SSSU 42
ISSN 0140 3818

COMPUTER AIDED DESIGN AND NUMERICALLY CONTROLLED MANUFACTURE OF A SPLIT MOULD FOR A COMPOSITE MODEL SHIP PROPELLER

by S.R. Turnock

Ship Science Report No. 42

December 1990

SSSU 42
ISSN 0140 3818

UNIVERSITY OF SOUTHAMPTON

DEPARTMENT OF SHIP SCIENCE

FACULTY OF ENGINEERING

AND APPLIED SCIENCE

COMPUTER AIDED DESIGN AND NUMERICALLY CONTROLLED MANUFACTURE OF A SPLIT MOULD FOR A COMPOSITE MODEL SHIP PROPELLER

by S.R. Turnock

Ship Science Report No. 42

December 1990

COMPUTER AIDED DESIGN
AND
NUMERICALLY CONTROLLED MANUFACTURE
OF A SPLIT MOULD FOR A
COMPOSITE MODEL SHIP PROPELLER

by

S.R. Turnock

UNIVERSITY OF SOUTHAMPTON

SHIP SCIENCE REPORT No. 42

December 1990

SUMMARY

A method is described of converting a standard table of offsets for a propeller design into a numerical description of the blade surface. This numerical surface description can then be used as a computer-aided design tool and also as a basis for producing data to drive a 3-axis numerically controlled (N.C.) machine tool for the manufacture of the blade or blade mould.

The report is divided into two main sections. The first section details the definition of the propeller numerical surface using quadratic parametric splines, and the elementary stress analysis and graphical output used in the development of a propeller design. The modified design was based on the Wageningen B4.40 propeller and was to be used for an investigation into ship propeller/rudder interactions. The second section covers the method by which the split female mould was N.C. machined for this modified propeller design.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

A	Section Area (m^2)
c	Chord of Section (m)
C_l	Two-Dimensional Lift Coefficient
D	Diameter of Propeller (m)
h_d	Distance from Leading Edge to Generator (m)
h_t	Distance from Maximum Thickness to Leading Edge (m)
P	Pitch Ratio = $\tan\phi/(2\cdot\pi\cdot r)$
r	Radius (m)
s	Arc Length (m)
t	Thickness (m)
X	Distance from maximum Thickness (%)
x,y,z	Orthogonal Cartesian Coordinate System
x_1, x_2, x_3	Cartesian Coordinate system in Section Plane
θ	angle subtended (s/r)
ϕ	Pitch Angle

1 INTRODUCTION

This report details the design and manufacture of a female split mould for the production of composite blades for a model ship propeller. The propeller blades were needed for a wind tunnel rig for studying interactions between ship propeller and rudder arrangements. Details of the rig are given in Turnock [1]. The overall diameter of the model propeller was 800mm with a maximum hub-to-propeller diameter ratio of 0.25. The propeller was to have four blades to give an overall blade area ratio of 0.4.

A representative propeller shape was chosen which was to be based on the Wageningen B-Series [2] propeller. This would allow experimental data for Torque, Thrust and efficiency obtained in the wind tunnel tests to be compared with previously published results. To give a variable propeller thrust loading for a given speed of revolution the blades were to incorporate a means for varying the pitch setting for both on and off design conditions.

The complex curves of a ship propeller require a highly accurate manufacturing technique to ensure a correct representation of the designed profile. Conventional methods of manufacture are expensive using materials such as bronze alloy and high precision casting and machining operations.

Following experience gained within the University of Southampton in the manufacture of carbon Fibre composite blades for a model wind turbine rig[3] it was decided to use a similar route for the manufacture of the ship propeller blades. The hybrid carbon/glass fibre composite design and manufacture of the blades themselves is detailed in Molland & Turnock[4]. The advantage in such a route lay primarily in the cost savings associated with the manufacturing technique. An additional benefit was that the composite blades could all be laid up using the same mould and would have an identical final blade shape. This compares with direct machining techniques of cast metal alloy blades where there are possibilities for variation in the final shape.

A local machining firm (Jason Engineering (Totton) Ltd.) had available a 3-axis Numerically Controlled (N.C.) vertical axis milling machine suitable for the manufacture of the split female mould surface. The machining of a male-plug and production of a female mould from it was considered. It was decided that it was simpler and more accurate to directly manufacture the female mould. Through the use of appropiately sized standard HSS mill cutters it was possible to accurately cut both the concave and convex surfaces present on the blade mould surface.

As input data the NC machine required a prepared data file. The production of these data files was carried out using a Nimbus 386VX personal computer. The use of the computer to detail the tool cutter paths for the production of the mould required an accurate numerical description of the propeller surface. An advantage of the numerical definition was that it allowed the final propeller profile to be refined and the effects of this refinement processs to be directly investigated.

The report is divided into the two main sections. Section 2 gives details of the mathematical and numerical representation of the propeller's three-dimensional surface. This numerical representation is used to produce a modified propeller design suitable for the wind tunnel tests using both elementary stress analysis and graphical representations. Section 3 is concerned with the actual manufacture of the blade moulds. The software for producing the NC data files is described. This used as a basis

the numerical representation described in Section 2. Finally, a description is given of the process of mould manufacture.

1.1 Software

All software described in the following two sections was developed using Borland TurboPascal Version 4[5]. This provided an integrated environment for developing, debugging and running the C.A.D. and N.C. programs written. This report will not detail the working of all the procedures included in these programs. However, where relevant, a direct reference will be made to the appropriate Procedure in a Program or Unit. A complete listing of all Programs and Units has been included as Appendix 1.

2 COMPUTER AIDED DESIGN OF PROPELLER

2.1 Introduction

Optimum propeller design requires that, for a given working fluid, the propeller will develop the maximum thrust for a minimum power input at given speed of revolution. The geometry of the propeller mount and surroundings constrain the overall shape of the propeller while the blades have to be made of a material which can withstand the resultant loading on the blades.

The working fluid for the Rudder and Propeller Interaction investigation was to be air. To provide a representative propeller for comparison with full scale the Wageningen B4-40 profile was chosen as a basis for the design of the model propeller. As a result of using the propeller in air rather than water to develop the same thrust loading the propeller revolutions had to be significantly increased. This changes the loading regime of the blades from one dominated by hydrodynamic forces to a loading controlled by centrifugal forces.

Due to the change in loading on the blade it was proposed to modify the shape of the Wageningen profile without significantly altering the aerodynamic performance of the propeller. By reducing the rake and sweep of the blade the moment about the propeller axis at the blade root due to the centripetal accelerations can be reduced. Also as the propeller was to be capable of being set at different pitches the blades were separate from the hub and so extra section thickness was required at the root to allow for blade attachment.

To investigate the influence of the modifications to the blade profile on the stress levels within the blade it was decided to write a computer program to carry out simple one-dimensional beam theory stress analysis of the proposed blade design. For this program the main requirement was an accurate numerical representation of the propeller blade surface. Once such a numerical definition is available it is then a straightforward exercise to carry out the stress analysis. It was also possible to provide various forms of graphical output for such things as accurate propeller drawings, meshes for more accurate finite element analysis, a surface definition for use in a theoretical lifting surface model, and also for isometric plots of the propeller surface.

Conventional description of propeller geometry is based on a cylindrical coordinate system which has been developed within Naval Architecture. The first stage of obtaining an accurate numerical description of the propeller surface was to transform the cylindrical coordinate into a conventional right-handed orthogonal cartesian system. The definition of the propeller in a cartesian coordinate system can then be directly used for the generation of Numerical Controlled cutting data files.

2.2 Definition of Propeller Shape

The accepted definition of a ship propeller as given in the ITTC Dictionary of Ship Hydrodynamics[6] is illustrated in Figure 1. The formal definition is based on defining blade sections at a given radial distance from the centre of rotation relative to a straight generator line with a defined rake forward or aft of a plane perpendicular to the axis of rotation. Each section has associated with it a pitch angle defined in terms of the pitch ratio.

There are three standard projections used in the graphical representation of propeller profiles:

1) Expanded - In this projection the cylindrical sections are represented as parallel straight lines at a distance equal to the radius with length equal to the section chord at that radius. This projection has no physical equivalent but it allows the section shape to be drawn as it would be seen on a cylindrical surface.

2) Developed - The sections are drawn on radial arcs about the generator origin with an arc-length equal to the section chord at that radius. This projection gives a view which is equivalent to a normal view of the local surface and independent of the local pitch angle

3) Projected - Each cylindrical section is defined at a given pitch angle and this projection gives the view obtained when each section is pitched to its correct pitch angle relative to the other sections.

Table 1 defines the Wageningen B-Series coordinates for a four-bladed propeller with a blade area ratio of 0.4. For each of the 10 sections six non-dimensional parameters are defined:

x - the radial distance divided by the maximum propeller radius (Diameter/2).
P/D - the pitch ratio P.
c/D - the total chord of the section.
h_d/D - the distance of the leading edge from the generator.
t/D - the maximum thickness
h_t/D - the position of the point of maximum thickness from the leading edge.

The section coordinates are given in terms of a percentage distance from the section maximum thickness based for the front (leading edge) on h_t and for the rear (trailing edge) on $(c - h_t)$. The position of the back and face coordinates is given as a fraction of the maximum section thickness t/D. For sections with a radiused leading edge extra information is required to define the nose details.

2.3 Transformation to Cartesian Co-ordinate System

The transformation of the propeller definition described in the previous section and based on pitched cylindrical sections into the more useful right-handed cartesian coordinate system requires the application of a series of simple transformations. This process is illustrated in Figure 2. The first stage for each section is to define a cartesian coordinate system with an origin at zero radius with x_1 out along the generator, x_3 vertically upwards and x_2 orthogonal to x_1 and x_3. The base of the cylindrical section lies in the origin plane formed by x_1 and x_2 along an arc at radius r from the origin so that the height of point P located on the propeller back or face is simply t. To determine the location of P in the x_1-x_2 plane the radial arc-length s from the generator line is required. The value of arc-length (Procedure Arc-length; Unit PropMenu) is defined as :

$$s = (h_d - h_t) + 0.01 * X * h_t \text{ for } X>0$$

$$s = (h_d - h_t) + 0.01 * X * (c - h_t) \text{ for } X<0$$

This arc-length s divided by the radius r defines an angle

$$\varphi = s / r$$

and this is used to give the value of P in x1,x2,x3 as

$$\overline{P} = \begin{bmatrix} r * \cos(\varphi) \\ r * \sin(\varphi) \\ t \end{bmatrix}$$

The final right handed coordinate system (x,y,z) has an identical origin to (x_1,x_2,x_3) with x coincident with x_1. The final transformation to give (x,y,z) is found by pitching (x_1,x_2,x_3) about x_1 by an angle equal to the pitch angle ϕ (Procedure PitchIt; Unit PropMenu).

$$\phi = \tan^{-1}((D * P) / (r * \pi))$$

The final transformation in matrix form is therefore:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Applying these transformations to the original propeller definition coordinates gives an exact cartesian representation of the propeller (Procedure XYZcoord; Unit PropMenu).

2.4 Modification to Original Propeller Sections

A standard input file was used which allows any propeller defined in the manner of Table 1 to be accessed by the computer program (Procedure DataReadIn; Unit PropMenu). Modifications to the overall chord length and maximum thickness of individual sections was carried out by directly editing the propeller definition input file prior to use by the program. However, the effective sweep and rake of the propeller can be modified by respectively changing the distance of the leading edge from the generator h_d and altering the position of the section baseline relative to the thickness.

The prime motive for altering the relative orientations of the sections was to reduce the moment at the blade root due to the centripetal acceleration. By determining the position of the centre of area of each section relative to the propeller generator line

(procedure FindCentroid; Unit PropMenu) the necessary changes in h_d and t for each section could be determined and adjusted accordingly (Procedure AdjustGenerator; Unit PropMenu). Where if s_c is the arc-length of the centroid from the generator and z_c the height of the section above the baseline the modified sections are given by

$$\begin{aligned} \hat{h_d} &= h_d - s_c \\ \hat{t} &= t - z_c \end{aligned}$$

The modification to hd and t, if used, is saved to an output file so that the actual section information used is recorded.

2.5 Interpolation of Propeller Surface.

The propeller back and face surfaces are only defined at 10 radii. Therefore, for intermediate locations some form of bidirectional interpolation is required to define the propeller surface. The surface definition has to be detailed enough for use in the N.C. machining of the blade or blade mould.

The procedure devised was a compromise between a straight linear interpolation and a full parametric cubic spline. There were insufficient points to define the complex curves using linear interpolation and it was judged the extra computational complexity of a parametric cubic spline was not needed. Instead, a parametric quadratic spline routine was developed (Procedure QuadSpline; Unit PropMenu). This requires as input three points only and the parametric equations to be solved are simple quadratic expressions which can be evaluated directly. Appendix 2 describes the operation of Quadspline.

The quadratic spline routine is designed to find points along a curve which have a given x, y or z. For the propeller surface two interpolations have to be carried out in the x and the y direction to define the surface height z. The first stage of the propeller surface definition was to define a complete section(y,z) at a given distance x from the origin (Procedure FindSection; Unit PropMenu). This was the most obvious method as the N.C. production of the mould was to be carried out by cuts in Y-Z planes. After definition of the two-dimensional section the height z of the face or back surface could be found for any y section within the section data. A special case was the nose where the face and back surfaces join. In this region there are two values for z for a given y and therefore the procedure was inverted and z defined and used to determine y. Figure 3 shows example two-dimensional sections generated by this method for various distances x from the origin for the unmodified Wageningen B4-40. It is interesting to note the characteristic ' S ' shape sections that are produced.

The procedure to generate a two-dimensional section at any distance from the origin is at the heart of the propeller surface definition. A robust procedure will allow a whole host of applications to be rapidly developed as detailed in the following sections.

2.6 Elementary Blade Stress Analysis

The propeller surface definition allowed elementary stress analysis to be carried out on the proposed modifications to the Wageningen B-series geometry. The two major components of load on the blade are that due to the aerodynamic lift generated by the blade and the centripetal acceleration which is a function of the blade mass distribution.

A simple one-dimensional beam theory is used in Naval Architecture to calculate the stress levels out along a propeller blade. The blade is represented as a straight beam with variable area and an assumed C_l distribution. An integration of the load distribution is then used to give the stress due to bending at a given section. The distribution used for lift along the blade has the form of (Procedure LiftForce; Unit PropMenu)

$$L(x) = \frac{k}{(x_{tip} - x_{hub})} \cdot \left[\frac{(x - x_{hub})}{(x_{tip} - x_{hub})}\right]^2 \cdot \sqrt{\frac{(x - x_{tip})}{(x_{tip} - x_{hub})}}$$

Changing the value of the constant k altering the total overall lift force generated by the blade.

A direct stress is generated by the revolution of the propeller and the centrifugal load at given section is simply the integration of the mass distribution outboard of that section. For a material with constant density the centrifugal load will be dependent on the volume of material. The area of a series of outboard sections is calculated (Procedure CalculateArea; Unit PropMenu) and a trapezium rule numerical integration will give the direct stress at a given section. A numerical integration of the area of each section can be used to calculate the total volume of the blade. The accuracy of the calculation will be increased with the use of additional sections.

As described earlier, one of the reasons for modifying the Wageningen geometry was to reduce the moment at the blade root due to centrifugal forces. A calculation of the centroid of a blade section allows the point of action of the centrifugal load to be calculated and hence the moment about the y and z axes can be evaluated.

The use of simple beam theory gives a conservative estimate of bending stresses in the blade and is useful as a preliminary design tool. The direct stress calculation is bounded by the accuracy of the numerical integration and the assumptions made about material density. It would be possible to generate a finite element mesh for a rigorous stress analysis using a commercial package such as ANSYS. For the purposes of the propeller design described in this report the elementary stress analysis was considered adequate.

2.7 Graphical Output

An important part of design work is the ability to visualise the modifications to the design as they occur. The numerical propeller surface definition provides an excellent basis for generating the data input files used by commercial packages, such as AutoCad, which allow three-dimensional shapes to be displayed. These packages allow surfaces to be viewed from different vantage points and at different resolutions so that fine surface details can be observed. Hard copy in the form of wire-frame diagrams also aids examination of the design.

A more standard from of output is the conventional Propeller drawing showing the expanded, developed and projected views of the blade. A data file for driving plotters using the HPGL language was produced and all of the propeller drawings used in this report were plotted using this means.

A surface plotting program (SPLOT) has been written within the Department of Ship Science and this was used to produce Figure 4 showing a view of a single blade and Figure 5 a view of a four-bladed propeller with hub. Different colours (or shades) can represent different heights of the surface segment above a datum. For example, it is also possible to use colours to represent surface stress levels.

2.8 C.A.D. Software

The procedures described in the preceding sections are contained in the TurboPascal Unit PropMenu. The actual calculation of stress, production of graphical data files, and modification of the propeller definition were carried out by the Program PropNC using the Unit PropMenu. It is a simple process to alter this program to investigate different properties of the propeller surface definition.

2.9 Final Propeller Design

Figure 6 shows the final outline of the modified Wageningen profile chosen. Table 2 lists the propeller definition used to produce this profile. The modifications investigated included altering:

1) Sweep (h_d) for the whole blade to reduce the centripetal moment.

2) Rake (z_c) for the whole blade to reduce the centripetal moment.

3) Both Sweep and Rake.

A comparison of the projected view of these three options and the original profile is shown as Figure 7. In the end it was decided to alter the rake by modifying zc. and only the sweep for the two innermost sections were altered as this gave a minimal change in shape while reducing the direct stress bending moment levels. Figure 8 shows a comparison of the direct stress levels, bending moment due to lift, and moments due to direct stress for the initial profile and the final design. The alteration of the sweep of the two innermost sections gives rise to the knee observed in Figure 7.

The propeller hub design described in Turnock[1] prescribed a propeller root chord of 90.0 mm at the radius of 0.25 of the maximum diametere(100mm). At this point the blade was to merge into a spherical cap of radius 100mm as shown in Figure 9. The spherical surface allowed the blade pitch to be varied without affecting the flow around the hub/blade root region. The propeller blade was to be manufactured using carbon/glass fibre composite material. This meant the blade root section thickness had to be increased to allow for the spreading of the fibres into the hub attachment stem. As shown in Figure 9 this stem had a minimum diameter of 60mm before belling out inside the hub. The maximum direct stress levels occur in this region.

To achieve a satisfactory merge between the blade, spherical cap and stem, four additional radial sections were defined (Procedure AddOnHandle; Unit PropMenu). This allowed the stress analysis to be carried through into the hub attachment region and also describing the mould surface for the NC cutting procedures.

3 NUMERICALLY CONTROLLED MANUFACTURE OF SPLIT MOULDS

3.1 Introduction

The modified Wageningen B4.40 propeller blade described in Section 2 was to be manufactured from hybrid carbon/glass fibre composite material[4]. For this process an accurate split female mould was required. This allowed four identical blades to be laid up using the same mould. Each blade was laid up separately in the two halves of the mould and then finally fixed together to produce the final blade. Once the epoxy matrix had cured the mould was removed leaving the finished blade.

To produce the two halves of the female mould a 3-axis N.C. vertical axis milling machine was to be used. This machine requires a prepared data file to drive the tool bed in the x, y and z directions, where the vertical direction z, is the direction of the milling cutter axis of rotation. The in-plane axes x and y form, with z, a left-handed orthogonal coordinate system. Although the tool itself remains in a fixed position in space, it is easier to describe the cutting motions as though the tool moves with the bed fixed and that is the convention followed in this report.

As described in the preceding section a numerical surface description of the propeller was available and it was using this description that the cutting method for the female moulds was developed.

3.1.1 NC Programming Language

Only a simple subset of the commands available for driving the N.C. machine tool were used in the data generation process. The commands for setting the tool datum, cutting speed and cutter RPM were added to the beginning of the file by the machine operator.

