, C.K., 1992, An introduction to wavelets, Academic Press, Inc., Boston.

Chui, C.K. and Quak, E., 1992, Wavelets on a bounded interval, Numerical Methods in Application Theory, Vol. 9, pp. 53-75.

Chuang, S. H and Kao, C. Z, 1999, One-sided arc approximation of B-spline curves for Interference-free offsetting, Computer-Aided Design, Vol. 31, No. 2.

Chuang, S.H. and Lin W. S., 1997, Tool-path generation for pockets with freeform curves using bezier convex hulls, Int. J. Adv. Manuf. Technol, Vol. 13, 1997, pp. 109-115.

Denis Zorin, Peter Schroder, Tony De Rose, Leif Kobbelt, Adi Levin, Wim Sweldens: Course material (Subdivision for modeling and animation) http://www.mrl.nyu.edu/~dzorin/sig00course/ (Retrieved on 02/11/2003)

Devor, R.E., Kline, W. A. and Zdeblick, W. J., 1980, A mechanistic model of the force system in end milling with application to machining airframe structures, Proc. 8th NAMRC Conference, pp. 297-303.

DeVore, R., Jawerth, B. and Lucier, B., 1992, Image compression through wavelet transform coding. IEEE Transactions on Information Theory, 38(2): 719–746, March 1992.

Dragomatz, D. and Mann, S., 1997, A classified bibliography of literature on NC milling path generation, Computer-Aided Design, Vol. 29, No. 3, pp. 239-247.

Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W., 1995, Multiresolution analysis of arbitrary meshes, Siggraph, pp. 173-182.

Elber, G. and Cohen, E., 1994, Toolpath generation for freeform surface models, Computer Aided Design, Vol. 26, No. 6, pp. 490-496.

Elber, G., Lee, I. and Kim, M., 1997, Comparing offset curve approximation methods, IEEE Computer Graphics and Applications," May-June 1997, pp.62-71.

Faramarz F. S., Nezam M. and Richard H. B., 2001, Multiresolution surfaces having arbitrary topologies by a reverse doo subdivision Submitted to Computer Graphics forum 2001.

Faramarz F. S. and Richard H. B., 1999, Multiresolution curve and surface representation: reversing subdivision rules. Computer Graphics Forum, 18(2): 97-119, June 1999

Finkelstein, A. and Salesin, D. H., 1994, Multiresolution Curves, Computer Graphics Proceedings, Annual Conferences, pp. 261-268.

Greenwood, F., 1989, Introduction to computer integrated manufacturing, Harcourt Brace Jovanovich.

Held, M., Lukacs, G. and Andor,A., 1994, Pocket machining based on contour-parallel tool paths generated by means of proximity maps, Computer Aided Design., Vol. 26, No. 3, pp. 189-203.

Jenson, C.G. and Anderson, D.C., 1992, Accurate tool placement and orientation for finish surface machining, Concurrent Engineering, Ped -Vol 59, ASME, pp. 127-145.

Jenson, C.G. and Anderson, D.C., 1996, A review of numerically controlled methods for finish-sculptured-surface machining, IIE Transactions, 28, pp. 30-39.

Jenson, C. G., Mullins, S. H., Anderson, D. C., 1993, Scallop elimination based on precise 5-axis tool placement, orientation and step-over calculations, ASME – Adv Des Automat, 65(2), pp. 535-44.

Jenson, C.G., Red, W.E., Pi, J., 2002, Tool selection for 5-axis curvature matched machining, Computer Aided Design, 34, pp.251-66.

Kneisl, G. provided Doo Subdivision code for some surfaces, Institute for analysis and scientific computing, Vienna University of Technology, Austria

Kruth J. P. and Klewais, P., 1994, Optimization and dynamic adaptation of the cutter inclination during 5-axis milling of sculptured surfaces, CIRP Ann, 43(1), pp. 443-8.

Lee, A., Sweldens, W., Schroder, P., Cowsar, L., and Dobkin, D., 1998, Maps: Multiresolution adaptive parameterization of surfaces, Siggraph, pp. 95-104

Lee, Y.S. and Chang, T.C., 1995, 2-Phase approach to global tool interference avoidance in 5-axis machining, Computer Aided Design, 27(1), pp. 715-29.

Lee, Y.S. and Chang, T.C., 1996, Automatic cutter selection for 5-axis sculptured surface machining, International Journal of Production Research, 34, pp. 997-8.

Lee, Y.S., Ji, H, 1997, Surface interrogation and machining strip evaluation for 5-axis CNC die and mold machining, International Journal of Production Research, 35(1), pp. 225-52.

Loop, C., 1987, Smooth subdivision surface based on triangles, Master's Thesis, Department of Mathematics, University of Utah.

Lounsbery, J. M., 1995 Multiresolution analysis for surface of arbitrary topological type, Ph.D. thesis, University of Washington.

Lounsbery, J. M., DeRose, T. and Warren, J., 1997, Multiresolution surfaces of arbitrary topological type, ACM Transactions on Graphics, Vol. 16, No 1, January, 1997, pp. 34-73.

Mallat, S., 1989, A theory for multiresolution signal decomposition, IEEE Trans. Pattern Anal. Machine Intell., V11, pp. 674-693.

Maekawa, T., 1996, Computation of shortest paths on free-form parametric surfaces, ASME Journal of Mechanical Design, V118, n4, pp. 499-508.

Marshall, S. and Griffiths, J.G., 1994, A survey of cutter path construction techniques for milling machines, International Journal of Production Research, 32(12), pp. 2861-77.

Oliver, J.H. and Huang, Y., 1994, Non-constant parameter NC tool path generation on sculptured surfaces, International Journal of Advanced Manufacturing Technology, Vol. 9, pp. 281-290.

Persson, H., 1978, NC Machining of arbitrary shaped pockets, Computer Aided Design, Vol. 10, No. 3, pp. 169-174.

Quak, E. and Weyrich, N., 1994, Decomposition and reconstruction algorithms for spline wavelets on a bounded interval, Applied and Computational Harmonic Analysis, Vol. 1, No.3, pp. 217-231.

Rao, N., Bedi, S., Buchal, R., 1996, Implementation of the principal-axis method for machining of complex surfaces International Journal of Advanced Manufacturing Technology, 11, pp. 249-57.