The data files used by the N.C. machine are in ASCII format and can be inspected and altered using any file editor. The tool cutter position can be specified either in mm (Real Number) or Microns (Integer) relative to the specified tool datum. In this case microns were used as the unit of movement. The two basic commands used were the absolute Move (G00) and Cut (G01). The Move command takes the cutter from its existing position to the new position specified with the command in absolute coordinates. During this process the cutter is not rotating and is moved at maximum speed in a straight line between the two points. The Move command should only be used when there is a guaranteed clear path. The Cut command causes the tool to spin at the designated RPM and follow a straight line between the two designated points at the set cutting speed.

3.2 Mould Cutting Strategy

The fundamental difficulty with manufacturing a female mould for a propeller blade is caused by the change in pitch angle out along the blade. For every two-dimensional section (y-z) the position of the intersection between the split line and the propeller section will be at a different height. By definition, the split line passes through the trailing edge of each section. However, at the leading edge of the blade the split line passes through the point on the section where the surface is locally vertical, which for a

pitched section, is not necessarily the defined leading edge. This constraint is applied so that the final blade can be removed from the mould.

The variation in pitch for the split-line of the propeller is shown in Figure 10. An advantage of the numerical surface description was that the propeller shape could be simply rotated around the x-axis. This rotation effectively reduced the absolute magnitude of the maximum pitch, thereby reducing the depth of material required for each half of the mould. The required angle of rotation for the blade was determined by minimising the depth of material required.

The non-planar split plane has to match identically on the two halves of the mould. The following three-stage strategy for manufacturing both halves of the mould was adopted:

1) Rough Cut: Excess material above the split-plane was removed using a flat bottomed tool with cuts in the longitudinal (x axis) direction. The position of the split plane was first determined and this was used to calculate the depth of cut to ensure the split-plane was not broached by the cutter.

2) Split Cut: A series of cuts in the transverse (y- axis) direction along the whole length of the blade were used to produce the final split plane. Outside the boundary of the propeller the split plane is kept horizontal. This substantially reduces the amount of material required for each half of the mould. This cut was carried out using a ball-nose cutter to give an accurate definition of the split-plane.

3) Final Cut: A series of cuts in the transverse direction following the geometry of the upper or lower surface of the propeller were to be made using a small diameter ball-nosed cutter to ensure the complex propeller shape was accurately reproduced.

3.3 Software

The Turbopascal program PropCut was used to generate the data files for carrying out the cuts for all three phases of the manufacture of the moulds. In addition to the Turbopascal Unit Propmenu already mentioned in this report, two additional Units PropMen2 and PropMen3 contain procedures used by PropCut. All of these procedures will not be described in detail, but the procedures used in transforming the numerical surface description of the propeller into a trajectory for the cutting tool will be referenced.

At the start of PropCut the size and shape of tool to be used are chosen and also the type of cutting operation (i.e. roughing, split plane or finish cut). Only standard ball-nose cutters where the radius of the spherical cutting surface is equal to the radius of the tool shaft, and flat bottomed cutters were used. This information is required to provide the offset between the actual point on the cutter which is cutting material and the tool position (Procedure FindToolTip; Unit PropMenu). For a ball-nosed cutter, the cutting point will be on the forward side of the tool where the local tangent to the surface and on the surface of the cutter are the same (see Figure 11).

The actual process of finding a section on the propeller surface and transforming the data into the series of positions for the tool to cut that section are similar for all three phases of the mould machining. The following steps are carried out:

1) Determine the coordinates on the face or back propeller surface, or the split-plane for the split-line and roughing cuts.

The procedure FindRoughProfile (Unit PropMen3) determines the maximum (for lower surface) or minimum (for upper surface) height on the split-plane for each transverse location. It does this by searching along the whole length (longitudinal) direction of the blade and determining z on the local split-line for every y. The split-line at a given x location is simply found in procedure CutSplit (Unit PropMen3) by determining the actual Trailing Edge and Leading Edge (Procedures FindTE,FindLE; Unit PropMenu) for the propeller surface.

2) Determine the profile of the existing cut surface.

For a roughing cut this will be the initial rectangular shape of the block from which the mould is to be machined. In the case of a split-line cut it will be that of the roughed profile and for the final cut the split-line.

3) Find the maximum depth of cut to produce the new surface.

This is simply found by subtracting the two section profiles and determining the maximum (Procedure FindMaxDepth; Unit Propmen2).

4) Determine the number of passes of the tool to cut the section so that the maximum depth of cut of each pass does not exceed a specified value.

5) For each individual pass add the correct height offset to the original section data so that pass will cut deeper until the final pass when the desired profile is produced.

6) For a final cut increase the definition of the section data by adding supplementary points along the section profile.

This is carried out by procedure NewCutLine (Unit PropMen2) which uses as a basis for determining the number of points to be added to the section the criteria that the tool cannot move more than a certain height increment for each individual movement. The procedure QuadSpline (Unit PropMenu) is used to generate the new points on the profile.

7) For each tool pass minimise wasted motions of the tool caused by always cutting in the same direction or cutting air.

This process involves ensuring that the section data is ordered in the opposite direction to that used in the last cut so that the tool cuts back and forth across the section

(Procedure reverseinfo; Unit propmen2). The section data is also modified to remove those parts of the trajectory which are trying to cut material which does not exist (Procedure FindReducedCut; Unit PropMen3)

8) Add the correct tool offset for the direction of travel and for the Back or Face surface of the propeller to the section data.

9) With the final modified section data for each pass the following steps are carried out on each data point:

i) Transform data from the propeller cartesian coordinate system into the machine cutting coordinate system relative to a defined datum. Procedure Translate (Unit PropMen3) carries this out. The transformation applied depends on whether it is the Back or Face mould which is being cut. It is crucial that this stage is correct, otherwise the split-plane may not meet or the propeller may be pitched the wrong way.

ii) Convert data from propeller surface definition (mm) into the machine units (microns) (Procedure Convert; Unit PropMen3).

10) As the section and pass data are produced in the machine coordinate system and units they are written to an ASCII file.

The number of sections stored in each data file is limited by the data storage capacity of the 3-axis N.C. machine (16KBytes) and also different tool sizes were used for different areas of the blade. The increment in x between sections dx for split-line and final cuts is determined from the set accuracy of definition of surface which was set to 0.1mm and the ball-nosed cutter diameter was as follows:

$$dx = 0.2 * \text{tooldiameter} - 0.01$$

Each data file stores a continuous stream of move and cut instructions for the cutting head with the aim being to minimise unnecessary tool movements and cuts.

3.4 Machining and Finishing of Moulds

As described in Section 2.9 there are three main parts to the final design of the propeller: cylindrical stem, spherical cap and the blade itself. The cylindrical stem was turned on a lathe and the cap and blade machined together. An aluminium alloy block was used for each half of the mould as shown in Figure 12. Aluminium alloy was used as it is, easy to machine, dimensionally stable and can be polished to give the smooth surface required for a mould surface. Each half of the mould had four 10mm diameter dowel holes, one of which was used as the machine datum. The machined lugs at either end of the block were used to clamp the block to the machine bed. The cylindrical stem was turned from a separate block of aluminium alloy which was then split into four along the vertical axis and at the position of the final root split-line. These four parts were bolted to the respective mould half. This allowed the blade to be made in two exact halves using a continuous split-plane, joined together using the dowels to exactly locate the moulds and then separated so that the blade could be removed as one piece.

About 70 data files were produced to use in the N.C. machining of the two halves of the moulds. A significant amount of time was needed in individually loading these files into the N.C. machine. The actual machining of the propeller surface into the mould took about 3 days work. The largest fraction of time being spent in the cutting of the final propeller surface using small diameter tools. The spherical cap and tip region were finish cut using a 6mm diameter tool whereas the rest of the blade was finish cut with a 10mm diameter tool.

The final surface of the mould was finished by hand using emery paper and a grinder. This was to remove the ridged surface left by the cutting process. A smaller step size between cuts would reduce the magnitude of the ridges at the expense of greatly increasing the overall cutting time.

Figures 13 and 14 show views looking towards the root and tip respectively of the finished moulds. These illustrate the complex shape of the twisted propeller blade and the change in split-plane out along the blade. Figure 15 shows a view of the finished hybrid composite blades produced using the mould.

4 CONCLUSION

The method of defining the numerical surface of the propeller from a standard propeller definition table of offsets, using quadratic parametric splines, provides an extremely versatile tool for the design of ship propeller blades. This method can be used at all stages in the design process from the preliminary design phase right through to the final manufacture of the blade or blade mould.

The difficulties of N.C. machining complex curves such as those present on a propeller blade should not be underestimated. However, the manufacture of a blade mould with both concave and convex surfaces has demonstrated that such shapes can be machined on a simple 3-axis N.C. machine using standard cutters if a sufficiently accurate surface description is provided.

ACKNOWLEDGEMENTS

The work described in this report covers part of a research project funded by the S.E.R.C./M.O.D. through the Marine Technology Directorate Ltd. under research grant Ref No GR/E/65289.

REFERENCES

[1] Turnock, S.R., "A Test Rig For The Investigation of Ship Propeller/Rudder Interactions", Ship Science Report No. 45, University of Southampton, November 1990.
[2] Oosterveld, M.W.C., & Oossanen, P. Van, "Further computer analysed data of the Wageningen B-screw series", International Shipbuilding Progress, Vol 22, No 251, July 1975.
[3] Barnsley, M.J., & Wellicome, J.F., "Design and Testing of a Horizontal Axis Wind Turbine Model for the Investigation of Stall Regulation Aerodynamics", Proceedings of the European Wind Energy Conference '89 pp724-728, Peter Peregrinus Ltd, 1989.
[4] Molland, A.F., & Turnock, S.R.,"The design and construction of propeller Blades in Composite Materials for a Wind Tunnel model", University of Southampton, Ship Science Report No. 41, November 1990
[5] ,"TURBO PASCAL 4.0", Borland International, 1987.
[6]ITTC Dictionary of Ship Hydrodynamics

APPENDICES

Appendix 1

PROGRAM PROPNC.PAS

```
program propNc;
uses inout,propmenu,propmen2,propmen3;
var
 Blk:BlockSize;
 finito,Offset:coord1;
 ledge,upper:boolean;
 tool:tooltype;
 Xstart,Xfinish,Xinc,MaxCut,DelZ,thi1,thi2,thi3:real;
 i,CutType:integer;
 videon,zplane,yplane,finished,CutFlag,MoveFlag:boolean;
 NCFile:text;
begin
 {= = = = = = = = = = = initialisation = = = = = = = = = = = = =}
 thi1:=0.0;
 thi2:=0.0;
 thi3:=-0.9;
 DataReadIn(diameter,Prop);
 For i:=2 to 10 do begin
  FindCentroid(i,Prop);
 end;
 yplane:=Question('   adjust y orientation of sections');
 zplane:=Question('   adjust z orientation of sections');
 adjustgenerator(yplane,zplane);
 handleit;
 finished:=false;

repeat
 {= = = = = = = = = = = cutting commences = = = = = = = = = =}
 BlockSetUp(Blk, Offset,upper,CutFlag,MoveFlag,MaxCut,delZ);
 ChooseCutType(CutType);
 ChooseToolType(tool);
 finito:=offset;
 {= = = = = = = = = = = open cut file    = = = = = = = = = =}

 OpenCutFile(CutType,NCfile);
 {= = = = = = = = = = = make cuts        = = = = = = = = = =}
 videon:=Question('  Do you want to see profiles as they are cut');
 CASE CutType of
 1:begin
    MakeRoughCut(videon,Blk,thi1,thi2,thi3,MaxCut,NCfile,CutFlag,MoveFlag,upper,
           offset,finito,tool);
  end;
 2:begin
    DefineCut(Blk,2,tool,Xstart,Xfinish,Xinc);
    MakeSplitCut(videon,Blk,thi1,thi2,thi3,MaxCut,
              NCfile,CutFlag,MoveFlag,upper,offset,finito,
                    tool,Xstart,Xfinish,Xinc,delZ);
  end;
 3:begin
   DefineCut(Blk,3,tool,Xstart,Xfinish,Xinc);
   MakeFinishCut(videon,Blk,thi1,thi2,thi3,MaxCut,NCfile,CutFlag,MoveFlag,upper,
                  offset,finito,tool,Xstart,Xfinish,Xinc,delZ);
  end;
 4:begin
   ledge:=Question('  Is this the leading edge to cut');
   DefineCut(Blk,4,tool,Xstart,Xfinish,Xinc);
```

```
    MakeEdgeCut(NCFile,offset,finito,CutFlag,MoveFlag,upper,ledge,
                    tool,Xinc,Xstart,Xfinish,thi1,thi2,thi3);
   end;
  end;

  {= = = = = = = = = = =  close cut file    = = = = = = = = = = =}

  Close(NCFile);
    finished:=Question('    HAVE YOU COMPLETED MAKING CUTS  ');
until finished=TRUE;
end.
```