Redonnet J.M., Rubio, W., Monies, F., Dessein, G., 1998, Optimizing tool positioning for end-mill machining of free form surfaces on five-axis machines for both semi finishing and finishing, International Journal of Advanced Manufacturing Technology, 16 pp. 383-91.

Sarma, R., 2000, An assessment of geometric methods in trajectory synthesis for shape-creating manufacturing operations, Journal of Manufacturing Systems, Vol. 19, No. 1, pp. 59-72.

Schroder, P. and Sweldens, W., 1995, Spherical wavelet: efficiently representing functions on the sphere, Siggraph '95 Proc., pp.161-72, ACM.

Shah, J.J., Sreevalsan, P. and Matthew, A., 1991, Survey of CAD/feature-based process planning and NC programming techniques, Computer Aided Engineering. Journal, V8, n1, pp. 832-838.

Stollnitz, E.J., DeRose, T.D. and Salesin, D.H., 1995, Wavelets for computer graphics: A Primer Part *1*, IEEE Computer Graphics and Applications, pp. 76-84.

Stollnitz, E.J., DeRose, T.D. and Salesin, D.H., 1995, Wavelets for computer graphics: A primer part 2, IEEE Computer Graphics and Applications, pp. 75-85.

Suresh, K. and Yang, D.C.H., 1994, Constant scallop height machining of free form surfaces, ASME Journal of Engineering For Industry, V116, pp. 253-259.

Sweldens, W., 1995, The lifting scheme: A new philosophy in biorthogonal wavelet constructions, In A. F. Laine and M. Unser, editors, Wavelet Applications in Signal and Image Processing III, pp.68-79, Proc. SPIE 2569.

Tiller, W. and Hanson, E.G., 1984, Offsets of two-dimensional profiles, IEEE Computer Graphics and Applications, pp. 36-46.

Toussaint, G. T., 1985, A simple linear algorithm for intersecting convex polygons, The Visual Computer, vol. 1, pp. 118-123.

Tseng, Y. J. and Joshi, S., 1994, Recognizing multiple interpretations of interacting machining features, Computer Aided Design, 26(9), pp.667-688.

Vickers, G.W. and Bradley, C., 1992, Curved surface machining through circular arc interpolation, Computers in Industry, Vol.19, pp. 329-337.

Wang, Y., Lee, S.L., and Toraichi, K., 1999, Multiscale curvature-based shape representation using b-spline wavelets, IEEE Transactions on Image Processing, Vol. 8, No. 11, pp. 1586-1592.

Wood, Z., Desbrun, M., Schroder, P., and Breen, D., 2000, Semi-regular mesh extraction from volumes, Siggraph.

APPENDIX

This research has used this doo subdivision code, part of which was provided by Günter Kneisl.

```
function varargout=qdoo(argpt,argpgon,n,varargin)
% DOO Doo-Sabin subdivision algorithm
%
% Given a polyhedral mesh defined by points PT and polygons PGON,
%
% [PTOUT,PGONOUT]=DOO(PT,PGON,N,PARAM1,VALUE1,PARAM2,...)
%
% applies the Doo-Sabin subdivision algorithm N times.
% MESH MUST BE CLOSED AND ORIENTABLE.
% For almost all input data (wrt to Lebesgue-measure), the sequence
% of meshes converges to a C^1 surface. (If for example all points
% of a polygon have the same coordinates (a pathological polygon),
% you may end up with a cusp.)
%
% PT should be an m-by-3 matrix containing the coordinates
% of the points, PGON should be a cell array of polygons, each
% polygon being a point list(of indices to points).
%
% Parameters:
%    ORIENTATION [Positive|Negative|{None}]
%    OUTPUT            [Full |{Simple}]
%    STATS             [{On} | Off]
%
% Set ORIENTATION to 'Positive' if each edge of the
% mesh is passed once in each direction, and polygon points
% rotate counterclockwise with respect to the normal vector.
% Set ORIENTATION to 'Negative' if points rotate clockwise.
%
% If OUTPUT is set to 'Simple' (the default) the output
% has the same form as the inputs PT and PGON.
% If OUTPUT is set to 'Full' PTOUT is an m-by-5 matrix, one
% row per point, the first three entries being the coordinates,
% the fourth the number of a polygon that the point belongs to,
% and the fifth the index of the point in that polygon.
% OUTPGON{i} is a 2-by-nrpts+1 matrix, the first row
% containing the points of which the polygon consists with the
% first point listed again at the end, the second the indices
% of the adjacent polygons.
%
% If STATS is 'On' some calculating statistics are displayed.
%
%
% Example(Dice):
%
% pt=[0 0 0;1 0 0;1 1 0;0 1 0;...
%       0 0 1;1 0 1;1 1 1;0 1 1];
% pgon={[1 2 3 4] [5 8 7 6] [1 5 6 2]...
%             [3 7 8 4] [2 6 7 3] [4 8 5 1]};
% doo(pt,pgon,3);
```

```
%
% See also PGONDISP PGONTRACE ORIENT CHSURF



global  pt ptlen pgon pgonlen nonedgepgonlen nrpts sumnrpts newpt newpgon
w;


%******************** Argument check
****************************************
%*************************************************************************
*****

error(nargchk(3,7,nargin));
if mod(nargin,2)==0
   error('wrong number of input arguments');
end


if size(argpt,2)~=3
   pt=argpt';
else
   pt=argpt;
end

if size(pt,2)~=3
   error(' pt should be a m-by-3 matrix ');
end

pgon=reshape(argpgon,length(argpgon),1);

orientation='none';
output='simple';
checks='on';
stats='on';

orientation=0;
or=zeros(length(pgon),1);
or(1)=1;

for i=1:2:length(varargin)
   switch lower(varargin{i}(1:2))
   case 'or'  % orientation
      switch lower(varargin{i+1}(1:2))
      case 'po' % positive
         orientation=1;
      case 'ne' % negative
         for j=1:length(pgon)
            pgoni=pgon{i};
            pgon{i}=pgoni(length(pgoni):-1:1);
         end
         orientation=1;
      case 'no' % none
```

```
      otherwise
         error('Bad value for property ''orientation''.');
      end

   case 'ou'  % output
      switch lower(varargin{i+1}(1))
      case 'f' % full
         output='full';
      case 's' % simple
      otherwise
         error('Bad value for property ''output''.');
      end

   case 'st' % stats
      switch lower(varargin{i+1}(1:2))
      case 'on'
      case 'of'
         stats='off';
      otherwise
         error('Bad value for property ''stats''.');
      end

   otherwise
      error(['Invalid property : ''' varargin{i} '''']);
   end
end



%******************* get additional information about input data
***************
%*************************************************************************
*****

tic;

if orientation==0
   t0=clock;

   pgon=pgonorient(pgon); % orients the polygons

   if stats(1:2)=='on'
      ortime=etime(clock,t0);
      fprintf('\n Orientation  :  %.2fs ',ortime);
   end
end

initpgonglobals(pgon);

pgon=order(pgon);
%***** orders the polygon list such that it starts with the extraordinary
%***** pgons (=pgons with n~=4 vertices)

pt=fullpointinfo(pt,pgon);
%***** To each point, this function calculates the polygons
```

```
%***** Containing it, and the indices of the point in these polygons.

initpointglobals(pt);


pgon=addfirstpoint(pgon);
%***** Adds the first point of the pgon to the
%***** end of the pgons's pointlist (convenient when referencing)

pgon=fulledgeinfo(pt,pgon);
%***** To each polygon's edge, this function calculates the index of the
%***** Adjacent polygon, and the index of the edge in this polygon.
%***** POLYGONS MUST BE ORIENTED NEGATIVELY


w=costable(pgon);
%***** Calculates the weights for convex-combination (doo-weights)




%************************ first doo step
************************************
%***** It is useful to treat this step separately, because the polygon-
point
%***** data looks much simpler after its completion.(e.g. all vertices
%***** connect exactly 4 edges)
%*************************************************************************
****



[planeidx,vertidx,edgeidx,ptidx]=linearindexing(pt,pgon);
%***** Defines linear indices of both points and pgons of the next doo-
step.
%***** planeidx contains indices of planepolygons, edgeidx->edgepolygons
%***** vertidx->vertexpolygons

[newpt,newpgon]=planepolygons(pt,pgon,planeidx,edgeidx);
%***** Calculates the new points of a doo-sabin step, and the polygons,
that
%***** Derive from the polygons of the previous step by shrinking.

[newpt,newpgon]=vertexpolygons(pt,pgon,newpt,newpgon,vertidx,edgeidx,ptidx
);
%***** Calculates the polygons that derive from cutting off the vertices.

newpgon=edgepolygons(newpt,newpgon,pgon,edgeidx,vertidx,ptidx);
%***** Calculates the polygons that derive from cutting off the edges.



%****************** Get some information about new data
********************
```

```
%*************************************************************************
**


redefineglobals;

nrpts=zeros(pgonlen,1);
for i=1:pgonlen
   nrpts(i)=size(pgon{i},2)-1;
end
sumnrpts=sum(nrpts);

%***** This overrides some settings of redefineglobals.
%***** we merely need it after the first doo-step.


%************************ Rest of the doo steps