UNIT PROPMENU

```
unit PropMenu;

interface

uses graph,crt;
{       series of procedures to generate data files to machine female mould
        for propeller blade.

                                written by S.Turnock
                        1/11/89}
{*********************************************************}
const
 diameter = 800.0;
type
 coord1   = record
                x,y,z : real;
               end;
 point    = record
                x,Yu,Yl :real;
               end;
 offset   = record
                y1,y2 : real;
               end;
 section = record
                radius : real;
                pitch,chord,Hd,Ht,thick,Hc,Zc : real;
                coord : array[1..15] of point;
                Nose : offset ;
               end;
 secty = array[1..300] of coord1;
 handy = array[1..5,1..30] of coord1;
 propdef = array[1..20] of section;

var
 prop:propdef;
 handit:handy;
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++

        PROCEDURES

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++}
PROCEDURE CutData(toolradius:real;

                        x,delZ,thi1,thi2,thi3 : real;
                orientation : boolean;   { upper surface = true }
                VAR  NP1 : integer;
                VAR dcuts:secty;
                VAR Intrude : boolean);

procedure QuadSpline( var ok,same : boolean; P1a,P2a,P3a : coord1;
                              val :real; typ : integer; var posta:coord1);

procedure pitchit(pitch,radius : real;var theta : real);
procedure arclength(X,hd,ht,chord:real; Var arc:real);
```

```
procedure xyzcoord(pitch,radius,arc,z,alp1,alp2,alp3 : real;
                                                    var Pcoord:coord1);
procedure writecoord(title:string;c1:coord1);
procedure findpoint(k,j : integer;thi1,thi2,thi3 : real;
                        upp:boolean;prop:propdef;var srt:coord1);
procedure DataReadIn(diameter: real;var prop : propdef);
procedure findcentroid(k :integer;var midd : propdef);
  procedure adjustgenerator(var prop:propdef;yplane,zplane:boolean);
procedure addonhandle;
procedure Generate3D;
  procedure FindSection(prop:propdef;  X,thi1,thi2,thi3:real;
                                      var dsec : secty;var xmax:real);
procedure drawsection(remsect,first,last:boolean;N:integer;dsec:secty);

  procedure drawYsection(remsect,first,last:boolean;N:integer;dsec:secty);
procedure drawtool(remsect,first,last:boolean;radius,diam :real;upper:boolean;
                        N:integer;dsec:secty);
procedure drawplan(remsect,first,last:boolean;N:integer;dsec:secty);
procedure QuadInterSec(NoTimes :integer;dsec :secty;var TotCount : integer;
                                                        var dnew:secty);
procedure Gen3d2(xstart,inc,thi1,thi2,thi3 :real;Nx:integer);
procedure handleit;

procedure removeclashs(upper:boolean;radius:real;
          prof:secty;npts:integer;var tool:secty;var npots:integer);
procedure splitlevel(x,thi1,thi2,thi3:real;
          var LE,TE :coord1;var radcurv :real);
procedure samepoint(c1,c2 : coord1;VAR answer : boolean);
procedure FindTE(x:real;dsec:secty;Npts : integer;var TE : coord1);
procedure checktoolclash(radius:real;
            upper:boolean;pos:coord1;
                  prof:secty;npts:integer;var good:boolean);
procedure cutline(x,delZ,thi1,thi2,thi3:real;var Npoints:integer;
             upper:boolean;var cutdata:secty);
procedure findtooltip(radius:real;upper : boolean;U1,U2,U3 : coord1;
               var pos :coord1);
procedure findprofile(radius,Xmin,Xmax,thi1,thi2,thi3:real;
               var cutle,cutTe : secty;
               var NTe,Nle : integer);
procedure beyondpoint(c1,c2,c3 : coord1;VAR answer : boolean);
procedure FindLE(x,xmax:real;dsec:secty;Npts : integer;var LE :coord1;
                        var radcurv,y0,z0 : real;
                        var locate:integer);
procedure RemoveDudPoints(var dsec:secty;var Npts : integer);
procedure calculatearea(x,thi1,thi2,thi3:real;var area,yy,zz,Kyy,Kzz:real);
procedure liftforce(X : real;var lift,torque:real);
procedure newfindtooltip(radius,diam:real;upper : boolean;U1,U2 : coord1;
               var pos1,pos2 :coord1;var samepoint,vertical:boolean);
IMPLEMENTATION
{*******************************************************}
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++'+++}
 procedure QuadSpline( var ok,same : boolean; P1a,P2a,P3a : coord1;
                              val :real; typ : integer; var posta:coord1);
  var
       a,b,c,P1,p2,p3,post : array[1..3] of real;
       L1,L2,T : real;
       t2,ta,tb,tt : real;
```

```
      i:integer;
  begin
   P1[1]:=P1a.x;
      P1[2]:=P1a.y;
      P1[3]:=P1a.z;
   P2[1]:=P2a.x;
      P2[2]:=P2a.y;
      P2[3]:=P2a.z;
      P3[1]:=P3a.x;
      P3[2]:=P3a.y;
      P3[3]:=P3a.z;
      same:=false;
      ok:=true;
      if (P1[1]=P2[1]) and (P1[2]=P2[2]) and (p1[3]=P2[3]) then same:=true;
      if (P2[1]=P3[1]) and (P2[2]=P3[2]) and (P2[3]=P3[3]) then same:=true;
      if same=false then begin
        L1:=sqrt(sqr(P2[1]-P1[1])+sqr(P2[2]-p1[2])+sqr(P2[3]-P1[3]));
        L2:=sqrt(sqr(P3[1]-P2[1])+sqr(P3[2]-P2[2])+sqr(P3[3]-P2[3]));
        T:=L1+L2;
        t2:=L1/T;
        For i:=1 to 3 do begin
         c[i]:=p1[i];
         a[i]:=((p3[i]-c[i])/(1-t2))-((p2[i]-c[i])/(t2*(1-t2)));
         b[i]:=p3[i]-a[i]-c[i];
        end;

        if sqr(b[typ])<(4*a[typ]*(c[typ]-val)) then begin
         ok:=false;
        end else begin
         if a[typ]<>0 then begin
     ta:=(-b[typ]-sqrt(sqr(b[typ])-(4*a[typ]*(c[typ]-val))))/(2*a[typ]);
     tb:=(-b[typ]+sqrt(sqr(b[typ])-(4*a[typ]*(c[typ]-val))))/(2*a[typ]);
         if (ta>=0) and (ta<=1) then
              tt:=ta
         else
              if (tb>=0) and (tb<=1) then tt:=tb else ok:=false;
    end else begin
         if b[typ]<>0 then tt:=-c[typ]/b[typ]
         else ok:=false;
        end;

        end;
        if ok=true then begin
          for i:=1 to 3 do post[i]:=a[i]*sqr(tt)+b[i]*tt+c[i];
          posta.x:=post[1];
          posta.y:=post[2];
          posta.z:=post[3];
        end;

      end;
      if same=true then ok:=false;

end;
procedure pitchit(pitch,radius : real;var theta : real);
  begin
      theta:=arctan(diameter*pitch/(2*radius*pi));
end;
procedure arclength(X,hd,ht,chord:real; Var arc:real);
```

```
  begin
       if x> =0.0 then
         arc:=(hd-ht) + 0.01*X*ht
       else
         arc:=(hd-ht) + 0.01*X*(chord-ht);
 end;
 procedure xyzcoord(pitch,radius,arc,z,alp1,alp2,alp3 : real;
                                                    var Pcoord:coord1);
  var
       thi,rb,x1,y1,z1 : real;
       x2,y2,z2  : real;
       x3,y3,z3  : real;
       x4,y4,z4 : real;
  begin


    rb:=sqrt(sqr(radius)-sqr(z));
       thi:=arc/radius;
       x1:=rb*cos(thi);
       y1:=rb*sin(thi);
       z1:=z;

    x2:=x1;
       y2:=y1*cos(pitch) - z1*sin(pitch);
       z2:=y1*sin(pitch) + z1*cos(pitch);

    {  rotate about z axis }
       x3:=x2*cos(alp1) - y2*sin(alp1);
       y3:=x2*sin(alp1) + y2*cos(alp1);
       z3:=z2;
       {rotate about y axis}
       x4:=x3*cos(alp2)-z3*sin(alp2);
       y4:=y3;
       z4:=x3*sin(alp2)+z3*cos(alp2);
       { rotate about x axis }
       Pcoord.x:=x4;
       Pcoord.y:=y4*cos(alp3) - z4*sin(alp3);
       Pcoord.z:=y4*sin(alp3) + z4*cos(alp3);
 end;
{*******************************************************}
 procedure findpoint(k,j : integer;thi1,thi2,thi3 : real;
                          upp:boolean;prop:propdef;var srt:coord1);
  var
       NoseRad,arc,zz,theta : real;
  begin
          pitchit(prop[j].pitch,prop[j].radius,theta);
          arclength(prop[j].coord[k].x,prop[j].hd,prop[j].ht,prop[j].chord,arc);
          if upp=false then zz:=prop[j].coord[k].yl else zz:=prop[j].coord[k].yu;
      if zz<-500.0 then begin

            zz:=(prop[j].nose.y1);

      end;

      XYZcoord(theta,prop[j].radius,arc,zz,
                  thi1,thi2,thi3,srt);
 end;
{*******************************************************}
```

```
procedure DataReadIn(diameter: real;var prop : propdef);
{  reads in data in standard format }
 var
      i,j,inp : integer;
      filename : string;
      filein : text;
 begin
      writeln;
      writeln('      Which data file do you wish to use');
      writeln;
      writeln('                NPL    (1):');
      writeln('     Vag - Origin        (2):');
      writeln('     Vag - Blade AR 0.35  (3):');
      writeln('     VAG - Blade AR 0.4   (4):');
      writeln('     Vag - Blade AR 0.4 <  (5):');
  write(' Enter choice 1-5 ');
      readln(inp);
      CASE inp of
        1:    filename:='C:\tpascal\nc\prop.dat';
        2:    filename:='C:\tpascal\nc\propvago.dat';
        3:    filename:='C:\tpascal\nc\propvag.dat';
        4:    filename:='C:\tpascal\nc\propvag2.dat';
        5:    filename:='C:\tpascal\nc\propvag3.dat';
      end;

  assign(filein,filename);
  reset(filein);
  For i:=1 to 10 do begin

   read(filein,prop[i].radius);
   read(filein,prop[i].pitch);
   read(filein,prop[i].chord);
   read(filein,prop[i].Hd);
   read(filein,prop[i].thick);
   readln(filein,prop[i].Ht);
   For j:=1 to 15 do begin
    read(filein,prop[i].coord[j].x);
        read(filein,prop[i].coord[j].yu);
        readln(filein,prop[i].coord[j].yl);
   end;


   read(filein,prop[i].nose.y1);
   readln(filein,prop[i].nose.y2);
  end;
  close(filein);
  For i:=1 to 10 do begin
      prop[i].radius:=diameter*0.5*prop[i].radius;

   prop[i].chord:=prop[i].chord*diameter;

      prop[i].Hd:=prop[i].Hd*diameter;

      prop[i].Ht:=prop[i].Ht*prop[i].chord;
      prop[i].thick:=prop[i].thick*diameter;

   writeln(i:8,prop[i].radius:10:6);
      For j:=1 to 15 do begin
```

```
          prop[i].coord[j].yu:=prop[i].coord[j].yu*prop[i].thick;

          prop[i].coord[j].yl:=prop[i].coord[j].yl*prop[i].thick;

         end;
         prop[i].nose.y1:=prop[i].nose.y1*prop[i].thick;
         prop[i].nose.y2:=prop[i].nose.y2*prop[i].thick;

    end;
end;
{*********************************************************}
 procedure findcentroid(k :integer;var midd : propdef);
 var
  Ym,Zm,arc1,arc2,pitch,area,Y2,Y1,Thi2,Thi1 : real;
  ii :integer;
  sumarea,sumareay,sumareaz : real;
 begin
  sumarea:=0.0;
  sumareay:=0.0;
  sumareaz:=0.0;
  For ii:=1 to 10 do begin
   arclength(midd[k].coord[ii].x,midd[k].Hd,midd[k].Ht,
                midd[k].chord,arc1);
   arclength(midd[k].coord[ii+1].x,midd[k].Hd,midd[k].Ht,
                midd[k].chord,arc2);
       thi1:=arc1/midd[k].radius;
       thi2:=arc2/midd[k].radius;
       Y2:=midd[k].radius*sin(thi2);
       Y1:=midd[k].radius*sin(thi1);
       area:=0.5*(Y2-Y1)*((midd[k].coord[ii].yu-midd[k].coord[ii].yl)+
                          (midd[k].coord[ii+1].yu-midd[k].coord[ii+1].yl));
       area:=abs(area);
       sumarea:=sumarea+area;
       sumareay:=sumareay+area*0.5*(Y2+Y1);
       sumareaz:=sumareaz+area*0.25*((midd[k].coord[ii].yu+midd[k].coord[ii].yl)+
                          (midd[k].coord[ii+1].yu+midd[k].coord[ii+1].yl));

  end;
  Ym:=sumareay/sumarea;
  Zm:=sumareaz/sumarea;
  midd[k].Zc:=Zm;
  thi1:=arctan(Ym/sqrt(sqr(midd[k].radius)+sqr(Ym)));
  midd[k].Hc:=thi1*midd[k].radius;
 end;
{*********************************************************}
 procedure adjustgenerator(var prop:propdef;yplane,zplane:boolean);
  var
       i,j : integer;
       delthi,thi1,thi2,hdmove : real;
  begin
       thi1:=prop[1].hd/prop[1].radius;
       thi2:=prop[2].hd/prop[2].radius;
       delthi:=thi2-thi1;
       hdmove:=(prop[10].hc)/prop[10].radius;

       For i:=2 to 10 do begin
```

```
        if yplane=true then prop[i].Hd:=prop[i].Hd-prop[i].Hc
        else if i>8 then prop[i].hd:=prop[i].hd-prop[i].hc;
        if zplane=true then begin
        For j:=1 to 15 do begin
          prop[i].coord[j].yu:=prop[i].coord[j].yu - prop[i].Zc;
          prop[i].coord[j].yl:=prop[i].coord[j].yl - prop[i].Zc;
        end;
        prop[i].nose.y1:=prop[i].nose.y1-prop[i].Zc;
        prop[i].nose.y2:=prop[i].nose.y2-prop[i].Zc;
        end;
       end;
       thi2:=prop[2].Hd/prop[2].radius;
       thi1:=thi2-delthi;
       if yplane=true then prop[1].Hd:=prop[1].radius*thi1;

       if zplane=true then prop[1].Zc:=0.0;
 end;
{*********************************************************}
 procedure addonhandle;
  var
   i,j:integer;
   pit,thi,mu,xx,rst : real;
  begin
        thi:=(prop[10].hd)/prop[10].radius;
        rst:=prop[10].radius*sin(thi);
    i:=12;
        prop[i].radius:=94.87;

    prop[i].pitch:=prop[10].pitch;

    prop[i].chord:=60.97;
        thi:=rst/prop[i].radius;

    prop[i].Ht:=0.5*prop[i].chord;
        prop[i].Hd:=(prop[i].radius*thi);


        prop[i].thick:=64.35;

        For j:=1 to 15 do begin
      prop[i].coord[j].x:=prop[10].coord[j].x;
         mu:=(0.5*prop[i].coord[j].x*0.01*prop[i].chord/prop[i].radius);
      xx:=prop[i].radius*sin(mu);
        writeln(xx:10:4,mu:10:4);
        prop[i].coord[j].yu:=sqrt(900.0-sqr(xx));

        prop[i].coord[j].yl:=-prop[i].coord[j].yu;

        end;


              i:=13;
        prop[i].radius:=50.0;

    prop[i].pitch:=prop[10].pitch;

    prop[i].chord:=64.35;
```

```
        thi:=rst/prop[i].radius;

    prop[i].Ht:=0.5*prop[i].chord;
        prop[i].Hd:=(prop[i].radius*thi);
        prop[i].thick:=64.35;

        For j:=1 to 15 do begin
     prop[i].coord[j].x:=prop[10].coord[j].x;
         mu:=(0.5*prop[i].coord[j].x*0.01*prop[i].chord/prop[i].radius);
     xx:=prop[i].radius*sin(mu);
         prop[i].coord[j].yu:=sqrt(900.0-sqr(xx));


         prop[i].coord[j].yl:=-prop[i].coord[j].yu;

        end;


              i:=11;
        prop[i].radius:=100.0;

    prop[i].pitch:=prop[10].pitch;
        thi:=rst/prop[i].radius;
    prop[i].chord:=90.207;
                 prop[i].Ht:=0.5*prop[i].chord;
        prop[i].Hd:=(prop[i].radius*thi);

        prop[i].thick:=43.59;

        For j:=1 to 15 do begin
     prop[i].coord[j].x:=prop[10].coord[j].x;

         mu:=(0.5*prop[i].coord[j].x*0.01*prop[i].chord/prop[i].radius);
     xx:=prop[i].radius*sin(mu);
         prop[i].coord[j].yu:=sqrt(1901.0-sqr(xx));


         prop[i].coord[j].yl:=-prop[i].coord[j].yu;

        end;

 end;
{*********************************************************}
 procedure Generate3D;
  var
       filename : string;
       fileout : text;
       srt : coord1;
       i,j,jl,ii : integer;
       theta,noserad,arc,thi1,thi2,thi3 : real;
  begin
       thi1:=0.0{-(prop[11].hd-prop[11].ht)/prop[11].radius};
       thi2:=0.0;
       thi3:=-pi/3;
       filename:='Shape.dat';
       assign(fileout,filename);
       rewrite(fileout);
       writeln(fileout,'   14   30');
```

```
        For ii:=2 to 5 do begin
      i:=7-ii;
         For j:=1 to 30 do begin
          writeln(fileout,handit[i,j].x:15:4,handit[i,j].y:15:4,
                  handit[i,j].z:15:4);
         end;
        end;

    For jl:=1 to 10 do begin
         j:=11-jl;
         For i:=1 to 15 do begin


         findpoint(i,j,thi1,thi2,thi3,false,prop,srt);
         writeln(fileout,srt.x:15:4,srt.y:15:4,srt.z:15:4);
     { writeln(srt.x:15:4,srt.y:15:4,srt.z:15:4);
         writeln(sqrt(sqr(srt.x)+sqr(srt.y)+sqr(srt.z)):15:4);}
         end;
         For ii:=1 to 15 do begin
         i:=16-ii;
      FindPoint(i,j,thi1,thi2,thi3,true,prop,srt);
      writeln(fileout,srt.x:15:4,srt.y:15:4,srt.z:15:4);

         end;
        end;
        close(fileout);
 end;
{*********************************************************}
 procedure FindSection(prop:propdef;  X,thi1,thi2,thi3:real;
                                      var dsec : secty;var xmax:real);
  var
        jj,j,k : integer;
        upp,find,ok,same : boolean;
        zz,theta,rap,arc,yip,noserad,y0,z0 : real;
        srt,P1,P2,P3,srt1,srt2,blih : coord1;