*****************************
%***** Algorithm is the same in principal, but takes advantage of the
%***** improved knowledge about the data and the improved data structure.
%***** For greater speed we use global variables, but the data dependence
%***** is exactly the same as in the non-quick-routines. (The q stands for
%***** quick)
%*************************************************************************
**



if stats(1:2)=='on'
   tnew=toc;
   fprintf('\n DooStep 1  :  %.2fs \n',tnew);
end


for i=1:n-1
   qlinearindexing;
   qplanepolygons;
   qvertexpolygons;
   qedgepolygons;
   redefineglobals;
   if stats(1:2)=='on'
      told=tnew;
      tnew=toc;
      fprintf(' DooStep %i  :  %.2fs \n',i+1,tnew-told);
   end
end

if stats(1:2)=='on'
   tnew=toc;
   fprintf('\n Total time    :  %.2fs \n',tnew);
   fprintf('\n Polygons : %i \n',pgonlen);
   fprintf(' Points      : %i \n\n',ptlen);
end
```

```
%************************ Output argument check
*****************************
%*************************************************************************
***

switch nargout
case 0
   pt=pt(:,1:3);
   for i=1:pgonlen
     pgon{i}=pgon{i}(1,1:nrpts(i));
   end
   pgondisp(pt,pgon);
case 2
   if output(1)=='s'  % simple
      pt=pt(:,1:3);
      for i=1:pgonlen
         pgon{i}=pgon{i}(1,1:nrpts(i));
      end
      varargout={pt pgon};
   else               % full
      for i=1:nonedgepgonlen
         pgon{i}=pgon{i}(1:2,:);
      end
      varargout={pt pgon};
   end
otherwise
   error(' Wrong number of output arguments');
end



%*************************** Subroutines
***********************************
%*************************************************************************
***



function initpgonglobals(pgon);
%***** we initialize our polygon-global
global pgonlen nrpts sumnrpts

pgonlen=length(pgon);
nrpts=zeros(pgonlen,1);
for i=1:pgonlen
   nrpts(i)=size(pgon{i},2);
end
sumnrpts=sum(nrpts);


function redefineglobals
%***** This converts the output of a doo-step to input for the next
global  pt ptlen pgon pgonlen nrpts newpt newpgon nonedgepgonlen sumnrpts
```

```
pt=newpt;pgon=newpgon;

nonedgepgonlen=pgonlen+ptlen;
%***** polygons that do not derive from edges

pgonlen=pgonlen+ptlen+sumnrpts/2;
%***** #planepgons + #vertpgons + #edgepgons

oldptlen=ptlen;
ptlen=sumnrpts;
nrpts=[nrpts ; 4*ones(oldptlen+sumnrpts/2,1)];
%***** new polygons have exactly 4 vertices

sumnrpts=sumnrpts+4*(oldptlen+sumnrpts/2);

function newpgon=addfirstpoint(pgon);
%***** Adds the first point of the pgon to the
%***** end of the pgons's pointlist (convenient when referencing)

for i=1:length(pgon)
   pgoni=pgon{i};
   newpgon{i}=[pgoni pgoni(1)];
end

function newpgon=order(pgon)
%***** This function takes the extraordinary pgons to the
%***** Beginning of the pgonlist.

global pgonlen nrpts

expgonidx=find(nrpts~=4);
expgon=pgon(expgonidx);

expgonlen=length(expgon);
orpgonlen=pgonlen-expgonlen;

pgon(expgonidx)=[];
newpgon=[expgon ; pgon];

nrpts =[nrpts(expgonidx); 4*ones(orpgonlen,1)];

function newpt=fullpointinfo(pt,pgon)
%***** To each point, this function calculates the polygons
%***** Containing it, and the indices of the point in these polygons.

%***** Data format: coos = 1-by-3 vector of point's coordinates
%*****               pgon = a 2-by-nrpts matrix, the first row is an
%*****                      index vector to the polygons containing this
point,
%*****                      the second indicates the position of the point
in
%*****                      these polygons

global pgonlen nrpts ptlen
```

```
newpt=cell(length(pt),1);
for i=1:length(pt)
   newpt{i}.coos=pt(i,:);  % convert points coordinates to new
   newpt{i}.pgon=[];       % data format
end

for i=1:pgonlen
   pgoni=pgon{i};
   for j=1:nrpts(i)
      newpt{pgoni(j)}.pgon=[newpt{pgoni(j)}.pgon [i;j]];
   end                        % concatenate pgon-data([i,j]) to
end                           % existing pt.data


function initpointglobals(pt)
%***** we initialize our point-globals

global ptlen nrpgons expgonlen

ptlen=length(pt);
nrpgons=zeros(ptlen,1);
for i=1:ptlen
   nrpgons(i)=size(pt{i}.pgon,2);
end

troubles=find(nrpgons<3);
if ~isempty(troubles)
   fprintf('\n An error has occured. The points : [%s]\n',...
            int2str(troubles));
   fprintf(' are contained in less than 3 polygons \n\n');
   error('Input Data Error');
end


function newpgon=fulledgeinfo(pt,pgon,orientation,or)
%***** To each polygon's edge, this function calculates the index of the
%***** adjacent polygon, and the index of the edge in this polygon.
%***** POLYGONS MUST BE ORIENTED NEGATIVELY

%***** data structure: newpgon{i} is a 3-by-nrpts+1 matrix, the first row
%***** containing the points of which the polygon consists, the second the
%***** Indices of the adjacent polygons, the third the indices of the
edges
%***** in the adjacent polygons. Edges are 1-2,2-3,...,end-1.

global pgonlen ptlen nrpts nrpgons

for i=1:pgonlen
   newpgon{i}=zeros(3,nrpts(i)+1);   % reserve space
   newpgon{i}(1,:)=pgon{i};          % and convert the polygondata

end


for i=1:pgonlen
```

```
   pgoni=pgon{i};              % speed

   for j=1:nrpts(i)
      if newpgon{i}(2,j)==0       % this pgon has not already
                                  % been taken into account

         thispt=pgoni(j);         % clearness
         nextpt=pgoni(j+1);       % clearness

         adjpgon =
intersection(pt{thispt}.pgon(1,:),pt{nextpt}.pgon(1,:),i);

                     % the index of the adjacent polygon
                     % can be determined by intersecting the polygons
containing
                     % the first edge point with those containing the
second.


         if isempty(adjpgon)
            fprintf('\n An error has occured at edge %i \n',j);
            fprintf(' of polygon %i. I did not find an adjactent polygon.
\n',i);
            error('Input Data Error');
         end

         adjedge=findfirst(pgon{adjpgon},nextpt);

                     % Now get the index of the edge in the calculated
pgon
                     % here we need the orientation of the polygons


         newpgon{i}(2:3,j)=[adjpgon adjedge];     % and set the values
         newpgon{adjpgon}(2,adjedge)=i;

                     % we do not need the other indices of the adjactent
pgon
      end            % any more, so we leave them zero
   end
end