  begin
   findpoint(7,1,thi1,thi2,thi3,true,prop,blih);
        Xmax:=blih.x;
        if x<Xmax then begin
        if x>=100.0 then begin
         For jj:=1 to 30 do begin
          if jj>15 then upp:=true else upp:=false;
          k:=9;
          if jj>15 then j:=31-jj else j:=jj;
          find:=false;
          repeat
               findpoint(j,k,thi1,thi2,thi3,upp,prop,srt);
       if srt.x>X then find:=true else k:=k-1;
          until (find=true) or (k<1);
      if find=false then begin
               dsec[jj].x:=-1000.0;
               dsec[jj].y:=-1000.0;
               dsec[jj].z:=-1000.0;
          end else begin
               if k=1 then begin
                P1.x:=srt.x;
                P1.y:=srt.y;
                P1.z:=srt.z;
```

```
            findpoint(j,2,thi1,thi2,thi3,upp,prop,P2);
            findpoint(j,3,thi1,thi2,thi3,upp,prop,P3);
          end else begin
            P2.x:=srt.x;
            P2.y:=srt.y;
            P2.z:=srt.z;
            findpoint(j,k-1,thi1,thi2,thi3,upp,prop,P1);
            findpoint(j,k+1,thi1,thi2,thi3,upp,prop,P3);
          end;
   Quadspline(ok,same,P1,P2,P3,X,1,srt);
          dsec[jj].x:=srt.x;
          dsec[jj].y:=srt.y;
          dsec[jj].z:=srt.z;
     end;
    end;
   end else begin
    if x<90.0 then begin
     For jj:=1 to 30 do begin

   P1:=handit[5,jj];
          P2:=handit[4,jj];
          P3:=handit[3,jj];
          Quadspline(ok,same,P1,P2,P3,X,1,srt);
          dsec[jj]:=srt;
     end;
    end else begin

Findpoint(1,10,thi1,thi2,thi3,false,prop,srt1);
Findpoint(15,10,thi1,thi2,thi3,false,prop,srt2);
z0:= 0.5*(srt1.z+srt2.z);
y0:= 0.5*(srt1.y+srt2.y);

  For jj:=1 to 30 do begin
      if jj>15 then begin
          j:=31-jj;
          upp:=true;
      end else begin
          j:=jj;
          upp:=false;
      end;
          P2:=handit[1,jj];
          P3:=handit[2,jj];
          FindPoint(j,10,thi1,thi2,thi3,upp,prop,P1);
   rap:=sqrt(sqr(100.0)-sqr(x));
          yip:=p1.y;
          if (x>=p2.x)  or (abs(yip-y0)>abs(rap)) then begin
            FindPoint(j,8,thi1,thi2,thi3,upp,prop,P3);
            FindPoint(j,9,thi1,thi2,thi3,upp,prop,P2);
            FindPoint(j,10,thi1,thi2,thi3,upp,prop,P1);
    Quadspline(ok,same,P1,P2,P3,X,1,srt);
          end else begin
            srt.x:=x;
            srt.y:=yip;

            if jj<16 then
              srt.z:=-sqrt((sqr(rap)-sqr(yip-y0)))
            else srt.z:=sqrt(sqr(rap)-sqr(yip-y0));
   end;
```

```
                dsec[jj]:=srt;
            end;

          end
        end;
        end else begin
          for jj:=1 to 30 do begin
            if jj>15 then upp:=true else upp:=false;
            if jj>15 then j:=31-jj else j:=jj;
       k:=2;
            findpoint(j,k,thi1,thi2,thi3,upp,prop,srt);
            dsec[jj].x:=x;
            dsec[jj].y:=srt.y;
            dsec[jj].z:=blih.z;

          end;
        end;
end;
{*********************************************************}
procedure drawsection(remsect,first,last:boolean;N:integer;dsec:secty);
  var
        i: integer;
        scalex,scaley:real;
        xasp,yasp :word;
        dataI : array[1..302,1..2] of integer;
  begin
    scaleX:=2.56;
        Getaspectratio(xasp,yasp);
        scaleY:=(ScaleX*yasp)/(xasp);
        if (remsect=false) or (first=true) then begin

    SetGraphMode(2);
    SetBkColor(0);

    SetColor(11);
        end;
for i:=1 to N do begin
   dataI[i,1]:=round(getmaxX /2)+round(scaleX*dsec[i].y);
   dataI[i,2]:=round(getMaxY /2)-round(scaleY*dsec[i].z);
   putpixel(dataI[i,1],dataI[i,2],15);

end;

moveto(DataI[1,1],DataI[1,2]);
for i:=2 to N do lineto(dataI[i,1],dataI[I,2]);

 if (remsect=false) or (last=true) then begin
    outtextxy(50,50,' Please press space bar');
    repeat until readkey=' ';
    dataI[N+1,1]:=datai[1,1];
        dataI[N+1,2]:=dataI[1,2];
    SetFillstyle(1,11);
    FillPoly(N+1,dataI);
repeat until readkey=' ';
restorecrtmode;
end;
end;
```

```
{*********************************************************}
 procedure drawYsection(remsect,first,last:boolean;N:integer;dsec:secty);
  var
       i: integer;
       scalex,scaley:real;
       xasp,yasp :word;
       dataI : array[1..302,1..2] of integer;
  begin
    scaleX:=1.06;
       Getaspectratio(xasp,yasp);
       scaleY:=(ScaleX*yasp)/(xasp);
       if (remsect=false) or (first=true) then begin

    SetGraphMode(2);
    SetBkColor(0);

    SetColor(11);
       end;
 for i:=1 to N do begin
   dataI[i,1]:=50+round(scaleX*dsec[i].x);
   dataI[i,2]:=round(getMaxY /2)-round(scaleY*dsec[i].z);
   putpixel(dataI[i,1],dataI[i,2],15);

 end;

 moveto(DataI[1,1],DataI[1,2]);
 for i:=2 to N do lineto(dataI[i,1],dataI[I,2]);

  if (remsect=false) or (last=true) then begin
     outtextxy(50,50,' Please press space bar');
     repeat until readkey=' ';
     dataI[N+1,1]:=datai[1,1];
        dataI[N+1,2]:=dataI[1,2];
     SetFillstyle(1,11);
     FillPoly(N+1,dataI);
 repeat until readkey=' ';
 restorecrtmode;
 end;
 end;



{*********************************************************}
 procedure drawtool(remsect,first,last:boolean;radius,diam:real;upper:boolean;
                N:integer;dsec:secty);
  var
       i,rad : integer;
       scalex,scaley:real;
       xasp,yasp :word;
       dataI : array[1..302,1..2] of integer;
       dim,dang,fang,sang:integer;
  begin
    scaleX:=2.56;
       rad:=round(scaleX*radius);
```

```
        if rad<0 then rad:=0;
        Getaspectratio(xasp,yasp);
        scaleY:=(ScaleX*yasp)/(xasp);
        if (remsect=false) or (first=true) then begin
    SetGraphMode(2);
    SetBkColor(0);

    SetColor(11);
        end;
        SetColor(7);
        if rad<>0 then begin
        if radius=0.5*diam then dang:=90
        else begin
          dang:=round((180/pi)*arctan(radius/(sqrt(sqr(radius)-sqr(0.5*diam)))));

        end;
        if upper=false then begin
          sang:=270-dang;
          fang:=270+dang;
        end else begin
          sang:=90-dang;
          fang:=90+dang;
        end;
        end else dim:=round(0.5*diam*ScaleX);
 for i:=1 to N do begin
   dataI[i,1]:=round(getmaxX /2)+round(scaleX*dsec[i].y);
   dataI[i,2]:=round(getMaxY /2)-round(scaleY*dsec[i].z);
   if upper=true then dataI[i,2]:=dataI[i,2]+rad
   else dataI[i,2]:=dataI[i,2]-rad;
   putpixel(dataI[i,1],dataI[i,2],15);
   if rad<>0 then
     arc(dataI[i,1],dataI[i,2],sang,fang,rad)
   else begin
        moveto(dataI[i,1]-dim,dataI[i,2]);
        lineto(dataI[i,1]+dim,dataI[i,2]);
   end;
 end;

 SetColor(11);


  if (remsect=false) or (last=true) then begin
     outtextxy(50,50,' Please press space bar');
     repeat until readkey=' ';
     dataI[N+1,1]:=datai[1,1];
        dataI[N+1,2]:=dataI[1,2];
     SetFillstyle(1,11);

 repeat until readkey=' ';
 restorecrtmode;
 end;
 end;



{*********************************************************}
 procedure drawplan(remsect,first,last:boolean;N:integer;dsec:secty);
  var
```

```
        i: integer;
        scalex,scaley:real;
        xasp,yasp :word;
        dataI : array[1..302,1..2] of integer;
  begin
    scaleX:=1.0;
        Getaspectratio(xasp,yasp);
        scaleY:=(ScaleX*yasp)/(xasp);
        if (remsect=false) or (first=true) then begin
    SetGraphMode(2);
    SetBkColor(0);

    SetColor(11);
        end;

 for i:=1 to N do begin
   dataI[i,1]:=round(0.85*getmaxX)+round(scaleX*dsec[i].y);
   dataI[i,2]:=round((2*getMaxY) /3)-round(scaleY*dsec[i].z);
   putpixel(dataI[i,1],dataI[i,2],15);

 end;

 moveto(DataI[1,1],DataI[1,2]);
 for i:=2 to N do lineto(dataI[i,1],dataI[I,2]);
 for i:=1 to N do begin
   dataI[i,1]:=round(scaleX*dsec[i].x);
   dataI[i,2]:=round((2*getMaxY) /3)-round(scaleY*dsec[i].z);
   putpixel(dataI[i,1],dataI[i,2],15);

 end;

 moveto(DataI[1,1],DataI[1,2]);
 for i:=2 to N do lineto(dataI[i,1],dataI[I,2]);
 for i:=1 to N do begin
   dataI[i,1]:=round(scaleX*dsec[i].x);
   dataI[i,2]:=round(getMaxY /3)-round(scaleY*dsec[i].y);
   putpixel(dataI[i,1],dataI[i,2],15);

 end;

 moveto(DataI[1,1],DataI[1,2]);
 for i:=2 to N do lineto(dataI[i,1],dataI[I,2]);

  if (remsect=false) or (last=true) then begin
     outtextxy(50,50,' Please press space bar');
     repeat until readkey=' ';
     dataI[N+1,1]:=datai[1,1];
         dataI[N+1,2]:=dataI[1,2];
     SetFillstyle(1,11);
     FillPoly(N+1,dataI);
 repeat until readkey=' ';
 restorecrtmode;
 end;
 end;



{*********************************************************}
```

```
procedure QuadInterSec(NoTimes :integer;dsec :secty;var TotCount : integer;
                                                        var dnew:secty);
 var
 L1,delL,z,L,Lp,dely,delz,y : real;
 fraction,val :real;
 i,jj,j,k : integer;
 total,typ ,count: integer;
 ok,same :boolean;
 D1,D2 : secty;
 srt : coord1;
 begin
      i:=1;
      total:=totcount;
      count:=0;

      repeat


   if (dsec[i].x=dsec[i+1].x) then
       if (dsec[i].y=dsec[i+1].y) then
            if (dsec[i].z=dsec[i+1].z) then
              same:=true
            else
              same:=false
       else same:=false
      else
       same:=false;
      if same=true then dsec[i].x:=-1000.0;
   if (dsec[i].x=-1000.0) then begin
       For j:=i to total-1 do begin
            dsec[j].x:=dsec[j+1].x;
            dsec[j].y:=dsec[j+1].y;
     dsec[j].z:=dsec[j+1].z;
    end;
       total:=total-1;
      end else begin
       i:=i+1;
       count:=count+1;
      end;
 until i>=total;

 dsec[total+1].x:=dsec[1].x;
 dsec[total+1].y:=dsec[1].y;
 dsec[total+1].z:=dsec[1].z;

 TotCount:=total+1;

 For i:=1 to TotCount do D1[i]:=dsec[i];

 For k:=1 to NoTimes do begin
      For j:=1 to TotCount-1 do begin
   if j=TotCount-1 then begin
       z:=0.5*(D1[j].z+D1[j+1].z);
    y:=0.5*(D1[j].y+D1[j+1].y);
{            writeln(' P1 ',D1[j].x:15:4,D1[j].y:15:4,D1[j].z:15:4);
            writeln(' P2 ',D1[j+1].x:15:4,D1[j+1].y:15:4,D1[j+1].z:15:4);
     writeln(' P3 ',D1[j-1].x:15:4,D1[j-1].y:15:4,D1[j-1].z:15:4);}
```

```
          if abs(z)>abs(y) then
        QuadSpline(ok,same,D1[j-1],D1[j],D1[j+1],y,2,srt)
          else
           Quadspline(ok,same,D1[j-1],D1[j],D1[J+1],z,3,srt);
          D2[(2*j)-1]:=D1[j];
      if ok=false then
           D2[(2*j)].x:=-1000.0
          else
           D2[(2*j)].x:=srt.x;
          D2[(2*j)].y:=srt.y;
          D2[(2*j)].z:=srt.z;
          D2[(2*(j+1)-1)]:=D1[j+1];

         end else begin
          z:=0.5*(D1[j].z+D1[j+1].z);
          y:=0.5*(D1[j].y+D1[j+1].y);

{              writeln(' P1 ',D1[j].x:15:4,D1[j].y:15:4,D1[j].z:15:4);
               writeln(' P2 ',D1[j+1].x:15:4,D1[j+1].y:15:4,D1[j+1].z:15:4);
      writeln(' P3 ',D1[j+2].x:15:4,D1[j+2].y:15:4,D1[j+2].z:15:4);
   }
          if abs(z)>abs(y) then
        QuadSpline(ok,same,D1[j],D1[j+1],D1[j+2],y,2,srt)
          else
           Quadspline(ok,same,D1[j],D1[j+1],D1[j+2],z,3,srt);

          D2[(2*j)-1]:=D1[j];
          if ok =false then
           D2[(2*j)].x:=-1000.0
          else
           D2[(2*j)].x:=srt.x;
          D2[(2*j)].y:=srt.y;
          D2[(2*j)].z:=srt.z;
        { writeln;
         writeln(((2*j)-1):4,(2*j):4,D2[(2*j)-1].y:15:4,D2[(2*j)-1].z:15:4,
                                              d2[(2*j)].y:15:4,d2[(2*j)].z:15:4);}
         end;
        end;
        TotCount:=(2*TotCount)-1;
   For j:=1 to TotCount do D1[j]:=D2[j];
  end;

 i:=1;
       total:=Totcount;
       count:=0;
       repeat


    if (d2[i].x<-900.0) then begin
         For j:=(i+1) to total do begin
              d2[j-1].x:=d2[j].x;
              d2[j-1].y:=d2[j].y;
      d2[j-1].z:=d2[j].z;
     end;
         total:=total-1;
       end else begin
        i:=i+1;
```

```
            count:=count+1;
          end;
    until i>=total;
    totcount:=total;

    For j:=1 to TotCount do dnew[j]:=D2[j];
  end;
procedure writecoord(title:string;c1:coord1);
  begin
    write(title);
    writeln(c1.x:10:4,c1.y:10:4,c1.z:10:4);
end;


{*****************************************}
 procedure Gen3d2(xstart,inc,thi1,thi2,thi3 :real;Nx:integer);
   var
        Ny : integer;
        oop :text;
        i,j:integer;
        dsec,dnew : secty;
     x,xmax:real;
   begin

    Ny:=30;
    assign(oop,'shape1.dat');
    rewrite(oop);
    writeln(oop,Nx:8,Ny:8);

    for i:=1 to Nx do begin

      x:=xstart+(i-1)*inc;


      Findsection(prop,x,thi1,thi2,thi3,dsec,xmax);
      For j:=1 to Ny do
          writeln(oop,dsec[j].x:15:4,dsec[j].y:15:4,dsec[j].z:15:4);
    end;
    close(oop);

 end;
procedure handleit;
 var
  i,j,jj : integer;
  thi1,thi2,thi3,radius,sgn,theta,xx,z0,y0 : real;
  theta1,theta2,dtheta2,dtheta1:real;
  srt1,srt2 : coord1;
 begin
   thi1:=0.0;
   thi2:=0.0;
   thi3:=-pi/3;
   prop[10].hd:=prop[10].hd-4.575;
   Findpoint(1,10,thi1,thi2,thi3,false,prop,srt1);
   Findpoint(15,10,thi1,thi2,thi3,false,prop,srt2);


   z0:= 0.0;
   y0:= 0.5*(srt1.y+srt2.y);
```

```
    For i:=1 to 15 do begin
        Findpoint(i,10,thi1,thi2,thi3,false,prop,srt1);
        Findpoint(i,10,thi1,thi2,thi3,true,prop,srt2);
        handit[1,i]:=srt1;
        handit[1,31-i]:=srt2;
    end;
        theta1:=pi+arctan((handit[1,1].z-z0)/(handit[1,1].y-y0));
        theta2:=2*pi+arctan((handit[1,15].z-z0)/(handit[1,15].y-y0));
        dtheta1:=(theta1-(theta2-2*pi))/14;
        dtheta2:=(2*pi-theta1)/14;
    For j:=1 to 4 do begin
        Case j of
         1:begin
           radius:=45.4;
           xx:=90.0;
          end;
         2:begin
           radius:=30.0;
           xx:=90.0;
           end;
         3:begin
           radius:=30.0;
           xx:=60.0;
           end;
         4:begin
           radius:=40.0;
           xx:=45;
           end;
        end;

        For i:=1 to 30 do begin
          if i<16 then begin {upper side}
       theta:=theta1+(i-1)*dtheta2;
       handit[j+1,i].y:=y0+radius*cos(theta);
           handit[j+1,i].z:=z0+radius*sin(theta);
          end else begin    {lower side}
     theta:=theta2+(i-16)*dtheta1-2*pi;

           handit[j+1,i].y:=y0+radius*cos(theta);
           handit[j+1,i].z:=z0+radius*sin(theta);
         end;

         handit[j+1,i].x:=xx;


        end;
    end;


end;
{*********************************************************}
procedure FindTE(x:real;dsec:secty;Npts : integer;var TE : coord1);
 var i:integer;
 begin
  if x>=100.0 then begin
   If dsec[1].y<dsec[Npts].y then
        TE:=dsec[1]
   else
```

```
        TE:=dsec[Npts];
   end else begin
     TE :=dsec[1];
     For i:=2 to Npts do begin
          If dsec[i].y<TE.y then TE:=dsec[i];
     end;
   end;

end;
{*********************************************************}
procedure FindLE(x,xmax:real;dsec:secty;Npts : integer;var LE :coord1;
                          var radcurv,y0,z0 : real;
                          var locate:integer);
  var
    i,j,midd :integer;
    a,b,c,Ymax,Zmin:real;
    P1,P2,P3 : coord1;
    found : boolean;
  begin
    found:=false;
    if x<xmax then begin
    Ymax:=dsec[2].y;
    midd:=2;

    For i:=3 to Npts-1 do begin
          if dsec[i].y>=Ymax then begin
        midd:=i;
            Ymax:=dsec[i].y;
          end;
    end;



    P1:=dsec[midd-1];
    P2:=dsec[midd];
    P3:=dsec[midd+1];

    locate:=midd-1;

    { find arc intersection }

    a:=((sqr(p3.y)-sqr(p1.y))+(sqr(p3.z)-sqr(p1.z)))/(2*(p3.z-p1.z));
    b:=((sqr(p2.y)-sqr(p1.y))+(sqr(p2.z)-sqr(p1.z)))/(2*(p2.z-p1.z));
    c:=((p1.y-p2.y)/(p2.z-p1.z))-((p1.y-p3.y)/(p3.z-p1.z));
    y0:=(a-b)/c;

    z0:=((sqr(p2.y)-sqr(p1.y))+2*y0*(p1.y-p2.y)+(sqr(p2.z)-sqr(p1.z)));
    z0:=z0/(2*(p2.z-p1.z));

    radcurv:=sqrt(sqr(p1.z-z0)+sqr(p1.y-y0));

    { set LE coords }

    LE.x:=dsec[i].x;
    LE.y:=Y0 + radcurv;
    LE.z:=Z0;
   end else begin
     LE.x:=x;
```

```
    LE.y:=dsec[round(Npts/2)].y;
    LE.z:=dsec[round(Npts/2)].z;
   end;

end;
{*********************************************************}
procedure RemoveDudPoints(var dsec:secty;var Npts : integer);
 var
  i,j,k,total,totcount,count :integer;

 begin
  count:=0;
  For i:=1 to Npts-1 do begin
       if (dsec[i].y=dsec[i+1].y) and (dsec[i].z=dsec[i+1].z) then begin

                dsec[i].x:=-1000.0;
    end;
       if dsec[i].x<-900.0 then count:=count+1;

  end;

  total:=npts;

  i:=1;
  totcount:=count;
  count:=0;
  j:=1;
  For i:=1 to Npts do begin
       if dsec[j].x<-900.0 then begin

        For k:=j to Total-1 do begin
         count:=count+1;
         dsec[k]:=dsec[k+1];

        end;
        total:=total-1;
       end else begin
        j:=j+1;
       end;
 end;
 Npts:=j-1;


end;
{*********************************************************}
procedure calculatearea(x,thi1,thi2,thi3:real;var area,yy,zz,Kyy,Kzz:real);
 var
 dsec:secty;
 darea,yarm,zarm,sumarea,sumM1y,sumM1z,thi,xmax :real;
 j,i:integer;
 begin
  Findsection(prop,x,thi1,thi2,thi3,dsec,xmax);

  sumarea:=0.0;
  sumM1y:=0.0;
  sumM1z:=0.0;
  Kyy:=0.0;
  kzz:=0.0;
```

```
    For I:=1 to 14 do begin
     j:=31-i;
     darea:=(dsec[j-1].y-dsec[i].y)*(dsec[j].z-dsec[i+1].z)*0.5;
     darea:=darea-0.5*((dsec[j].y-dsec[i+1].y)*(dsec[j-1].z-dsec[i].z));
     sumarea:=sumarea+abs(darea);
     Yarm:=0.25*(dsec[j].y+dsec[j+1].y+dsec[i].y+dsec[i+1].y);
     Zarm:=0.25*(dsec[j].z+dsec[j+1].z+dsec[i].z+dsec[i+1].z);
     sumM1y:=sumM1y+Yarm*darea;
     sumM1z:=Zarm*darea+sumM1z;
     Kyy:=Kyy+sqr(Yarm)*darea;
     Kzz:=Kzz+sqr(Zarm)*darea;
    end;
    area:=sumarea;
    yy:=sumM1y/sumarea;
    zz:=sumM1z/sumarea;
    {parallel axis theorem}
    Kyy:=kyy-sqr(yy)*area;
    Kzz:=Kzz-sqr(yy)*area;

end;
procedure liftforce(X : real;var lift,torque:real);
  const
    X0 = 90.0;
    X1 = 400.0;
    kap1 = 1230.5;
    kap2 = 180.47;
  var
    XX :real;
  begin
   if x>90.0 then begin
    XX:=(X-X0)/(X1-X0);
    lift:=kap1*sqr(xx)*sqrt(1.0-xx);
    torque:=kap2*sqr(xx)*sqrt(1.0-xx);
   end else lift:=0.0;
   lift:=lift/(X1-X0);
   torque:=torque/(X1-X0);

end;
{*********************************************************}
procedure splitlevel(x,thi1,thi2,thi3:real;
          var LE,TE :coord1;var radcurv :real);
  var
    dsec:secty;
    Npoints,locate:integer;
    y0,z0,xmax:real;
  begin
    Findsection(prop,x,thi1,thi2,thi3,dsec,xmax);
    NPoints:=30;
    RemoveDudPoints(dsec,Npoints);
    FindTe(x,dsec,npoints,TE);
    FindLe(x,xmax,dsec,Npoints,LE,radcurv,y0,z0,locate);
end;
{*********************************************************}
procedure samepoint(c1,c2 : coord1;VAR answer : boolean);
  begin
    answer:=false;

    IF (c1.x=c2.x) then begin
```

```
        if c1.y=c2.y then begin
          if c1.z=c2.z then begin
            answer:=true;
          end;
        end;
  end;


end;
{*********************************************************}
procedure beyondpoint(c1,c2,c3 : coord1;VAR answer : boolean);
 begin
  answer:=false;
  IF (c2.y>=c1.y) and (c3.y<=c2.y) then
       if ((c1.z<c2.z) and (c2.z<c3.z)) or ((c1.z>c2.z) and (c2.z>c3.z)) then
          answer:=true;

end;
{*********************************************************}
procedure findprofile(radius,Xmin,Xmax,thi1,thi2,thi3:real;
                var cutle,cutTe : secty;
                var NTe,Nle : integer);
 var
 Nop,i,locate,Npts,plan:integer;
 delx,x,radcurv,y0,z0,xmx : real;
 Le,Te : coord1;
 dsec,blip1,blip2:secty;
 begin
  Nop:=round((Xmax-Xmin)/radius);
  delx:=(Xmax-Xmin)/(Nop-1);
  For i:=1 to Nop do begin
       x:=Xmin+(i-1)*delx;
    Findsection(prop,x,thi1,thi2,thi3,dsec,xmx);
    Npts:=30;
    RemoveDudPoints(dsec,Npts);
    FindTE(x,dsec,Npts,TE);
    FindLE(x,xmx,dsec,Npts,LE,radcurv,y0,z0,locate);
    cutle[i]:=le;
       cutte[i]:=te;
  end;
  DrawPlan(true,true,false,Nop,Cutle);
  DrawPlan(true,false,true,Nop,Cutte);
  NTe:=Nop;
  Nle:=Nop;
end;
{*********************************************************}
procedure newfindtooltip(radius,diam:real;upper : boolean;U1,U2 : coord1;
                var pos1,pos2 :coord1;var samepoint,vertical:boolean);
 var
 r,t,d :coord1;
 sgn,tm,ts,blip:real;
 procedure writeoutpos(mess:string;pos:coord1);
  begin
       writeln(mess,pos.x:15:6,pos.y:15:6,pos.z:15:6);
 end;
 begin
  vertical:=false;
  t.x:=U2.x-U1.x;
```

```
    t.y:=U2.y-U1.y;
    t.z:=U2.z-U1.z;
    d.x:=0.0;
    d.y:=0.0;
    if 0.5*diameter< radius then
      blip:=radius-sqrt(sqr(radius)-sqr(0.5*diameter))
    else if radius>=0.0 then blip:=radius else blip:=0.0;

    if upper=true then d.z:=blip else d.z:=-blip;
    ts:=sqrt(sqr(t.x)+sqr(t.y)+sqr(t.z));
    if ts=0 then samepoint:=true else
    begin
    samepoint:=false;
    if upper=true then sgn:=1.0 else sgn:=-1.0;
    if radius>=0.0 then begin
      tm:=sqrt(sqr(t.x)+sqr(t.y));
          if tm>0 then begin

        r.x:=-radius*sgn*t.z*t.x/(tm*ts);
            r.y:=-radius*sgn*t.z*t.y/(tm*ts);
            r.z:=radius*sgn*tm/ts;
            pos1.x:=U1.x-r.x+d.x;
            pos1.y:=U1.y-r.y+d.y;
            pos1.z:=U1.z-r.z+d.z;
            pos2.x:=U2.x-r.x+d.x;
            pos2.y:=U2.y-r.y+d.y;
            pos2.z:=U2.z-r.z+d.z;
          end else begin
        vertical:=true;

          end;
    end else begin {flat bottomed tool}
      tm:=sqrt(sqr(t.x)+sqr(t.y));
          if tm>0.0 then begin
      r.z:=0.0;
            r.x:=0.5*diameter*sgn*t.z*t.x/(tm*ts);
            r.y:=0.5*diameter*sgn*t.z*t.x/(tm*ts);
            pos1.x:=U1.x-r.x;
            pos1.y:=U1.y-r.y;
            pos1.z:=U1.z-r.z;
            pos2.x:=U2.x-r.x;
            pos2.y:=U2.y-r.y;
            pos2.z:=U2.z-r.z;
          end else begin
            vertical:=true;
          end;
    end;
    end;
end;
{*********************************************************}
procedure findtooltip(radius:real;upper : boolean;U1,U2,U3 : coord1;
                  var pos :coord1);
  var
  dx,dy,th1,th2,dT1,dT2,L,dz1,dz2,dzdt : real;
  begin
    dt1:=sqrt(sqr(u2.x-u1.x)+sqr(u2.y-u1.y));
    dt2:=sqrt(sqr(u3.x-u2.x)+sqr(u3.y-u2.y));
    dz1:=u2.z-u1.z;
```

```
  dz2:=u3.z-u2.z;
  dzdt:=0.5*((dz1/dt1)+(dz2/dt2));
  th1:=arctan(dzdt);
  if upper = true then
   pos.z:=U2.z + radius*(1-cos(th1))
  else
      pos.z:=U2.z - radius*(1-cos(th1));
  dx:=U2.x-U1.x;
  dy:=u2.y-U1.y;
  L:=radius*sin(th1);
  if upper =true then begin
   pos.x:=U2.x + L*dx/dt1;
   pos.y:=U2.y + L*dy/dt1;
  end else begin
   pos.x:=U2.x - L*dx/dt1;
   pos.y:=U2.y - L*dy/dt1;
  end;

end;
procedure cutline(x,delZ,thi1,thi2,thi3:real;var Npoints:integer;
          upper:boolean;var cutdata:secty);
 var
  ztry:real;
  ytry,radcurv,xmax,zip,ActZ,y0,z0:real;
  dsec,dcuts:secty;
  npots,Npts,cob,index,count,position,i,noop,locate : integer;
  TE,LE,P1,P2,P3,srt :coord1;
  ans,ans1,ok,same : boolean;
  MaxZ,MinZ,StartZ,FinZ,dipZ,wipZ : real;
 begin
  wipZ:=0.1;
  Findsection(prop,x,thi1,thi2,thi3,dsec,xmax);
  Npts:=30;
  RemoveDudPoints(dsec,Npts);
  FindTE(x,dsec,Npts,TE);
  FindLE(x,xmax,dsec,Npts,LE,radcurv,y0,z0,locate);
  If upper=true then begin
      dcuts[1]:=TE;
      index:=2;
      npots:=1;
      count:=Npts;
      samepoint(dsec[count],TE,ans);
      IF ans=false then begin
       dcuts[index]:=dsec[count];
       index:=index+1;
       count:=count-1;
       npots:=npots+1;
      end else count:=count-1;
   ans:=false;

      repeat
       if count>(locate+4) then
        ans:=false
       else begin

        beyondpoint(dsec[count],LE,dsec[count-1],ans);

    end;
```

```
        IF ans=false then begin
          dcuts[index]:=dsec[count];
          index:=index+1;
          npots:=npots+1;
          count:=count-1;
        end else begin
          samepoint(dsec[count],LE,ans1);
          IF ans1=false then begin
                dcuts[index]:=LE;

                Npots:=Npots+1;

          end else begin

                dcuts[index]:=LE;

                Npots:=Npots+1;

          end;
        end;


  until (ans=true) or (count<1);
  if count<2 then begin
        writeln('   program crash count out of range');
        readln;
      end;
end else begin
  dcuts[1]:=TE;
      index:=2;
      npots:=1;
      count:=1;
      samepoint(dsec[count],TE,ans);
      IF ans=false then begin
        dcuts[index]:=dsec[count];
        index:=index+1;
        count:=count+1;
        npots:=npots+1;
      end else count:=count+1;
      ans:=false;
      repeat
        if count<locate then
          ans:=false
        else begin
          beyondpoint(dsec[count],LE,dsec[count+1],ans);

        end;
        IF ans=false then begin
          dcuts[index]:=dsec[count];
          index:=index+1;
          npots:=npots+1;
          count:=count+1;
        end else begin
          samepoint(dsec[count],LE,ans1);
          IF ans1=false then begin
                dcuts[index]:=le;
       Npots:=Npots+1;
```

```
          end else begin

               dcuts[index]:=LE;

               Npots:=Npots+1;

          end;
        end;

  until (ans=true) or (count>Npts);
  if count>Npts-1 then begin
       writeln('  program crash count out of range');
       readln;
      end;
end;

{  Drawsection(false,true,true,Npots,dcuts);  }
{   generate   actual cut coordinates }

startZ:=dcuts[1].z;
FinZ:=dcuts[Npots].z;
MinZ:=StartZ;
MaxZ:=StartZ;
position:=1;
For i:=2 to Npots do begin
      if dcuts[i].z>MaxZ then begin
        MaxZ:=dcuts[i].z;
        if upper=true then position:=i;
      end;
      if dcuts[i].z<MinZ then begin
        MinZ:=dcuts[i].z;
        if upper=false then position:=i;
      end;
end;
if upper=true then begin
      dipZ:=MaxZ-StartZ;
 if dipZ>0.0 then begin
       if position=2 then begin
         Npoints:=2;
         CutData[1]:=dcuts[1];
         CutData[2]:=dcuts[2];
         writeln('  hoorah');
       end else begin
      Noop:=round(dipZ/delZ);
      if noop<1 then noop:=2 else noop:=noop+1;
      ActZ:=dipZ/(noop-1);
      CutData[1]:=dcuts[1];
      CutData[noop]:=dcuts[position];
      Npoints:=noop-1;

      For i:=2 to noop-1 do begin
        zip:=StartZ + (i-1)*ActZ;
        cob:=0;
        Repeat
         cob:=cob+1
        until (zip>=dcuts[cob].z) and (zip<dcuts[cob+1].z);

        P1:=dcuts[cob];
```

```
        P2:=dcuts[cob+1];
        P3:=dcuts[cob+2];

        QuadSpline(ok,same,P1,P2,P3,zip,3,srt);

        CutData[i]:=srt;
      end;
      end;
      end else begin
        CutData[1]:=dcuts[1];
    Npoints:=1;
      end;
  dipZ:=MaxZ-FinZ;
      if dipZ>0 then begin
      Noop:=round(dipZ/delZ);
      if noop<1 then noop:=2 else noop:=noop+1;

      ActZ:=dipZ/(noop-1);
      CutData[Npoints+noop-1]:=dcuts[Npots];

      For i:=2 to (noop-1) do begin
        zip:=MaxZ - (i-1)*ActZ;
        cob:=0;
        Repeat
         cob:=cob+1
        until (zip<=dcuts[cob].z) and (zip>dcuts[cob+1].z);
        if cob<2 then cob:=2;
        P1:=dcuts[cob-1];
        P2:=dcuts[cob];
        P3:=dcuts[cob+1];
        QuadSpline(ok,same,P1,P2,P3,zip,3,srt);
        CutData[Npoints+i-1]:=srt;
      end;
  Npoints:=Npoints+noop-1;
      end else begin
        Npoints:=Npoints+1;
        CutData[NPoints]:=dcuts[Npots];
      end;
end else begin
      dipZ:=StartZ-MinZ;
 if dipZ>0.0 then begin
      Noop:=round(dipZ/delZ);
      if noop<1 then noop:=2 else noop:=noop+1;
      ActZ:=dipZ/(noop-1);
      CutData[1]:=dcuts[1];

      CutData[noop]:=dcuts[position];
      Npoints:=noop-1;
      For i:=2 to noop-1 do begin
        zip:=StartZ - (i-1)*ActZ;
        cob:=0;
        Repeat
         cob:=cob+1
        until (zip<=dcuts[cob].z) and (zip>dcuts[cob+1].z);
        if cob>=1 then cob:=cob+1;
        if (cob>=position) and (cob>2) then cob:=position-1;
        if zip<MinZ then minZ:=Zip;
        P1:=dcuts[cob-1];
```

```
          P2:=dcuts[cob];
          P3:=dcuts[cob+1];
          QuadSpline(ok,same,P1,P2,P3,zip,3,srt);
          CutData[i]:=srt;
        end;
        end else begin
          CutData[1]:=dcuts[1];
          Npoints:=1;
        end;
    dipZ:=FinZ-MinZ;
   If dipZ>0.0 then begin
        Noop:=round(dipZ/delZ);
        if noop<1 then noop:=2 else noop:=noop+1;
        ActZ:=dipZ/(noop-1);
        CutData[Npoints+noop-1]:=dcuts[Npots];
         For i:=2 to (noop-1) do begin
          zip:=MinZ + (i-1)*ActZ;
          cob:=0;
          Repeat
           cob:=cob+1
          until (zip>=dcuts[cob].z) and (zip<dcuts[cob+1].z);
          if cob<2 then cob:=2;
          P1:=dcuts[cob-1];
          P2:=dcuts[cob];
          P3:=dcuts[cob+1];
          QuadSpline(ok,same,P1,P2,P3,zip,3,srt);
          CutData[Npoints-1+i]:=srt;

        end;
     Npoints:=Npoints+noop-1;
     end else begin
          writeln('  Yabba dabba doo');
          Npoints:=Npoints+1;
          CutData[Npoints]:=dcuts[Npots];
        end;
   end;


end;
{*********************************************************}
procedure checktoolclash(radius:real;
            upper:boolean;pos:coord1;
                  prof:secty;npts:integer;var good:boolean);
 var
   i,j : integer;
   Ymin,Ymax,limit,y,z,z0,dz:real;
 begin
   limit:=0.1;  { max error 0.1mm }
   good:=true;
   Ymin:=pos.y-radius;
   Ymax:=pos.y+radius;
   For i:=1 to npts do begin
        If (prof[i].y>=Ymin) and (prof[i].y<=Ymax) then begin
     if upper=true then z0:=pos.z-radius else z0:=pos.z+radius;
        y:=prof[i].y-pos.y;
        z:=sqrt(sqr(radius)-sqr(y));
     if upper=true then begin
          z:=z+z0;
```

```
          if z>(prof[i].z+limit) then good:=false;
        end else begin
          z:=z0-z;
          if z<(prof[i].z-limit) then good:=false;
        end;

      end;

  end;

end;
procedure removeclashs(upper:boolean;radius:real;
          prof:secty;npts:integer;var tool:secty;var npots:integer);
 var
  i:integer;
  good : boolean;
  pos:coord1;
  start,finish:boolean;
  I1,I2 : integer;
  limit : real;
 begin
  limit:=0.1;
  I1:=1;
  I2:=Npots;
  start:=false;
  finish:=false;
  For i:=1 to npots do begin
      pos:=tool[i];
      checktoolclash(radius,upper,pos,prof,npts,good);
      if good=false then begin
        pos.y:=pos.y-limit;
    checktoolclash(radius,upper,pos,prof,npts,good);
        if good=true then tool[i]:=pos;
      end;
      if start=false then begin
        if good=true then begin
          I1:=i;
          start:=true;
        end;
      end else begin
        if finish=false then begin
          if good=false then begin
              I2:=i;
              finish:=true;
          end;
        end;
      end;
  end;
  For i:=I1 to I2 do tool[(i+1-I1)]:=tool[i];
  npots:=I2-I1+1;
end;
{-------------------------------------------------------------}

PROCEDURE CutData(toolradius:real;

                  x,delZ ,thi1,thi2,thi3: real;
            orientation : boolean;   { upper surface = true }
            VAR  NP1 : integer;
```

```
                VAR dcuts:secty;
                VAR Intrude : boolean);

{ Procedure to generate coordinates for the location of the base of the
  cutter at its centre }

VAR
 i,NumPts      : integer;
 pos,U1,U2,U3 :coord1;
 CutData : secty;

BEGIN
   writeln('  reached cutline ');
   cutline(x,delZ,thi1,thi2,thi3,NumPts,orientation,cutdata);

   writeln('  out of cutline');


   For i:=1 to NumPts-2 do begin
         U1:=Cutdata[i];
         U2:=Cutdata[i+1];
         U3:=Cutdata[i+2];
         findtooltip(toolradius,orientation,U1,U2,U3,pos);
         dcuts[i]:=pos;
   end;
   Np1:=NumPts-2; '
   removeclashs(orientation,toolradius,CutData,NumPts,dcuts,Np1);



   {drawtool(true,true,false,toolradius,orientation,Np1,dcuts);
   DrawSection(true,false,true,NumPts,Cutdata);}


END; { of PROCEDURE CutDat }
Var Grdriver,Grmode : integer;
begin
 GrDriver:=9;
 GrMode:=2;
 initgraph(GrDriver,GrMode,'c:\tpascal');
 restorecrtmode;
end.
```