function val=intersection(a,b,n)
%***** if a and b contain exactly two equal
%***** values, and one of them is given as n,
%***** this function calculates the other.

val=[];
for i=1:length(a)
   if a(i)~=n
      f=b(find(b==a(i)));
      if ~isempty(f)
         val=f(1);
         break;
      end
```

```
   end
end

function first=findfirst(array,number)
%***** If array is known to contain number, this function returns
%***** the index of the first appearance of number in array.

i=1;
while 1
   if array(i)==number
      first=i;
      break;
   end
   i=i+1;
end
function out=costable(pgons)
%***** Calculates the weights for convex-combination of the polygon points

global nrpts nrpgons

for n=3:max([nrpts ; nrpgons ; 4]) % this number does not increase along
iterations
   out{n}=zeros(n,n);
   w(1)=1/4*(1+5/n);
   for j=2:n
      w(j)=(3+2*cos(2*pi*(j-1)/n))/(4*n);
   end                          % calculate the doo-weights for first point
   out{n}(1,:)=w;

   for k=2:n                    % and now rotate them to get the doo-
weights
      w=[w(n) w(1:n-1)];        % for the other points
      out{n}(k,:)=w;
   end
end


function [planeidx,vertidx,edgeidx,ptidx]=linearindexing(pt,pgon);
%***** Defines linear indices of both points and pgons of the next doo-
step.

global pgonlen nrpgons nrpts

pgons4=(nrpgons==4);          % we determine which points are ordinary

orptidx=find(pgons4);         % and get their indices
orptlen=length(orptidx);

exptidx=find(~pgons4);        % the same for extraordinary points
exptlen=length(exptidx);


expgonlen=length(find(nrpts~=4));  % number of extraordinary polygons
orpgonlen=pgonlen-expgonlen;
```

```
lxlow=1;                    % linear index
lxhigh=expgonlen+1;         % the first pgons derive from extraordinary
planepgons.
planeidx=lxlow:lxhigh-1;    % they are extraordinary themselves


lxlow=lxhigh;               % then we reserve space for the extraordinary
lxhigh=lxlow+exptlen;       % pgons that derive from extraordinary vertices.
vertidx(exptidx)=lxlow:lxhigh-1;

lxlow=lxhigh;               % all extraordinary pgons have been taken into
account,
lxhigh=lxlow+orpgonlen;     % next polygons are those from ordinary pgons
planeidx=[planeidx lxlow:lxhigh-1];


lxlow=lxhigh;               % polygons from ordinary vertices
lxhigh=lxlow+orptlen;
vertidx(orptidx)=lxlow:lxhigh-1;

lx=lxhigh;                  % polygons that derive from edges are last

edgeidx=cell(pgonlen,1);
for i=1:pgonlen
   edgeidx{i}=zeros(2,nrpts(i)+1);
end

for i=1:pgonlen
   pgoni=pgon{i};           % speed
   pgoni2=pgoni(2,:);
   pgoni3=pgoni(3,:);

   for j=1:nrpts(i)
      if edgeidx{i}(1,j)==0  % this pgon has not already been taken into
account

         edgeidx{i}(:,j)=[lx ; 4];  % this edgepgon derives from the
                                    % j-th edge of the pgon i

                                    % the edgepolygon starts at the
current
                                    % point, and therefore its fourth edge
                                    % will be adjacent to the shrunken
pgon i

         edgeidx{pgoni2(j)}(:,pgoni3(j))=[lx 2];
                           % the appropriate settings for the adjacent pgon
         lx=lx+1;
      end
   end
end

ptlx=1;                  % while the linear indexing of pgons looks rather
for i=1:pgonlen          % complicated, linear indexing of points is very
simple
```

```
   ptlxhigh=ptlx+nrpts(i)-1;
   ptidx{i}=[ptlx:ptlxhigh ptlx];
   ptlx=ptlxhigh+1;
end
```

```
function [newpt,newpgon]=planepolygons(pt,pgon,planeidx,edgeidx)
%***** Calculates the new points of a doo-sabin step, and the polygons,
that
%***** derive from the polygons of the previous step by shrinking.

global pgonlen ptlen nrpts sumnrpts nrpgons w

newpgon=cell(pgonlen+ptlen+sumnrpts/2,1);% #planepgons + #vertpgons +
#edgepgons

newpt=ones(sumnrpts,5); % note that we change the data-structure of the
points.
                        % this is doo to the fact, that all new points are
                        % ordinary (connect 4 edges). The first three
entries
                        % are the points coordinates, the fourth is a pgon
                        % that contains the point (no matter which one)
and
                        % the fifth is the index of the point in that pgon

lx=1;                   % linear index
for i=1:pgonlen
   nrptsi=nrpts(i);     % speed
   lxhigh=lx+nrptsi-1;

   newpgon{planeidx(i)}=[[lx:lxhigh lx] ; edgeidx{i}];
                        % the shrunken polygon is found easily


   p=zeros(nrptsi,3);    % now for the new points:
   for j=1:nrptsi        % we gather the pgon's point's coordinates in p
      p(j,:)=pt{pgon{i}(1,j)}.coos;
   end

   newpt(lx:lxhigh,:)=[w{nrptsi}*p   planeidx(i)*ones(nrptsi,1)
[1:nrptsi]'];
                        % and convex-combine them using the weights w
   lx=lxhigh+1;
end

function
[newpt,newpgon]=vertexpolygons(pt,pgon,newpt,newpgon,vertidx,edgeidx,ptidx
)
```

```
%***** Calculates the polygons that derive from cutting off the vertices.
%***** This algorithm may look a bit clumsy, but consider that we do not
%***** know the order in which to connect the points that form our new
%***** Vertex polygon. The points themselves are easily found.

global ptlen nrpts nrpgons

for i=1:ptlen
   nrpgonsi=nrpgons(i);                    % speed
   vertpgoni=zeros(3,nrpgonsi+1);          % speed

   pgonnr  =pt{i}.pgon(1,1);     % pgonnr stands for a the index of a
polygon
   pgonptnr=pt{i}.pgon(2,1);     % containing the point i. We start
arbitrarily
   if pgonptnr==1                % with the first pgon in the pgon list of
i.
      edgenr=nrpts(pgonnr);      % pgonptnr is the index of the current
point
   else                          % in this pgon, and edgenr is the number
of
      edgenr=pgonptnr-1;         % the edge leading to this point.
   end


   for j=1:nrpgonsi-1
      vertpgoni(1:3,j)=[ptidx{pgonnr}(pgonptnr);
edgeidx{pgonnr}(1,edgenr);...
                        edgeidx{pgonnr}(2,edgenr)-1];

         % stores the information about the j-th vertex of the
         % vertex polygon in vertpgoni. First are the points the
         % vertex polygon consists of, second the adjacent pgons
         % and third the edge indices.

      pgonnr=pgon{pgonnr}(2,edgenr);
      pgonptnr=findfirst(pgon{pgonnr}(1,:),i);

         % The new pgonnr is the number of the polygon adjactent
         % to the old pgonnr along the edge edgenr. This polygon
         % must contain our point i,so we search for its index.
         % Doing this nrpgonsi times we encircle our vertex i and
         % find our vertexpolygon.

      if pgonptnr==1
         edgenr=nrpts(pgonnr);
      else
         edgenr=pgonptnr-1;
      end

   end

   vertpgoni(1:3,nrpgonsi)=...
      [ptidx{pgonnr}(pgonptnr); edgeidx{pgonnr}(1,edgenr);...
       edgeidx{pgonnr}(2,edgenr)-1];
```

```
   newpgon{vertidx(i)}=vertpgoni;
         % now take all the info about the vertexpgon to newpgon
   newpgon{vertidx(i)}(1,nrpgonsi+1)=vertpgoni(1,1);
         % for conveniece's sake we add the first point as last again
end

function newpgon=edgepolygons(newpt,newpgon,pgon,edgeidx,vertidx,ptidx)
%***** Calculates the polygons that derive from cutting off the edges.

global pgonlen nrpts sumnrpts

edgepgon=cell(sumnrpts/2,1);     % We know how many this will be

lx=edgeidx{1}(1,1);
for i=1:pgonlen
   pgoni=pgon{i};
   pgoni1=pgoni(1,:);
   pgoni2=pgoni(2,:);
   pgoni3=pgoni(3,:);
   ptidxi=ptidx{i};

   for j=1:nrpts(i)
      pgoni3j=pgoni3(j);       % speed

      if pgoni3j~=0            % an edge belongs to 2 pgons => no double
counts,
         pgoni2j=pgoni2(j);    % (we left pgon{i}(3,j) zero for pgons that
         ptidxpgoni2j=ptidx{pgoni2j};
                               % share edge j with a pgon of lower lx
                               % (in fulledgeinfo))
         newpgon{lx}...
         =[ptidxi(j)                          ptidxpgoni2j(pgoni3j+1)...
           ptidxpgoni2j(pgoni3j)            ptidxi(j+1)
ptidxi(j);
           vertidx(pgoni1(j))                 0 ...
           vertidx(pgoni1(j+1))               0              0];

                  % the last zero is dummy, the other ones replace
                  % pgondata that will not be used in subsequent
calculations.
         lx=lx+1;
     end
   end
end


function qlinearindexing;
%***** Defines linear indices of both points and pgons of the next doo-
step.
%***** Quick version, taking advantage of improved data structure

global  pt ptlen nrpts pgon pgonlen nonedgepgonlen vertidx edgeidx ptidx
```

```
lx=pgonlen+ptlen;      % we do not need planeidx anymore, since
planeidx(i)=i
vertidx=pgonlen+1:lx;  % so we start with vertidx

lx=lx+1;

edgeidx=cell(pgonlen,1);

for i=nonedgepgonlen+1:pgonlen      % then edgeidx
   edgeidx{i}=2*ones(2,nrpts(i)+1); % the last coordinate is dummy (concat
fit)
end                                 % the edge indices are known to
                                    % be equal to 2 (sketch).


for i=1:nonedgepgonlen
   nrptsi=nrpts(i);                 % speed
   pgoni=pgon{i};

   edgeidx{i}=[lx:lx+nrptsi ; 4*ones(1,nrptsi+1)];

                                    % this edgepgon derives from the
                                    % j-th edge of the pgon i
                                    % the edge polygon starts at the
current
                                    % point, and therefore its fourth edge
                                    % coincides with the current.
   for j=1:nrptsi
      edgeidx{pgoni(2,j)}(1,pgoni(3,j))=lx;
      lx=lx+1;          % the appropriate settings for the adjactent pgon
   end
end



ptlx=1;
for i=1:pgonlen                     % linear indexing of points stays the
   ptlxhigh=ptlx+nrpts(i)-1;        % same.
   ptidx{i}=[ptlx:ptlxhigh ptlx];
   ptlx=ptlxhigh+1;
end


function qplanepolygons
%***** Calculates the new points of a doo-sabin step, and the polygons,
that
%***** derive from the polygons of the previous step by shrinking.
%***** Quick version, taking advantage of improved data structure

global pt ptlen pgon pgonlen nrpts newpt newpgon edgeidx sumnrpts w

newpgon=cell(pgonlen+ptlen+sumnrpts/2,1);%#planepgons + #vertpgons +
#edgepgons
newpt=ones(sumnrpts,5);
```

```
lx=1;                    % linear index
for i=1:pgonlen
   nrptsi=nrpts(i);

   lxhigh=lx+nrptsi-1;

   newpgon{i}=[[lx:lxhigh lx] ; edgeidx{i}];      % planepolygon easily
found

   p=pt(pgon{i}(1,1:nrptsi),1:3); % We gather the pgon's point's
coordinates in p

   newpt(lx:lxhigh,:) = [w{nrptsi}*p i*ones(nrptsi,1) [1:nrptsi]'];
                                  % and convex-combine them to the new
points
   lx=lxhigh+1;

end

function qvertexpolygons
%***** Calculates the polygons that derive from cutting off the vertices.
%***** This algorithm may look a bit clumsy, but consider that we do not
%***** know the order in which to connect the points that form our new
%***** vertex polygon. The points themselves are easily found.
%***** Quick version, taking advantage of improved data structure


global pt ptlen pgon nrpts newpgon edgeidx vertidx ptidx


for i=1:ptlen

   vertpgoni=zeros(3,5);

   pgonnr  =pt(i,4);  % Due to the changed data format
   pgonptnr=pt(i,5);
   if pgonptnr==1
      edgenr=nrpts(pgonnr);
   else
      edgenr=pgonptnr-1;
   end


   for j=1:3

      vertpgoni(1,j)=ptidx{pgonnr}(pgonptnr);
      vertpgoni(2,j)=edgeidx{pgonnr}(1,edgenr);



      pgonnr=pgon{pgonnr}(2,edgenr);
      pgonptnr=findfirst(pgon{pgonnr}(1,:),i);
      if pgonptnr==1
         edgenr=nrpts(pgonnr);
```

```
      else
         edgenr=pgonptnr-1;
      end
   end

   vertpgoni(1,4)=ptidx{pgonnr}(pgonptnr);
   vertpgoni(2,4)=edgeidx{pgonnr}(1,edgenr);


   vertpgoni(3,1:4)=[3 1 3 1];  % due to the ordering of the polygons,
                                % we know this in advance (sketch)

   newpgon{vertidx(i)}=vertpgoni;
   newpgon{vertidx(i)}(1,5)=vertpgoni(1,1);
end



function qedgepolygons
%***** Calculates the polygons that derive from cutting off the edges.
%***** Quick version, taking advantage of improved data structure

global  pgon nonedgepgonlen nrpts newpgon vertidx edgeidx ptidx

lx=edgeidx{1}(1);
for i=1:nonedgepgonlen
   pgoni=pgon{i};
   pgoni1=pgoni(1,:);
   pgoni2=pgoni(2,:);
   pgoni3=pgoni(3,:);
   ptidxi=ptidx{i};

   for j=1:nrpts(i)
      pgoni2j=pgoni2(j);
      pgoni3j=pgoni3(j);
      ptidxpgoni2j=ptidx{pgoni2j};

      newpgon{lx}...
         =[ptidxi(j)                    ptidxpgoni2j(pgoni3j+1)...
           ptidxpgoni2j(pgoni3j)        ptidxi(j+1)       ptidxi(j);...
           vertidx(pgoni1(j))           0 ...
           vertidx(pgoni1(j+1))         0                 0];