PROGRAM PROPCUT

```
program PropCut;
uses propmen2,propmenu,crt,util,plotter;
{       program to generate data files to machine female mould
        for propeller blade.

                                written by S.Turnock
                                        30/8/89}
{*******************************************************}
{*******************************************************}
var
 xmx,cutterrad,cutterdiam:real;
 tnp,Np2,Np1,locate,i,NoSec,totcount,Npoints1,Npoints2 : integer;
 tool1,tool2,dsec,dnew,cutdata1,cutdata2 : secty;
 delZ,torque,delX,
 thi1,thi2,thi3,arm,Ftot,My,Mz,volume,area,yy,zz,radcurv,x,xinc,starting : real;
 TE,LE,U1,U2,U3,pos : coord1;
 blodge,yplane,zplane,handle,first,last,pling : boolean;
 Y,Liftarm,TotalBM,TotalLift,lift,Kyy,Kzz,density,stress,rpm:real;
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++}

procedure plotplan(typ:Integer;N:integer;dsec:secty;scaleit :Scale_Rec);

 var
 scalex,scaley:real;
 xasp,yasp :word;
 idiot :Scale_Rec;
 i,ix,iym,iyx,izx,izm:integer;
 Ymax,Ymin,Zmax,Zmin :real;
 begin

  { draw XZ }

    scaleX:=1.0;

        scaleY:=1.0;
 CASE typ OF
 1:begin

 P_moveto(dsec[1].x,dsec[1].z,ScaleIt);
 for i:=2 to N do P_lineto(dsec[i].x,dsec[i].z,ScaleIt);

 end;
  {draw YZ}
  2:begin

 P_moveto(dsec[1].z,dsec[1].y,ScaleIt);
 for i:=2 to N do P_lineto(dsec[i].z,dsec[i].y,ScaleIt);
 end;
  {draw XY}
 3:begin

 P_moveto(dsec[1].x,dsec[1].y,ScaleIt);
 for i:=2 to N do P_lineto(dsec[i].x,dsec[i].y,ScaleIt);
 end;
 4:begin
```

```
    ix:=round(ScaleIt.X_off + dsec[1].x*ScaleIt.X_Sca);
    Ymax:=dsec[15].y;
    Ymin:=dsec[1].y;
    iyx:=round(scaleIt.Y_off + Ymax*ScaleIt.Y_Sca);
    iym:=round(scaleIt.Y_off + Ymin*ScaleIt.Y_Sca);
    Zmax:=-1000.0;
    Zmin:=1000.0;
    For i:=1 to N do begin
        If dsec[i].z>Zmax then Zmax:=dsec[i].z
        else if dsec[i].z<Zmin then Zmin:=dsec[i].z;

    end;
    iZm:=round(ScaleIt.Y_Sca*Zmin);
    iZx:=round(Scaleit.Y_Sca*Zmax);
    writeln(Ymin:10:2,Ymax:10:2,Zmax:10:2,Zmin:10:2);
    writeln(ix:10,iZm:10,iZx:10,iym:10,iyx:10);readln;
    Set_P_Scale((ix-iZm),iym,(ix+iZx),iyx,Zmin,Ymin,Zmax,Ymax,idiot);
  P_Moveto(dsec[1].z,dsec[1].y,idiot);

  for i:=2 to N do P_lineto(dsec[i].z,dsec[i].y,Idiot);
  P_MoveTo(0.0,Ymin,Idiot);
  P_lineto(0.0,Ymax,Idiot);

  end;
 end;
 writeln(' reached end of plot plan');
end;
procedure plotprofile(thi1,thi2,thi3:real);
  var
  Xmin,Xmax,Ymin,Ymax,Zmin,Zmax,dpx,dpx2,xmx :real;
  ijk,NoSections :integer;
  fname:string;
  CutData1,CutData2,dsec :secty;
  npoints1,npoints2:integer;
  XYScale,YZScale,XZScale:Scale_Rec;
  begin
    fname:='n:PlotProp.Dat';
    Write('  Enter Minimum X (X>45.0)');readln(Xmin);
    Write('  Enter Maximum X (X<400.0)');readln(Xmax);
    Write('  Enter X increment       ');readln(dpX);
    write('  Enter No of Sections');readln(NoSections);
    findprofile(dpX,Xmin,Xmax,thi1,thi2,thi3,Cutdata1,cutdata2,npoints1,npoints2);

    Set_P_Scale(250,250,4250,1250,Xmin,-50.0,Xmax,50.0,XZScale);
    Set_P_Scale(250,1750,4250,3750,Xmin,-100.0,Xmax,100.0,XYScale);
    Set_P_Scale(4750,1750,5750,3750,-50.0,-100.0,50.0,100.0,YZScale);
    P_Start(fname,4.0,0,0,6000,4000);
    P_DrawBox(0,0,6000,4000);

    PlotPlan(1,npoints1,CutData1,XZScale);
    PlotPlan(1,npoints2,CutData2,XZScale);
    PlotPlan(2,npoints1,CutData1,YZScale);
    PlotPlan(2,npoints2,CutData2,YZScale);
    PlotPlan(3,npoints1,CutData1,XYScale);
    PlotPlan(3,npoints2,CutData2,XYScale);
    writeln('  reached end of here ');
    if NoSections>0 then begin
```

```
   dpX2:=(Xmax-Xmin)/NoSections;

   For ijk:=0 to (NoSections-1) do begin
        X:=Xmin+(ijk*dpX2);
        FindSection(x,thi1,thi2,thi3,dsec,xmx);
        PlotPlan(4,30,dsec,XYSCale);
   end;
   end;
   P_Close;

end;
{*********************************************************}
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++}
 procedure GenSplit3d(xstart,inc,thi1,thi2,thi3 :real;Nx:integer);
   var
        Ny : integer;
        oop :text;
        i,j:integer;
        dsec,dnew : secty;
    x:real;
   begin

    Ny:=4;
    assign(oop,'split1.dat');
    rewrite(oop);
    writeln(oop,Nx:8,Ny:8);

    for i:=1 to Nx do begin

     x:=xstart+(i-1)*inc;
     cutsplit(x,-100.0,100.0,thi1,thi2,thi3,dsec,Ny);

     For j:=1 to Ny do
         writeln(oop,dsec[j].x:15:4,dsec[j].y:15:4,dsec[j].z:15:4);
    end;
    close(oop);

 end;

{-----------------------------------------------------------}
begin
 thi1:=0.0;
 thi2:=0.0;
 thi3:=-0.9;
 DataReadIn(diameter,Prop);
 For i:=2 to 10 do begin
  FindCentroid(i,Prop);
 end;
 yplane:=Question('   adjust y orientation of sections');
 zplane:=Question('   adjust z orientation of sections');
 adjustgenerator(yplane,zplane);
 handleit;
 pling:=Question('    Do you wish to alter rotations');
 if pling=true then begin
  repeat
   writeln('  Current value of thi1  is  ',thi1:15:6);
   pling:=Question('  Enter new value ');
```

```
 if pling =true then begin
      write('  Enter new value  ');readln(thi1);
 end;
 writeln('  Current value of thi2  is  ',thi2:15:6);
 pling:=Question('  Enter new value ');
 if pling =true then begin
      write('  Enter new value  ');readln(thi2);
 end;
 writeln('  Current value of thi3  is  ',thi3:15:6);
 pling:=Question('  Enter new value ');
 if pling =true then begin
      write('  Enter new value  ');readln(thi3);
 end;
 findmaxmin(thi1,thi2,thi3,prop);
 blodge:=Question('  Are you satisfied  ');
until blodge=true;
end;

Generate3d;
pling:=Question(' Do you wish to generate profile  ');
if pling =true then
 findprofile(5.0,46.0,390.0,thi1,thi2,thi3,Cutdata1,cutdata2,npoints1,npoints2);
pling:=Question(' Do you wish to look at profiles ');
if pling=true then begin
repeat
 write('  enter value of x between 60 and 380 (360 to finish)  ');
 readln(x);
 write('  enter value of delZ ');
 readln(delz);
 write('  enter cutter radius');readln(cutterrad);
 write('  enter cutter diameter');;readln(cutterdiam);

 FindSection(x,thi1,thi2,thi3,dsec,xmx);


 totcount:=30;
 Drawsection(false,true,true,totcount,dsec);
 cutsplit(x,-100.0,100.0,thi1,thi2,thi3,CutData1,Npoints1);
 cutterposition(true,cutterrad,cutterdiam,cutdata1,Npoints1,tnp);
 readcutdata(tnp,tool1);
 drawtool(true,true,false,cutterrad,cutterdiam,true,tnp,tool1);
 DrawSection(true,false,false,Npoints1,Cutdata1);
 newcutline(x,delZ,thi1,thi2,thi3,Npoints1,true,cutdata1);
 DrawSection(true,false,false,Npoints1,Cutdata1);

 cutterposition(true,cutterrad,cutterdiam,cutdata1,Npoints1,tnp);
 readcutdata(tnp,tool1);

 drawtool(true,false,false,cutterrad,cutterdiam,true,tnp,tool1);
 DrawSection(true,false,true,Npoints1,Cutdata1);

 cutsplit(x,-100.0,100.0,thi1,thi2,thi3,CutData1,Npoints1);
 cutterposition(false,cutterrad,cutterdiam,cutdata1,Npoints1,tnp);
 readcutdata(tnp,tool1);
 drawtool(true,true,false,cutterrad,cutterdiam,false,tnp,tool1);
 DrawSection(true,false,false,Npoints1,Cutdata1);
 newcutline(x,delZ,thi1,thi2,thi3,Npoints2,false,cutdata2);
 cutterposition(false,cutterrad,cutterdiam,cutdata2,Npoints2,tnp);
```

```
  readcutdata(tnp,tool1);

  drawtool(true,false,false,cutterrad,cutterdiam,false,tnp,tool1);
  DrawSection(true,false,true,Npoints2,Cutdata2);




 until x=360.0;
 end;
 pling:=Question(' Do you wish to examine profiles out along blade');
 if pling=true then begin
  write('  Enter start cross-section ');readln(starting);
  write('  Enter x increment        ');readln(xinc);
  write('  Enter No sections        ');readln(Nosec);
  write('  Enter delta increment    ');readln(delZ);
  for i:=1 to NoSec do begin

   x:=starting+(i-1)*xinc;
   totcount:=30;
   newcutline(x,delZ,thi1,thi2,thi3,totcount,true,dnew);
   if i=1 then begin first:=true;
         writeln('  totcount ',totcount:8);
         readln;
   end else first:=false;
   last:=false;
   DrawSection(true,first,last,totcount,dnew);
   first:=false;
   newcutline(x,delZ,thi1,thi2,thi3,totcount,false,dnew);
   if i=nosec then last:=true else last:=false;
   DrawSection(true,first,last,totcount,dnew);
  end;
 end;
 pling:=Question('  Examine Y-Z profiles ');
if pling=true then begin
  write('  Enter start cross-section ');readln(starting);
  write('  Enter x increment        ');readln(xinc);
  write('  Enter No sections        ');readln(Nosec);
    y:=0.0;
    FindYSection(Y,starting,xinc,thi1,thi2,thi3,NoSec,true,dsec);
    first:=true;
    DrawYSection(true,first,false,NoSec,dsec);
    FindYSection(Y,starting,xinc,thi1,thi2,thi3,NoSec,false,dsec);
    last:=true;
    DrawYSection(true,false,last,NoSec,dsec);
end;
 pling:=Question('  Do you wish to calculate stresses ');
 if pling=true then begin
volume:=0.0;
My:=0.0;
Mz:=0.0;
delx:=1.25;
Ftot:=0.0;
TotalLift:=0.0;
TotalBM:=0.0;
write('  Input density (Kg/m^3)  ');readln(density);
write('  Input rpm  ');readln(rpm);
```

```
rpm:=rpm*2*pi/60;
for i:=2 to 283 do begin
  X:=45.01+(284-i)*delX;
  write(i:4);

  calculatearea(X,thi1,thi2,thi3,area,yy,zz,Kyy,Kzz);
  liftforce(X,lift,torque);
  TotalLift:=TotalLift+(Lift*delX);
  TotalBm:=TotalBm+(Lift*X*delx);
  volume:=volume+delX*area;
  Ftot:=Ftot+(delX*area*X);
  stress:=Ftot/area;
  stress:=stress*1e-6*density*sqr(rpm);
  My:=My+x*(delX*area)*yy;
  Mz:=Mz+x*(delX*area)*zz;
  writeln(' X ',x:5:1,' Direct (N/m^2) : ',stress:10:0,
          '  Lift (N): ',(delx*lift):8:1,' Torque ',(delX*torque):8:4);


end;
arm:=Ftot/volume;
Liftarm:=TotalBm/TotalLift;
writeln('volume mm^4 :',volume:15:4,' My  :',My:15:4,' Mz  : ',Mz:15:4);
writeln(' centripetal :',Ftot:15:4,'  arm  : ',arm:10:1,
                ' Lift :',totallift:8,' BM ',TotalBm:10:2,' arm ',Liftarm:6:1);
readln;
end;
pling:=Question('  Generate 3-d data file ');
if pling=true then begin
  write('   Enter start cross-section ');readln(starting);
  write('   Enter x increment         ');readln(xinc);
  write('   Enter No sections         ');readln(Nosec);
  Gen3D2(starting,xinc,thi1,thi2,thi3,Nosec);
end;
pling:=Question('  generate split data file');
if pling=true then begin
  write('   Enter start cross-section ');readln(starting);
  write('   Enter x increment         ');readln(xinc);
  write('   Enter No sections         ');readln(Nosec);
  GenSplit3d(starting,xinc,thi1,thi2,thi3,Nosec);
end;
pling:=Question('  Produce Plot File ');
if pling=true then begin
 plotprofile(thi1,thi2,thi3);
end;
end.
```