     lx=lx+1;
   end
end


function opgon=pgonorient(argpgon)
% PGONORIENT orients a closed surface of polygons
%
% Let pgon be a cell array of polygons, each polygon
% being a pointlist(of indices to points), then
% orpgon=PGONORIENT(pgon) orients the polygons defined by
% pgon with respect to each other.
% More accurately, it reverses the order of some polygons
```

```
% such that in the outgoing surface each edge is run once
% in each direction. SURFACE MUST BE CLOSED AND ORIENTABLE.
% The first polygon is assumed to have the right orientation.
%
% Example(Tetraeder):
%
%   >>pgon={[1 2 3] [1 2 4] [2 3 4] [1 3 4]};
%   >>newpgon=pgonorient(pgon);
%
% See also DOO  PGONDISP  PGONTRACE  CHSURF

global opgon opgonlen onrpts or nrorpgons orientation
%***** To keep the recursive function orient simple, globals are used


opgon=argpgon;                  % pass the argument to global
opgonlen=length(opgon);


maxpt=zeros(opgonlen,1);        % speed
onrpts=zeros(opgonlen,1);       % speed
for i=1:opgonlen
   maxpt(i)=max(opgon{i});
   onrpts(i)=length(opgon{i});  % number of points in opgon{i}
end
ptlen=max(maxpt);               % number of points alltogether


pt=fullpointinfo(ptlen,opgon,opgonlen,onrpts);
%***** To each point, this function calculates the polygons
%***** Containing it.

opgon=addfirstpoint(opgon);
%***** Adds the first point of the pgon to the
%***** end of the pgons's pointlist (convenient when referencing)

opgon=fulledgeinfo(pt,opgon,opgonlen,onrpts);
%***** To each polygon's edge, this function calculates the index
%***** of the adjacent polygon.

or=zeros(1,opgonlen);  % or=1 for oriented pgons, zero for not yet
oriented
or(1)=1;               % the first pgon is assumed oriented correctly
nrorpgons=1;           % hence the number of oriented pgons is one
orientation='pos';     % If not, orient will set it to 'none'

orient(1);             % this recursive function does all the work


for i=1:opgonlen
   opgon{i}=opgon{i}(1,1:onrpts(i));  % brings pgon into input form
end

if orientation(1)=='p'
   fprintf('\n Polyhedral mesh was already oriented. \n');
```

```
end


%************************************************************************
*
%****************************** SUBROUTINES
*****************************


function pt=fullpointinfo(ptlen,opgon,opgonlen,onrpts)
%***** To each point, this function calculates the polygons
%***** containing it.

pt=cell(ptlen,1);
for i=1:ptlen
   pt{i}=[];                  % reserve space
end

for i=1:opgonlen
   for j=1:onrpts(i)
      pgonij=opgon{i}(j);      % speed
      pt{pgonij}=[pt{pgonij} i];
   end                        % concatenate pgon-data i to
end                           % existing pt-data


%************************************************************************
*


function newpgon=addfirstpoint(opgon);
%***** Adds the first point of the pgon to the
%***** end of the pgons's pointlist (convenient when referencing)

for i=1:length(opgon)
   pgoni=opgon{i};
   newpgon{i}=[pgoni pgoni(1)];
end


%************************************************************************
*


function newpgon=fulledgeinfo(pt,opgon,opgonlen,onrpts)
%***** To each polygon's edge, this function calculates the index of the
%***** adjacent polygon.

%***** datastructure: newpgon{i} is a 2-by-onrpts+1 matrix, the first row
```

```
%***** Containing the points of which the polygon consists, the second the
%***** Indices of the adjacent polygons.

newpgon=cell(opgonlen,1);

for i=1:opgonlen
   pgoni=opgon{i};              % speed
   nrptsi=onrpts(i);            % speed
   row2=zeros(1,nrptsi);  % speed


   for j=1:nrptsi

         thispt=pgoni(j);          % speed,clearness
         nextpt=pgoni(j+1);        % speed,clearness

         adjpgon = intersection(pt{thispt},pt{nextpt},i);

                     % the index of the adjacent polygon
                     % can be determinded by intersecting the polygons
containing
                     % the first edgepoint with those containing the
second.

         if isempty(adjpgon)
            fprintf('\n An error has occured at edge %i \n',j);
            fprintf(' of polygon %i. Did not find an adjacent polygon.
\n\n',i);
            error('Input Data Error');
         end

         row2(j)=adjpgon;      % and set the value

   end
   newpgon{i}=[pgoni ; [row2 row2(1)] ];
end

function val=intersection(a,b,n)
%***** if a and b contain exactly two equal
%***** values, and one of them is given as n,
%***** this function calculates the other.
val=[];
for i=1:length(a)
   if a(i)~=n
      f=b(find(b==a(i)));
      if ~isempty(f)
         val=f(1);
         break;
      end
   end
end
```

```
%*************************************************************************
*



function orient(pgonnr)
%***** Actual orientation. Algorithm is as follows: Starting with the
first
%***** pgon we orient its neighboring pgons, then the neighboring
%***** pgons of these if not already oriented and so on.

global opgon opgonlen onrpts or nrorpgons orientation

notoriented=[];           % neighbouring pgons that are not yet oriented
pgonpgonnr=opgon{pgonnr};  % speed

for j=1:onrpts(pgonnr)

   adjpgon=pgonpgonnr(2,j);     % adjactent pgon

   if or(adjpgon)==0            % if adjactent pgon is not oriented
      notoriented=[notoriented adjpgon];  % put it in the list

      pgonadjpgon=opgon{adjpgon};      % speed
      thispt=pgonpgonnr(1,j);         % speed,clearness
      nextpt=pgonpgonnr(1,j+1);       % speed,clearness

      idxnextpt=findfirst(opgon{adjpgon}(1,:),nextpt);
      %***** the index of the next point in the adjactent polygon

      if pgonadjpgon(1,idxnextpt + 1) ~= thispt     % wrong orientation
         pgonadjpgon=fliplr(pgonadjpgon);           % reverse order
         pgonadjpgon(2,:)=[pgonadjpgon(2,2:onrpts(pgonnr)+1) 0];
                                                    % correct edge
information
         opgon{adjpgon}=pgonadjpgon;                 % and store the data
         orientation='none';
      end


      or(adjpgon)=1;                  % now the adjactent pgon is oriented
      nrorpgons=nrorpgons+1;          % and the number of oriented pgons
                                      % increases by one.

   end
end

if nrorpgons ~=opgonlen               % if not already finished
   for j=notoriented                  % orient the neighboring pgons
      orient(j);
   end
end

function first=findfirst(array,number)
%***** If array is known to contain number, this function returns
```

```
%***** the index of the first appearance of number in array.

i=1;
while 1
   if array(i)==number
      first=i;
      break;
   end
   i=i+1;
end

function varargout=pgondisp(pt,pgon)
% PGONDISP Display polyhedral mesh.
%
% [LI,PAT]=PGONDISP(PT,PGON) displays the polyhedral mesh
% defined by PT and PGON and returns a light handle LI and
% an array of patch handles PAT.
%
% PT should be an m-by-3 matrix containing the coordinates
% of the points, PGON should be a cell array of polygons, each
% polygon being a point list(of indices to points).
%
% Example(Dice):
%
% pt=[0 0 0;1 0 0;1 1 0;0 1 0;...
%        0 0 1;1 0 1;1 1 1;0 1 1];
% pgon={[1 2 3 4] [5 8 7 6] [1 5 6 2]...
%             [3 7 8 4] [2 6 7 3] [4 8 5 1]};
% pgondisp(pt,pgon);
%
% See also DOO PGONTRACE PGONORIENT CHSURF


%************************* INPUT ARGUMENT CHECK ************************
%***********************************************************************

error(nargchk(2,2,nargin));

if size(pt,2)~=3
   pt=pt';
end

if size(pt,2)~=3
   error(' pt should be an m-by-3 matrix');
end

%***************************** GRAPHICS ********************************
%***********************************************************************

axis off;
view(17,14);
rotate3d on;

pat=zeros(1,length(pgon));
for i=1:length(pgon)
```

```
   ptpgoni=pt(pgon{i},:);    % speed
   pat(i)=patch(ptpgoni(:,1) ,  ptpgoni(:,2) , ptpgoni(:,3) ,[1.0000
0.7812    0.4975],...
      'facelighting','flat','edgecolor',[0 0 0],...
      'edgelighting','flat','backfacelighting','lit');
end


li=light;
material dull;

%************************ OUTPUT ARGUMENT CHECK ***********************
%**********************************************************************

switch nargout
case 1
   varargout{1}=li;
case 2
   varargout{1}=li;
   varargout{2}=pat;
end

function varargout=pgontrace(pt,pgon,colorfile)
% PGONTRACE Raytraces a polyhedral mesh
%
% [LI,PAT,IVN]=PGONTRACE(PT,PGON[,COLORFILE]) ray traces the
% Polyhedral mesh defined by PT and PGON using Vertex normal-
% Interpolation and returns an array of light handles LI, an
% Array of patch handles PAT and the interpolated
% vertex normal IVN.
%
% PT should be an m-by-3 matrix containing the coordinates
% of the points, PGON should be a cell array of polygons, each
% polygon being a point list (of indices to points).
% COLORFILE is the name of an m-file that defines the color of
% a point on the surface in dependency of its coordinates
% and its normal vector. See COLORFILE for details.
%
% See also COLORFILE DOO PGONDISP


%************************ INPUT ARGUMENT CHECK ***********************
%**********************************************************************

error(nargchk(2,3,nargin));

if size(pt,2)~=3
   pt=pt';
end

if size(pt,2)~=3
   error(' pt should be a m-by-3 matrix');
end

if nargin==2
```

```
   colorfile='defaultcolormethod';
end


%************************** GRAPHICS *********************************
%*********************************************************************

map=get(gcf,'colormap');      % set finer colormap
newmap=map;
if size(map,1)<=64
   for i=1:size(map,1)-1
      newmap(4*i-3:4*i,:)=[map(i,:); 0.75*map(i,:)+0.25*map(i+1,:);...
                                     0.50*map(i,:)+0.50*map(i+1,:);...
                                     0.25*map(i,:)+0.75*map(i+1,:)];
   end
end
set(gcf,'colormap',newmap);


cla;
axis off;                                  % some graphic settings
set(gca,'Projection','perspective');
view(17,14);
rotate3d on;
li=light;


pat=zeros(length(pgon),1);                 % patch handles
vn =cell(1,length(pgon));                  % vertex normals

for i=1:length(pgon)

   pgoni=pgon{i};                          % speed
   ptpgoni=pt(pgoni,:);                    % speed
   pat(i)=patch(ptpgoni(:,1) ,  ptpgoni(:,2) , ptpgoni(:,3) ,[0 0
0],...%sum(ptpgoni')',...
               'FaceLighting','Phong','EdgeColor','None',...
               'EdgeLighting','None','Backfacelighting','unlit');

   vni=get(pat(i),'VertexNormal');         % get Vertexnormals

   for j=1:size(pgoni,2)
      vnij=vni(j,:);                  ,    % speed
      vn{i}(j,:)=vnij/norm(vnij);          % normalize Vertexnormals
   end
end

ptdata=fullpointinfo(pt,pgon);

ivn=zeros(length(pt),3);                   % interpolated Vertexnormal
colsize=length(feval(colorfile,[0 0 0],[1 1 1]));
col=zeros(length(pt),colsize);             % color of pt

for i=1:length(pt)
   ptdatai=ptdata{i};                                % speed
```

```
   ivni=vn{ptdatai(1,1)}(ptdatai(2,1),:);        % first vertex normal
   for j=2:size(ptdatai,2)
      ivni=ivni+vn{ptdatai(1,j)}(ptdatai(2,j),:);    % add Vertex normal
belonging
   end                                                % to the same point
   ivni=ivni/norm(ivni);                              % normalize,
   ivn(i,:)=ivni;                                     % and store them

   col(i,:)=feval(colorfile,pt(i,:),ivni);            % get color of pt
end

for i=1:length(pgon)
   set(pat(i),'VertexNormal',ivn(pgon{i}(:),:),...
              'FaceVertexCdata',col(pgon{i}(:),:),...
              'FaceColor','interp');
end



%************************* OUTPUT ARGUMENT CHECK ************************
%************************************************************************


switch nargout
case 1
   varargout={li};
case 2
   varargout={li pat};
case 3
   varargout={li pat ivn};
end


%***************************** SUBROUTINES ******************************
%************************************************************************


function newpt=fullpointinfo(pt,pgon)
%***** To each point, this function calculates the polygons
%***** Containing it, and the indices of the point in these polygons.

newpt=cell(length(pt),1);

for i=1:length(pgon)
   pgoni=pgon{i};             % speed
   for j=1:size(pgoni,2)
      pgonij=pgoni(j);        % speed
      newpt{pgonij}=[newpt{pgonij} [i;j]];
   end                        % concatenate pgon-data([i,j]) to
end                           % existing pt-data


function color=defaultcolormethod(pt,nv)
% ***** If no colorfile is specified, defaultcolormethod is used.
color=sum(nv);
```