UNIT PROPMEN2.PAS

```
unit propmen2;
interface
uses propmenu;
procedure newcutline(x,delZ,thi1,thi2,thi3:real;var Npoints:integer;
            upper:boolean;var cutdata:secty);
procedure findmaxmin(thi1,thi2,thi3:Real;prop:propdef);


procedure FindYSection(Y,Xmin,delX,thi1,thi2,thi3:real;
                NoSec:integer; upper:boolean;
                var dsec:secty);
procedure findMaxDepth(upper:boolean;
     Ncts,Nsp:integer;cutty,splity:secty;var MaxDepth:real);

procedure addoffset(upper:boolean;zoff:real;Npts:integer;dsec:secty;
                var dnew:secty);
procedure findreducedcut(upper:boolean;Nsp,Ncts:integer;
                dsp,dcut:secty;var Nnew:integer;var Dnew:secty);

procedure reverseinfo(ncts:integer;dcut:secty;var dnew:secty;
               var oldfinish,finish,start:coord1);
procedure cutterposition(upper:boolean;radius,diam :real;
                dsec:secty;Npoints:integer;var tnp:integer);

procedure cutsplit(prop:propdef;x,Ymin,Ymax,thi1,thi2,thi3:real;
       var dsc:secty;var nts :integer);
procedure cutrough(y,Xmin,Xmax,thi1,thi2,thi3,cutterdiam:real;upper:boolean;
            var dsc:secty;var npts:integer);
procedure readcutdata(var tnp:integer;var dsec:secty);

procedure removeextrapoints(fac,Npts:integer;dsec:secty;Var Nnew:integer;
                  Var dnew:secty);

procedure FindRedSplitCut(upper:boolean;Nsp,Nruf:integer;
                dsp,druf:secty;var Nnew:integer;var Dnew:secty);
procedure findedgetool(radius,diam,depth:real;upper:boolean;
       Ledge:boolean;CutIt:secty;Ncut:integer;var Nock:integer);


implementation
procedure newcutline(x,delZ,thi1,thi2,thi3:real;var Npoints:integer;
            upper:boolean;var cutdata:secty);
 var
  iniy,ztry,val,TotalLength,xmax,dp,dx,tip,dy,dz,L:real;
  y01,z01,radip,zti,yti,dl,ytry,radit,radcurv,zip,ActZ,y0,z0:real;
  lents: Array[1..30] of real;
  dsec,dcuts,drot:secty;
  fig:text;
  j,typ,kip,npots,Npts,cob,index,count,position,Nop,i,noop,locate : integer;
  srt1,srt2,TE,LE,P1,P2,P3,srt :coord1;
  ans,ans1,ok,same : boolean;
  thibest,SumL,MaxZ,MinZ,thet,StartZ,FinZ,dipZ,wipZ : real;
 {+++++++++++++++++}
 procedure lengthvector(t2,t1:coord1;var leng:real);
  begin
       leng:=sqr(t2.x-t1.x)+sqr(t2.y-t1.y)+sqr(t2.z-t1.z);
```

```
      leng:=sqrt(leng);
end;
{+++++++++++++++}
begin
 wipZ:=0.1;
 Findsection(prop,x,thi1,thi2,thi3,dsec,xmax);
 Npts:=30;
 RemoveDudPoints(dsec,Npts);
 FindTE(x,dsec,Npts,TE);
 FindLE(x,xmax,dsec,Npts,LE,radcurv,y0,z0,locate);
 If upper=true then begin
      dcuts[1]:=TE;
      index:=2;
      npots:=1;
      count:=Npts;
      samepoint(dsec[count],TE,ans);
      IF ans=false then begin
       dcuts[index]:=dsec[count];
       index:=index+1;
       count:=count-1;
       npots:=npots+1;
      end else count:=count-1;
  ans:=false;

      repeat
       if count>(locate+4) then
        ans:=false
       else begin

        beyondpoint(dsec[count],LE,dsec[count-1],ans);

    end;
       IF ans=false then begin
        dcuts[index]:=dsec[count];
        index:=index+1;
        npots:=npots+1;
        count:=count-1;
       end else begin
        samepoint(dsec[count],LE,ans1);
        IF ans1=false then begin
             dcuts[index]:=LE;

             Npots:=Npots+1;

        end else begin

             dcuts[index]:=LE;

             Npots:=Npots+1;

        end;
       end;


 until (ans=true) or (count<1);
 if count<2 then begin
      writeln('   program crash count out of range');
      readln;
```

```
        end;
    end else begin
      dcuts[1]:=TE;
        index:=2;
        npots:=1;
        count:=1;
        samepoint(dsec[count],TE,ans);
        IF ans=false then begin
          dcuts[index]:=dsec[count];
          index:=index+1;
          count:=count+1;
          npots:=npots+1;
        end else count:=count+1;
        ans:=false;
        repeat
          if count<locate then
            ans:=false
          else begin
            beyondpoint(dsec[count],LE,dsec[count+1],ans);

          end;
          IF ans=false then begin
            dcuts[index]:=dsec[count];
            index:=index+1;
            npots:=npots+1;
            count:=count+1;
          end else begin
            samepoint(dsec[count],LE,ans1);
            IF ans1=false then begin
                dcuts[index]:=le;
        Npots:=Npots+1;

            end else begin

                dcuts[index]:=LE;

                Npots:=Npots+1;

            end;
          end;

      until (ans=true) or (count>Npts);
      if count>Npts-1 then begin
          writeln('   program crash count out of range');
          readln;
        end;
    end;

{   Drawsection(false,true,true,Npots,dcuts);   }
  {    generate    actual cut coordinates }

  TotalLength:=0.0;
  lents[1]:=0.0;
  For i:=1 to Npots-1 do begin
    lengthvector(dcuts[i],dcuts[i+1],dl);
    TotalLength:=TotalLength+dl;
    lents[i+1]:=TotalLength;
  end;
```

```
Nop:=round(TotalLength/delZ);
if nop>300 then nop:=295;

 dipZ:=TotalLength/(Nop-1);
 CutData[1]:=dcuts[1];
 CutData[Nop]:=dcuts[Npots];


For i:=2 to Nop-1 do begin
 L:=(i-1)*dipZ;
 SumL:=0.0;
 j:=0;
 repeat
      j:=j+1;
 until (j>(Npots-2)) or (lents[j]>L);
 dp:=(L-lents[j-1])/(lents[j]-lents[j-1]);
 dx:=dcuts[j].x-dcuts[j-1].x;
 dy:=dcuts[j].y-dcuts[j-1].y;
 dz:=dcuts[j].z-dcuts[j-1].z;

 if (abs(dx)>=abs(dy)) and (abs(dx)>=abs(dz)) then begin
      typ:=1;
      val:=dcuts[j-1].x+dp*(dcuts[j].x-dcuts[j-1].x);
 end else begin
      if abs(dy)>abs(dz) then begin
            typ:=2;
      val:=dcuts[j-1].y+dp*(dcuts[j].y-dcuts[j-1].y);
      end else begin
   typ:=3;
      val:=dcuts[j-1].z+dp*(dcuts[j].z-dcuts[j-1].z);
      end;
 end;
 if j>=Npots-1 then begin
      P1:=dcuts[npots-2];
      P2:=dcuts[npots-1];
      P3:=dcuts[npots];
 end else begin
      P1:=dcuts[j-1];
      P2:=dcuts[j];
      P3:=dcuts[j+1];
 end;
 QuadSpline(ok,same,P1,P2,P3,val,typ,srt);

 CutData[i]:=srt;

end;
tip:=0.0;
iniy:=CutData[Nop].y;
For i:=2 to Nop do begin
 If CutData[i].y<=(CutData[i-1].y) then begin
  CutData[i].y:=CutData[i-1].y+0.1;
 end;
end;
tip:=CuTdata[Nop].y-iniy;
If tip >0.0 then begin
For i:=2 to Nop do begin
 CutData[i].y:=CutData[i].y - ((i-1)*tip)/(Nop-1);
end;
```

```
  end;
  Npoints:=Nop;
  assign(fig,'tempcut.dat');
  rewrite(fig);
  for i:=1 to Nop do writeln(fig,cutdata[i].y:10:3,cutdata[i].z:10:3);
  close(fig);
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++}
procedure findmaxmin(thi1,thi2,thi3:Real;prop:propdef);
 var
  i,j,jj:integer;
  srt,max,min:coord1;
  upp:boolean;
 begin
  findpoint(1,1,thi1,thi2,thi3,true,prop,srt);
  max:=srt;
  min:=srt;
  For i:=1 to 10 do begin
       For j:=1 to 30 do begin
        if j<16 then begin
              upp:=false;
              jj:=j;
        end else begin
          upp:=true;
          jj:=31-j;
       end;
       findpoint(jj,i,thi1,thi2,thi3,upp,prop,srt);
       if srt.x>Max.x then Max.x:=srt.x else
       if srt.x<Min.x then Min.x:=srt.x;
       if srt.y>Max.y then Max.y:=srt.y else
       if srt.y<Min.y then Min.y:=srt.y;
       if srt.z>Max.z then Max.z:=srt.z else
       if srt.z<Min.z then Min.z:=srt.z;
    end;
  end;
  writeln('  maximum  ',max.x:15:6,max.y:15:6,max.z:15:6);
  writeln('  minimum  ',min.x:15:6,min.y:15:6,min.z:15:6);
  write('  press return to continue');readln;

end;

{++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure FindYSection(Y,Xmin,delX,thi1,thi2,thi3:real;
                NoSec:integer; upper:boolean;
                var dsec:secty);
 var
  ijk,ip,position,npts: integer;
  X,xmax,Ymin,Ymax:real;
  dsc:secty;
  srt,P1,P2,P3:coord1;
  ok,same :boolean;
 begin
  For ijk:=1 to NoSec do begin
       X:=Xmin+delX*(ijk-1);
    Findsection(prop,x,thi1,thi2,thi3,dsc,xmax);
    Npts:=30;
    RemoveDudPoints(dsc,Npts);
```

```
        Ymin:=dsc[1].y;
        Ymax:=Ymin;
        For ip:=2 to Npts do begin
      if dsc[ip].y>Ymax then Ymax:=dsc[ip].y
        else if dsc[ip].y<Ymin then Ymin:=dsc[ip].y;
        end;
        if (Y>=Ymin) and (Y<=Ymax) then begin
         if upper=true then begin
          position:=0;
          repeat
              position:=position+1;
          until dsc[position+1].y>=y;
          P1:=dsc[position];
          P2:=dsc[position+1];
          P3:=dsc[position+2];
       QuadSpline(ok,same,P1,P2,P3,y,2,srt);
          end else begin
          position:=npts+1;
          repeat
              position:=position-1;
          until dsc[position+1].y>=y;
          P1:=dsc[position];
          P2:=dsc[position+1];
          P3:=dsc[position+2];
       QuadSpline(ok,same,P1,P2,P3,y,2,srt);
          end;
          dsec[ijk]:=srt;
         end else begin
          dsec[ijk].x:=x;
          dsec[ijk].y:=y;
          dsec[ijk].z:=0.0;
        end;
   end;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure findMaxDepth(upper:boolean;
      Ncts,Nsp:integer;cutty,splity:secty;var MaxDepth:real);
 var i,j :integer;
 zsplit,depth:real;
 begin
  MaxDepth:=0.0;
  For i:= 1 to Ncts do begin
   j:=0;
        repeat
         j:=j+1;
        until (splity[j+1].y>=cutty[i].y) or (j>=Nsp-1);
    zsplit:=splity[j].z+
    ((splity[j+1].z-splity[j].z)*
    ((cutty[i].y-splity[j].y)/(splity[j+1].y-splity[j].y)));
        if upper=true then begin
         depth:=cutty[i].z-zsplit;
        end else begin
         depth:=zsplit-cutty[i].z;
        end;
        if depth>maxdepth then maxdepth:=depth;


  end;
```

```
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure addoffset(upper:boolean;zoff:real;Npts:integer;dsec:secty;
                    var dnew:secty);
 var i:integer;
 sgn:real;
 begin
  if upper=true then sgn:=-1.0 else sgn:=1.0;
  For i:=1 to Npts do begin
        dnew[i].z:=dsec[i].z+sgn*zoff;
        dnew[i].y:=dsec[i].y;
        dnew[i].x:=dsec[i].x;
  end;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure findreducedcut(upper:boolean;Nsp,Ncts:integer;
                    dsp,dcut:secty;var Nnew:integer;var Dnew:secty);
 var
  i,j :Integer;
  zsplit:real;
  init:boolean;
  plip:coord1;
 begin
  Nnew:=0;
  init:=false;
  For i:=1 to ncts do begin
   j:=0;
       repeat
         j:=j+1;
       until dsp[j+1].y>=dcut[i].y;

   zsplit:=dsp[j].z+
   ((dsp[j+1].z-dsp[j].z)*
   ((dcut[i].y-dsp[j].y)/(dsp[j+1].y-dsp[j].y)));
   if upper=true then begin
         if dcut[i].z>=zsplit then begin
      if init=false then begin
       init:=true;
             if i>1 then begin
             plip.x:=dcut[i-1].x;
       plip.y:=dcut[i-1].y;
          plip.z:=dcut[i-1].z;
             NNew:=Nnew+1;
             dNew[Nnew]:=plip;
             end;
         end;
         Nnew:=Nnew+1;
         Dnew[Nnew]:=dcut[i];
        end else begin
         if init=true then begin
             init:=false;
```

```
                plip.x:=dcut[i].x;
                plip.y:=dcut[i].y;
                plip.z:=dcut[i].z;

                Nnew:=Nnew+1;
                dNew[Nnew]:=plip;
            end;
          end;
        end else begin
          if dcut[i].z<=zsplit then begin
       if init=false then begin

        init:=true;
                if i>1 then begin
                plip.x:=dcut[i-1].x;
                plip.y:=dcut[i-1].y;
                plip.z:=dcut[i-1].z;
                NNew:=Nnew+1;
                dNew[Nnew]:=plip;
        end;
          end;
          Nnew:=Nnew+1;
          Dnew[Nnew]:=dcut[i];
         end else begin
          if init=true then begin
                init:=false;
                plip.x:=dcut[i].x;
                plip.y:=dcut[i].y;
                plip.z:=dcut[i].z;
                Nnew:=Nnew+1;
                dNew[Nnew]:=plip;
          end;
         end;
        end;
 end;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++
+++}
procedure reverseinfo(ncts:integer;dcut:secty;var dnew:secty;
                var oldfinish,finish,start:coord1);
 var dist1,dist2:real;
 i:integer;
 begin
  dist1:=sqrt(sqr(finish.y-dcut[1].y)+sqr(finish.x-dcut[1].x));
  dist2:=sqrt(sqr(finish.y-dcut[ncts].y)+sqr(finish.x-dcut[ncts].x));
  if dist1<=dist2 then begin
       For i:=1 to ncts do dnew[i]:=dcut[i];
  end else begin
       For i:=1 to Ncts do dnew[i]:=dcut[ncts+1-i];
  end;
  oldfinish:=finish;
  finish:=dnew[Ncts];
  start:=dNew[1];

end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++}
```

```
procedure removeextrapoints(fac,Npts:integer;dsec:secty;Var Nnew:integer;
                    Var dnew:secty);
 var
  inc,j,k:integer;
 begin
  dnew[1]:=dsec[1];
  inc:=round(Npts/fac);
  if inc=0 then inc:=1;

  j:=1;
  k:=1;
  repeat
       k:=k+1;
       j:=j+inc;
       dnew[k]:=dsec[j];

  until (j+inc)>=Npts;
  k:=k+1;
  dnew[k]:=dsec[Npts];
  Nnew:=k;

end;

{*********************************************************}
procedure cutterposition(upper:boolean;radius,diam :real;
                  dsec:secty;Npoints:integer;var tnp:integer);
 const
  concave=true;
  convex=false;
 var
  fin:text;
  ijk:integer;
  U1,U2,prev1,mid,prev2,pos1,pos2,at:coord1;
  samepoint,vertical,result:boolean;
 procedure compare(upper:boolean;prev1,prev2,pos1,pos2:coord1;
                                                  var result:boolean);
 var t2,t1:coord1;
 th1,th2,tm1,tm2:real;
 begin
  t2.x:=pos2.x-pos1.x;
  t2.y:=pos2.y-pos1.y;
  t2.z:=pos2.z-pos1.z;
  t1.x:=prev2.x-prev1.x;
  t1.y:=prev2.y-prev1.y;
  t1.z:=prev2.z-prev1.z;
  tm1:=sqrt(sqr(t1.x)+sqr(t1.y)+sqr(t1.z));
  tm2:=sqrt(sqr(t2.x)+sqr(t2.y)+sqr(t2.z));
   if tm1<>0.0 then th1:=t1.z/tm1 else th1:=0.0;
  th2:=t2.z/tm2;
  if upper=true then
       if th1<th2 then result:=convex else result:=concave
  else
       if th2<th1 then result:=convex else result:=concave;
 end;
 {=============}
 procedure writepos(pos:coord1;var count:integer);
  begin
       count:=count+1;
```

```
        writeln(fin,pos.x:15:6,pos.y:15:6,pos.z:15:6);
end;
{+++++++++++++++}
procedure verticality(prev1,prev2,U1,U2:coord1;var pos1,pos2:coord1);
 var
        t,that:coord1;
        tm:real;
 begin
        t.x:=prev2.x-prev1.x;
        t.y:=prev2.y-prev1.y;
        t.z:=prev2.z-prev1.z;
   tm:=sqrt(sqr(t.x)+sqr(t.y)+sqr(t.z));
        that.x:=t.x/tm;
        that.y:=t.y/tm;
        that.z:=t.z/tm;
        pos1.x:=prev1.x+t.x-0.5*diam*that.x;
        pos1.y:=prev1.y+t.y-0.5*diam*that.y;
        pos1.z:=prev1.z;
        pos2.x:=pos1.x+t.x-0.5*diam*that.x;
        pos2.y:=pos1.y+t.y-0.5*diam*that.y;
        pos2.z:=pos1.z+(U2.z-U1.z);
end;
{+++++++++++++++}
 procedure findinter(U1,U2,U3,U4:coord1;var at:coord1);
        var
          t1,t2,t3:coord1;
          a,b,num,denom:real;
        begin
    t1.z:=U2.z-U1.z;
         t1.y:=U2.y-U1.y;
    t2.z:=U4.z-U3.z;
         t2.y:=U4.y-U3.y;
         t3.z:=U1.z-U3.z;
         t3.y:=U1.y-U3.y;
         num:=t3.y*t2.z-t3.z*t2.y;
         denom:=t1.z*t2.y-t1.y*t2.z;
         if denom=0.0 then at:=U3
         else begin
           a:=num/denom;
      at.y:=u1.y+a*t1.y;
           at.z:=u1.z+a*t1.z;
           at.x:=0.5*(u2.x+u3.x);
    end;
 end;

procedure intersection(prev1,prev2,pos1,pos2:coord1;var at:coord1);
 procedure findab(p1,p2:coord1;var a,b:real;var typ:integer);
        var
          tm:real;
        begin
         if abs(p2.x-p1.x)>abs(p2.y-p1.y) then begin
           tm:=p2.x-p1.x;
           typ:=1;
         end else begin
       tm:=p2.y-p1.y;
           typ:=2;
         end;
```

```
    b:=p1.z;
        if tm=0.0 then typ:=3 else
    a:=(p2.z-p1.z)/tm;
  end;
  var
      val,test,a1,b1,a2,b2:real;
      typ1,typ2 :integer;
  {=========}
  begin
   test:=sqrt(sqr(pos1.y-prev2.y)+sqr(pos1.z-prev2.z));
   if test>0 then begin
      findab(prev1,prev2,a1,b1,typ1);
      findab(pos1,pos2,a2,b2,typ2);
      if (a1<>a2) and (typ1<>3) and (typ2<>3) then begin
       if typ2=2 then begin
        at.x:=pos1.x;
    val:=((b2-b1)-a2*pos1.y+a1*prev1.y)/(a1-a2);
        at.y:=val;
        at.z:=a2*(val-pos1.y)+b2;
       end else begin
        at.y:=pos1.y;
    val:=((b2-b1)-a2*pos1.x+a1*prev1.x)/(a1-a2);
        at.x:=val;
        at.z:=a2*(val-pos1.x)+b2;
       end;
      end else begin
        at:=pos1;
      end;
   end else begin
        at:=pos1;
   end;



  end;
 {++++++++++++++++}

begin
 assign(fin,'tempfile.dat');
 rewrite(fin);
 tnp:=0;
 For ijk:=1 to (Npoints-1) do begin
  U1:=dsec[ijk];
  U2:=dsec[ijk+1];
  newfindtooltip(radius,diam,upper,U1,U2,pos1,pos2,samepoint,vertical);
  if samepoint=false then begin
      if vertical=true then begin
        verticality(prev1,prev2,U1,U2,pos1,pos2);
        writepos(pos1,tnp);
        prev1:=pos1;
        prev2:=pos2;
        halt;
      end else begin
       if ijk=1 then begin
        writepos(pos1,tnp);
        prev1:=pos1;
        prev2:=pos2;
        if Npoints=2 then writepos(pos2,tnp);
```

```
          end else begin
            compare(upper,prev1,prev2,pos1,pos2,result);
            findinter(prev1,prev2,pos1,pos2,at);
{         writeln(prev1.y:10:3,prev1.z:10:3);
            writeln(prev2.y:10:3,prev2.z:10:3);
            writeln(pos1.y:10:3,pos1.z:10:3);
            writeln(pos2.y:10:3,pos2.z:10:3);
            writeln(ijk:4,at.y:10:3,at.z:10:3);
            writeln;    }
            if result=concave then begin
                  writepos(at,tnp);
                  prev1:=pos1;
                  prev2:=pos2;
            end else begin
                  writepos(at,tnp);
                  prev1:=pos1;
                  prev2:=pos2;
            end;


        if ijk>=(Npoints-1) then writepos(pos2,tnp);
          end;
        end;
   end else begin
         {ignore if same point}
   end;

 end;
  close(fin);

end;
{=========================================================
=========}
procedure readcutdata(var tnp:integer;var dsec:secty);
 var fin:text;
 ijk:integer;
 begin
  if tnp>300 then tnp:=300;
  assign(fin,'tempfile.dat');
  reset(fin);
  for ijk:=1 to tnp do begin
        readln(fin,dsec[ijk].x,dsec[ijk].y,dsec[ijk].z);
  end;
  close(fin);

end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++}
procedure cutsplit(prop:propdef;x,Ymin,Ymax,thi1,thi2,thi3:real;
        var dsc:secty;var nts :integer);
 var
  dsec:secty;
  TE,LE,start,fint:coord1;
  radcurv,y0,z0,xmax:real;
  locate,npts:integer;
 begin
  Findsection(prop,x,thi1,thi2,thi3,dsec,xmax);
  NPts:=30;
```

```
  RemoveDudPoints(dsec,Npts);
  FindTe(x,dsec,npts,TE);
  FindLe(x,xmax,dsec,Npts,LE,radcurv,y0,z0,locate);

  start.x:=TE.x;
  start.y:=Ymin;
  start.z:=TE.z;
  fint.x:=LE.x;
  fint.y:=Ymax;
  fint.z:=LE.z;
  nts:=4;
  dsc[1]:=start;
  dsc[2]:=TE;
  dsc[3]:=LE;
  dsc[4]:=fint;

end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++}
procedure cutrough(y,Xmin,Xmax,thi1,thi2,thi3,cutterdiam:real;upper:boolean;
            var dsc:secty;var npts:integer);
 var
 zmax,zmin,x,yleft,yright,xblade:real;
 tip1,tip2,TE,Le:coord1;
 i,total,locate:integer;
 dsec:secty;
 z0,y0,radcurv,ztry1,xmx,ztry2:real;
 begin
  yleft:=y-0.5*cutterdiam;
  yright:=y+0.5*cutterdiam;
  findpoint(1,7,thi1,thi2,thi3,true,prop,tip1);
  findpoint(1,22,thi1,thi2,thi3,false,prop,tip2);
  if tip2.x>tip1.x then xblade:=tip2.x else xblade:=tip1.x;
  zmax:=0.5*(tip1.z+tip2.z);
  zmin:=zmax;
  total:=round((XBlade-Xmin)/5.0);
  for i:=1 to (total-1) do begin
   x:=Xblade-i*5.0;
   Findsection(prop,x,thi1,thi2,thi3,dsec,xmx);
   NPts:=30;
   RemoveDudPoints(dsec,Npts);
   FindTe(x,dsec,npts,TE);
   FindLe(x,xmx,dsec,Npts,LE,radcurv,y0,z0,locate);
   if yleft<TE.y then ztry1:=TE.z else
        if yleft>LE.y then ztry1:=LE.z else begin
          ztry1:=TE.z + (LE.z-TE.z)*((yleft-TE.y)/(LE.y-TE.y));
       end;
   if yright<TE.y then ztry2:=TE.z else
        if yright>LE.y then ztry2:=LE.z else begin
          ztry2:=TE.z + (LE.z-TE.z)*((yright-TE.y)/(LE.y-TE.y));
       end;
       if ztry1>Zmax then zmax:=ztry1 else if ztry1<Zmin then Zmin:=ztry1;
       if ztry2>Zmax then zmax:=ztry2 else if ztry2<Zmin then Zmin:=ztry2;
  end;
  npts:=2;
  if upper=true then begin
       dsc[1].x:=Xmin;
       dsc[1].y:=y;
```

```
        dsc[1].z:=Zmin-1;
        dsc[2].x:=Xmax;
        dsc[2].y:=y;
        dsc[2].z:=Zmin-1
    end else begin
        dsc[1].x:=Xmin;
        dsc[1].y:=y;
        dsc[1].z:=Zmax+1;
        dsc[2].x:=Xmax;
        dsc[2].y:=y;
        dsc[2].z:=Zmax+1;
    end;

end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++}
procedure FindRedSplitCut(upper:boolean;Nsp,Nruf:integer;
                dsp,druf:secty;var Nnew:integer;var Dnew:secty);
  var
    i,j,imem,jblip:integer;
    Zis,zx,delZ:real;
    init:boolean;
    below:Array[1..4] of boolean;
  begin
    For i:=1 to 4 do begin

        j:=0;
        repeat
          j:=j+1;
        until druf[j].y>=dsp[i].y;
        If j=1 then zis:=druf[j].z else
        begin
          Zis:=druf[j-1].z+
                (druf[j].z-druf[j-1].z)*(dsp[i].y-druf[j-1].y)/(druf[j].y-druf[j-1].y);
        end;
        if i=1 then Zis:=druf[1].z;
        if i=4 then Zis:=druf[Nruf].z;
     if (upper=true) then
          if (dsp[i].z>=Zis) then below[i]:=true else below[i]:=false
        else
       if (dsp[i].z<=Zis) then below[i]:=true else below[i]:=false

    end;
    NNew:=0;
    For i:=1 to 4 do begin
        if below[i]=true then begin
          NNew:=NNew+1;
          Dnew[Nnew]:=dsp[i];
          if i<4 then begin
            if below[i+1]=false then begin
        jblip:=0;
            repeat
              jblip:=jblip+1;
            until druf[jblip].y>=dsp[i].y;
        j:=jblip;
                repeat
                  j:=j+1;
                  zis:=dsp[i].z+
```

```
                (dsp[i+1].z-dsp[i].z)*(druf[j].y-dsp[i].y)/(dsp[i+1].y-dsp[i].y);
               if upper =true then
                delz:=zis-druf[j].z
               else
                delz:=druf[j].z-zis;
              until (delZ<0.0) or (j=Nruf);
              NNew:=Nnew+1;
              dNew[NNew].x:=dsp[i].x;
              DNew[NNew].y:=druf[j].y;
              DNew[NNew].z:=druf[j].z;

          end;
         end;
        end else begin
         if i<4 then begin
          if below[i+1]=true then begin
        jblip:=0;
            repeat
             jblip:=jblip+1;
            until druf[jblip].y>=dsp[i].y;
        j:=jblip;
              repeat
               j:=j+1;
               zis:=dsp[i].z+
                (dsp[i+1].z-dsp[i].z)*(druf[j].y-dsp[i].y)/(dsp[i+1].y-dsp[i].y);
               if upper =true then
                delz:=zis-druf[j].z
               else
                delz:=druf[j].z-zis;
              until (delZ>0.0) or (j=Nruf);
              NNew:=Nnew+1;
              dNew[NNew].x:=dsp[i].x;
              DNew[NNew].y:=druf[j].y;
              DNew[NNew].z:=druf[j].z;

          end;
         end;
        end;
   end;

end;
{++++++++++++++++++++++++++++++++++++++++}
procedure findedgetool(radius,diam,depth:real;upper:boolean;
       Ledge:boolean;CutIt:secty;Ncut:integer;var Nock:integer);
   var
       fin:text;
       pos1,pos2,prev1,prev2,at:coord1;
    i,ii:integer;
   procedure writepos(pos:coord1;var count:integer);
   begin
       count:=count+1;

       writeln(fin,pos.x:15:6,pos.y:15:6,pos.z:15:6);
   end;
   {++++++++++++++++}
   procedure findpoint(upper:boolean;radius,diam,depth:real;
                              U1,U2:coord1;var P1,P2:coord1);
    var
```

```
        t,that:coord1;
        tlen,offs:real;
      begin
        t.x:=U2.x-U1.x;
        t.y:=U2.y-U1.y;
        tlen:=sqrt(sqr(t.x)+sqr(t.y));
        that.x:=t.x/tlen;
        that.y:=t.y/tlen;
        if radius<0.0 then begin
          offs:=diam*0.5;
        end else begin
          offs:=sqrt((2*radius*depth)-sqr(depth));
        end;
        P1.x:=U1.x - that.y*offs;
        P1.y:=U1.y + that.x*offs;
   if upper=true then
     p1.z:=U1.z+depth
        else
          p1.z:=U1.z-depth;
        P2.x:=U2.x - that.y*offs;
        P2.y:=U2.y + that.x*offs;
   if upper=true then
     p2.z:=U2.z+depth
        else
          p2.z:=U2.z-depth;
end;
{+++++++++++++++++}
procedure findinter(U1,U2,U3,U4:coord1;var at:coord1);
      var
        t1,t2,t3:coord1;
        a,b,num,denom:real;
      begin
  t1.x:=U2.x-U1.x;
       t1.y:=U2.y-U1.y;
  t2.x:=U4.x-U3.x;
       t2.y:=U4.y-U3.y;
       t3.x:=U1.x-U3.x;
       t3.y:=U1.y-U3.y;
       num:=t3.x*t2.y-t3.y*t2.x;
       denom:=t1.y*t2.x-t1.x*t2.y;
       if denom=0.0 then at:=U3
       else begin
         a:=num/denom;
    at.x:=u1.x+a*t1.x;
         at.y:=u1.y+a*t1.y;
         at.z:=0.5*(u2.z+u3.z);
  end;
end;
{+++++++++++++++++}

begin
       assign(fin,'tempfile.dat');
   rewrite(fin);
   Nock:=0;
       For ii:=1 to Ncut do begin
         if ledge=false then i:=ii else i:=Ncut+1-ii;
         If ii=1 then begin
              findpoint(upper,radius,diam,depth,CutIt[i],CutIt[i+1],prev1,prev2);
```

```
                writepos(prev1,nock);
            end else begin
                if ii=Ncut then begin
                  writepos(prev2,nock);
                end else begin
                  findpoint(upper,radius,diam,depth,CutIt[i],CutIt[i+1],pos1,pos2);
                  findinter(prev1,prev2,pos1,pos2,at);
                  writepos(at,nock);
                  prev1:=pos1;
                  prev2:=pos2;
                end;
            end;
        end;

            close(fin);
    end;
end.
```

UNIT PROPMEN3.PAS

```
unit propmen3;
{       NC cutter routine  }
interface
uses crt,graph,propmenu,propmen2,inout;
const
        move='G00';
        cut ='G01';
        Tx = 1; Ty = 2 ; Tz = 4 ;
        Txy = 3; Txz = 5; Tyz = 6;
        Txyz = 7;
        Top = True;
        Lower = False;
type
        NCcoord = Array[1..3] of longint;
    tooltype = record
                        diam,radius :real;
                    end;
        BlockSize = record
                        Max,Min:coord1;
                       end;
procedure writecut(var NCFile:text;instruc : string;
            XYZ: NCcoord; typ : integer;
            Var Cutflag,MoveFlag:boolean);
procedure ChooseToolType(var tool:tooltype);
procedure translate(upper:boolean;offset,coordies:coord1;var NCMach : NCcoord);
procedure BlockSetUP(Var Blk:BlockSize;Var Offset:coord1;
              Var upper,CutFlag,moveflag:boolean;
                        Var MaxCut,delZ:real);
procedure OpenCutFile(CutTyp:integer;Var NCFile:text);
procedure ChooseCutType(var cutype:integer);
procedure MakeFinishCut(videon:boolean;Blk:Blocksize; thi1,thi2,thi3,MaxCut:real;
              var NCfile:text;CutFlag,MoveFlag:boolean;
              upper:boolean;offset,finito:coord1;tool:tooltype;
               Xstart,Xfinish,Xinc,delZ :real);

procedure defineCut(blk:blocksize;typ:integer;
             tool:tooltype;var Xstart,Xfinish,Xinc :real);
procedure MakeSplitCut(videon:boolean;Blk:Blocksize; thi1,thi2,thi3,MaxCut:real;
              var NCfile:text;CutFlag,MoveFlag:boolean;
              upper:boolean;offset,finito:coord1;tool:tooltype;
               Xstart,Xfinish,Xinc,delZ :real);
procedure drawboundary(remsect,first,last:boolean;blk:blocksize;offset:coord1);

procedure MakeRoughCut(videon:boolean;Blk:Blocksize; thi1,thi2,thi3,MaxCut:real;
              var NCfile:text;CutFlag,MoveFlag:boolean;
              upper:boolean;offset,finito:coord1;tool:tooltype);

procedure MakeEdgeCut(var NCFile:text;offset:coord1;var finito:coord1;
              CutFlag,MoveFlag:boolean;
              upper,ledge:boolean;tool:tooltype;
              Depth,Xmin,Xmax,thi1,thi2,thi3:real);

implementation
{**********************************************************}
 procedure drawboundary(remsect,first,last:boolean;blk:blocksize;offset:coord1);
  var
```

```
        i: integer;
        scalex,scaley:real;
        xasp,yasp :word;
        dataI : array[1..5,1..2] of integer;
   begin
    scaleX:=2.56;
        Getaspectratio(xasp,yasp);
        scaleY:=(ScaleX*yasp)/(xasp);
        if (remsect=false) or (first=true) then begin

    SetGraphMode(2);
    SetBkColor(0);

    SetColor(11);
        end;
    dataI[1,1]:=round(getmaxX /2)+round(scaleX*blk.min.y);
    dataI[1,2]:=round(getMaxY /2)-round(scaleY*blk.min.z);
    dataI[2,1]:=round(getmaxX /2)+round(scaleX*blk.min.y);
    dataI[2,2]:=round(getMaxY /2)-round(scaleY*blk.max.z);
    dataI[3,1]:=round(getmaxX /2)+round(scaleX*blk.max.y);
    dataI[3,2]:=round(getMaxY /2)-round(scaleY*blk.max.z);
    dataI[4,1]:=round(getmaxX /2)+round(scaleX*blk.max.y);
    dataI[4,2]:=round(getMaxY /2)-round(scaleY*blk.min.z);
    dataI[5,1]:=round(getmaxX /2)+round(scaleX*blk.min.y);
    dataI[5,2]:=round(getMaxY /2)-round(scaleY*blk.min.z);

  moveto(DataI[1,1],DataI[1,2]);
  for i:=2 to 5 do lineto(dataI[i,1],dataI[I,2]);

  if (remsect=false) or (last=true) then begin
     outtextxy(50,50,' Please press space bar');
     repeat until readkey=' ';
  repeat until readkey=' ';
  restorecrtmode;
  end;
  end;



{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++}
procedure ChooseToolType(var tool:tooltype);
  procedure writetool(num:integer;tool:tooltype);
    begin
        write('  (',num:2,')  ');
        write('   Tool shaft diameter  : ');
        if tool.radius>0 then begin
          write(tool.diam:6:2,'   end radius  : ',tool.radius:6:2);
          writeln('   Ball Nose ');
        end else begin
          write(tool.diam:6:2,'                    ');
          writeln('   End Mill');
        end;
  end;
  {+++++++++}
  var inf:text;
        chose,numty,i :integer;
        tooling : Array[1..10] of tooltype;
```

```
 begin
  assign(inf,'c:\tpascal\nc\tooltype.dat');
  reset(inf);
  readln(inf,numty);
  For i:=1 to numty do readln(inf,tooling[i].diam,tooling[i].radius);
  writeln;
  writeln('     TOOLS  AVAILABLE     ');
  writeln;
  For i:=1 to Numty do begin
       writetool(i,tooling[i]);
       writeln;
  end;
  chose:=InputInt('  Enter number of tool required ',1,numty);
  tool:=tooling[chose];
  close(inf);
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++}
procedure convert(a : coord1;var b: NCcoord);
Var i:integer;
begin
 a.x:=a.x*1000.0;
 a.y:=a.y*1000.0;
 a.z:=a.z*1000.0;
 b[1]:=round(a.x);
 b[2]:=round(a.y);
 b[3]:=round(a.z);
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++}
procedure writecut(var NCFile:text;instruc : string;
            XYZ: NCcoord; typ : integer;
            Var Cutflag,MoveFlag:boolean);
var
 outstr : array[1..3] of string;
 i : integer;
begin
 if (MoveFlag = False) and (instruc = move) then begin
   MoveFlag := true;
   CutFlag := false;
   write(NCfile,move);
 end;
 if (CutFlag = false) and (instruc = cut) then begin
   CutFlag := true;
   MoveFlag := false;
   write(NCfile,cut);
 end;

 for i := 1 to 3 do  str(XYZ[i],outstr[i]);

 outstr[1]:=concat('X',outstr[1]);
 outstr[2]:=concat('Y',outstr[2]);
 outstr[3]:=concat('Z',outstr[3]);

Case Typ of
 Tx: Writeln(NCfile,outstr[1]);
 Ty: Writeln(NCfile,outstr[2]);
 Tz: Writeln(NCfile,outstr[3]);
```

```
  Txy: Writeln(NCfile,outstr[1],outstr[2]);
  Txz: Writeln(NCfile,outstr[1],outstr[3]);
  Tyz: Writeln(NCfile,outstr[2],outstr[3]);
  Txyz: Writeln(NCfile,outstr[1],outstr[2],outstr[3]);
end;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++}
procedure translate(upper:boolean;offset,coordies:coord1;var NCMach : NCcoord);
  var Ncoordies:coord1;
begin
  Ncoordies.x:=Coordies.x-Offset.x;
  If upper=true then begin
    Ncoordies.y:=Coordies.y-Offset.y;
    Ncoordies.z:=-Coordies.z+Offset.z;
  end else begin
    Ncoordies.y:=-coordies.y+Offset.y;
    Ncoordies.z:=coordies.z-Offset.z;
  end;
  Convert(Ncoordies,NCmach);


end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++}
procedure BlockSetUP(Var Blk:BlockSize;Var Offset:coord1;
                Var upper,CutFlag,moveflag:boolean;
                        Var MaxCut,delZ:real);
const
  XBlockMin=90.0;
  XBlockMax=425.0;
  YBlockMin=-120.0;
  YBlockMax=120.0;
  ZBlockMinLow=-50.0;
  ZBlockMaxLow=25.0;
  ZBlockMinUp=-25.0;
  ZBlockMaxUp=50.0;
  OrigX=115.0;
  OrigY=75.0;
  OrigZ=75.0;
var
  yip:NcCoord;
begin
  writeln;
  upper:=Question('    Is this the Upper surface to be cut ');
  writeln;
  blk.max.x:=XBlockMax;
  blk.min.x:=XBlockMin;
  blk.max.y:=yBlockMax;
  blk.min.y:=yBlockMin;
  if upper=true then begin
    blk.max.z:=zBlockMaxUp;
    blk.min.z:=zBlockMinUp;
    offset.z:=zblockmaxUp;
  end else begin
    blk.max.z:=zBlockMaxlow;
    blk.min.z:=zBlockMinlow;
    offset.z:=Zblockminlow;
```

```
 end;
 Offset.x:=OrigX;
 Offset.y:=OrigY;
 CutFlag:=False;
 MoveFlag:=false;


 write('  Enter maximum depth of cut : ');readln(MaxCut);
 writeln;
 write('  Enter maximum cut step     : ');readln(delZ);
 writeln;
 writeln('          PROPELLER COORDINATES        ');
 writeln;
 if upper=true then begin
 writeln('Xmax  ',XBlockMax:10:2,' Ymax  ',YBlockMax:10:2,' Zmax ',
          ZBlockMaxUp:10:2);
 writeln('Xmin  ',XBlockMin:10:2,' Ymin  ',YBlockMin:10:2,' Zmin ',
          ZBlockMinUp:10:2);
 end else begin
 writeln('Xmax  ',XBlockMax:10:2,' Ymax  ',YBlockMax:10:2,' Zmax ',
          ZBlockMaxLow:10:2);
 writeln('Xmin  ',XBlockMin:10:2,' Ymin  ',YBlockMin:10:2,' Zmin ',
          ZBlockMinLow:10:2);
 end;
 writeln;
 writeln('          MACHINE COORDINATES');
 writeln;
 translate(upper,offset,Blk.max,yip);
 writeln('Xmax  ',Yip[1]:10,' Ymax  ',Yip[2]:10,' Zmax ',yip[3]:10);
 translate(upper,offset,Blk.min,yip);
 writeln('Xmin  ',Yip[1]:10,' Ymin  ',Yip[2]:10,' Zmin ',yip[3]:10);
 anykeytocont;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++
++}
procedure movestart(bgn:boolean;
             var ncfile:text;var cutflag,moveflag:boolean;
                upper:boolean;offset,finish,start:coord1);
 var
 fig1,fig2:coord1;
 nfig: NCcoord;
 begin
  if upper=true then begin
       if finish.z<start.z then begin
         fig1.x:=finish.x;
         fig1.y:=finish.y;
         fig1.z:=finish.z-5.0;
         fig2.x:=start.x;
         fig2.y:=start.y;
         fig2.z:=finish.z-5.0;
       end else begin
         fig1.x:=finish.x;
         fig1.y:=finish.y;
         fig1.z:=start.z-5.0;
         fig2.x:=start.x;
         fig2.y:=start.y;
         fig2.z:=fig1.z
```

```
        end;
    end else begin
     if finish.z>start.z then begin
          fig1.x:=finish.x;
          fig1.y:=finish.y;
          fig1.z:=finish.z+5.0;
          fig2.x:=start.x;
          fig2.y:=start.y;
          fig2.z:=finish.z+5.0;
        end else begin
          fig1.x:=finish.x;
          fig1.y:=finish.y;
          fig1.z:=start.z+5.0;
          fig2.x:=start.x;
          fig2.y:=start.y;
          fig2.z:=fig1.z
        end;
     translate(upper,offset,fig1,nfig);
        if bgn=true then
     writecut(NCFile,move,nfig,Txyz,Cutflag,MoveFlag)
        else
     writecut(NCFile,move,nfig,Tz,Cutflag,MoveFlag);
        translate(upper,offset,fig2,nfig);
     writecut(NCFile,move,nfig,Txyz,Cutflag,MoveFlag);



    end;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++}
procedure writecutdatatofile(tnp:integer;var CutFlag,MoveFlag:boolean;
                    var NCfile:text;upper:boolean;offset:coord1);
  var fin:text;
  ijk,typ:integer;
  dimp,last:coord1;
  limp:NCcoord;
  begin
    assign(fin,'tempfile.dat');
    reset(fin);
    for ijk:=1 to tnp do begin
        readln(fin,dimp.x,dimp.y,dimp.z);
        translate(upper,offset,dimp,limp);
        IF ijk=1 then typ:=Txyz else
        begin
          If dimp.x=last.x then Typ:=Tyz else
          If dimp.y=last.y then typ:=Txz else
          If dimp.z=last.z then typ:=Txy else typ:=Txyz;
        end;
        writecut(NCFile,cut,limp,typ,CutFlag,MoveFlag);
        last:=dimp;
    end;
    close(fin);
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++}
procedure MakeFinishCut(videon:boolean;Blk:Blocksize; thi1,thi2,thi3,MaxCut:real;
                var NCfile:text;CutFlag,MoveFlag:boolean;
```

```
            upper:boolean;offset,finito:coord1;tool:tooltype;
             Xstart,Xfinish,Xinc,delZ :real);
var
 i,count,NoCuts:integer;
 ActX,Xval,offzet:real;
 oldfinish,finish,start:coord1;
 dtemp,dsplit,dcut,dnew,dprn:secty;
 fac,Nspl,Ncut,NoPass,Nnew,tnp,Nnop:integer;
 MaxDepth,deep:real;
 bgn:boolean;
begin

 finish:=finito;

 NoCuts:=round((Xfinish-Xstart)/Xinc);
 ActX:=(Xfinish-Xstart)/(NoCuts-1);

 For i:=1 to NoCuts do begin
  if i=1 then bgn:=true else bgn:=false;
  Xval:=Xstart + (i-1)*ActX;

  cutsplit(xval,Blk.min.y,Blk.max.y,thi1,thi2,thi3,dsplit,Nspl);
     if videon=true then DrawSection(true,true,false,Nspl,dsplit);
  newcutline(xval,delZ,thi1,thi2,thi3,Ncut,upper,Dcut);
     if videon=true then DrawSection(true,false,false,Ncut,dcut);
  findMaxDepth(upper,Ncut,Nspl,dcut,dsplit,MaxDepth);

  if MaxCut>=MaxDepth then begin
        reverseinfo(Ncut,Dcut,Dtemp,oldfinish,finish,start);
    cutterposition(upper,tool.radius,tool.diam,Dtemp,Ncut,tnp);
    readcutdata(tnp,dprn);
    if videon=true then DrawTool(true,false,true,tool.radius,
                                 tool.diam,upper,tnp,dprn);
       movestart(bgn,ncfile,cutflag,moveflag,upper,offset,oldfinish,start);
    writecutdatatofile(tnp,CutFlag,MoveFlag,NCfile,upper,offset);
     end else begin
      NoPass:=round(MaxDepth/MaxCut);
      If NoPass=1 then NoPass:=2;
      deep:=(MaxDepth/NoPass);

   For count:=1 to NoPass do begin

    Offzet:=MaxDepth-count*deep;
    IF count< NoPass then begin
     addoffset(upper,offzet,Ncut,dcut,dtemp);
     findreducedcut(upper,Nspl,Ncut,dsplit,dtemp,Nnew,Dnew);
     reverseinfo(Nnew,Dnew,Dtemp,oldfinish,finish,start);
          fac:=10*count;
     removeextrapoints(fac,Nnew,Dtemp,Nnop,dnew);
     cutterposition(upper,tool.radius,tool.diam,Dnew,Nnop,tnp);
       end else begin
          reverseinfo(Ncut,Dcut,Dtemp,oldfinish,finish,start);
     cutterposition(upper,tool.radius,tool.diam,Dtemp,Ncut,tnp);
       end;
       readcutdata(tnp,dprn);

    if videon=true then
       DrawTool(true,false,false,tool.radius,tool.diam,upper,tnp,dprn);
```

```
          if (nopass=count) and (videon=true) then
         drawboundary(true,false,true,blk,offset);
          movestart(bgn,ncfile,cutflag,moveflag,upper,offset,oldfinish,start);
       writecutdatatofile(tnp,CutFlag,MoveFlag,NCfile,upper,offset);
         end;

        end;


  end;
end;
{++++++++++++++++++++++++++++++++++++++++++}
procedure ChooseCutType(var cutype:integer);
 begin
 writeln;
 writeln;
 writeln('    POSSIBLE CUTS ARE :');
 writeln;
 writeln('    (1)    Rough Cut');
 writeln('    (2)    Split Line Cut');
 writeln('    (3)    Finish Cut  ');
 writeln('    (4)    Edge Cut');
 writeln;
 cutype:=InputInt('   Enter choice  ',1,4);
end;
{++++++++++++++++++++++++++++++++++++++++++}
procedure OpenCutFile(CutTyp:integer;Var NCFile:text);
 var
 fname,fn:string;
 lentil:integer;
 begin
  writeln;
  writeln;
  write('   ENTER FILE NAME (maxlength=8 characters)  ');readln(fname);
  if length(fname)>8 then lentil:=8 else lentil:=length(fname);
  fn:=Copy(fname,1,8);
  fn:='c:\tpascal\nc\'+fn;
  CASE Cuttyp of
       1:fn:=fn+'.Ruf';
       2:fn:=fn+'.Spl';
       3:fn:=fn+'.Fin';
       4:fn:=fn+'.Edg';
  end;

  assign(NCFile,fn);
  rewrite(NCFile);

end;
{++++++++++++++++++++++++++++++++++++++++++}
{++++++++++++++++++++++++++++++++++++++++++}
procedure defineCut(blk:blocksize;typ:integer;
            tool:tooltype;var Xstart,Xfinish,Xinc :real);
 var
  Xbig:real;
  blih:coord1;
 begin
  Case typ of
       2:begin
```

```
          Xbig:=blk.max.x;
        writeln;
        writeln('       SPLIT CUT LIMITS');
          end;
         3:begin
          Xbig:=(400.0-0.5*tool.diam);
        writeln;
        writeln('       FINAL CUT LIMITS');
       end;
         4:begin
          writeln;
          writeln('     Leading or Trailing Edge Cut');
          writeln;
          XStart:=Blk.min.x;
          Xfinish:= 399-0.5*tool.diam;
          if tool.radius<0.0 then xbig:=5.0 else
          xbig:=tool.radius-sqrt(sqr(tool.radius)-sqr(0.5*tool.diam));
          Xinc:=InputReal(' Depth of Cut  ',0.0,xbig,6,1);
         end;
   end;
   if typ<4 then begin
   writeln;
   Xstart:=InputReal('  Start Cutting at X = ',Blk.min.x,xbig,6,1);
   writeln;
   Xfinish:=InputReal('  Finish Cutting at X= ',Xstart,Xbig,6,1);
   writeln;
   IF tool.radius>0.0 then Xinc:=sqrt(0.2*tool.radius-0.01) else
   xinc:=tool.diam;
   end;
end;

{++++++++++++}
procedure FindZ(TE,LE:coord1;Y:real;Var Zis:real);
 begin
   IF Y<=TE.y then Zis:=TE.z
   else if Y>=LE.y then Zis:=LE.z
   else begin
        IF LE.y=TE.y then zis:=0.5*(Le.z+TE.z)  else
        Zis:=TE.z+((LE.z-TE.z)*(Y-TE.y)/(LE.y-TE.y));
   end;
end;

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++}
procedure FindRoughProfile(upper:boolean;thi1,thi2,thi3:real;
                    Blk:BlockSize;var Npots:integer;var Maxy:secty;
                     var xmax:real);
 var
   dsec:secty;
   LE,TE:coord1;
   Npts,i,j,locate:integer;
   dX,Dy,X,Y,Y0,Z0,radcurv,Zis :real;
 begin
   dX:=(Blk.Max.x-Blk.Min.x)/(100-1);
   dy:=(Blk.Max.y-Blk.Min.y)/(200-1);
   For j:=1 to 200 do begin
   Maxy[j].x:=0.0;
   Maxy[j].y:=Blk.Min.Y+(j-1)*dY;
```

```
  if upper=true then Maxy[j].z:=1000.0 else Maxy[j].z:=-1000.0;
  end;


  For i:=1 to 100 do begin
      X:=Blk.Min.x + (i-1)*dX;
   Findsection(x,thi1,thi2,thi3,dsec,xmax);
   NPts:=30;
   RemoveDudPoints(dsec,Npts);
   FindTe(x,dsec,npts,TE);
   FindLe(x,xmax,dsec,Npts,LE,radcurv,y0,z0,locate);
   For j:=1 to 200 do begin
       Y:=Maxy[j].y;
       FindZ(TE,LE,Y,Zis);
       if upper=true then begin
        IF zis<Maxy[j].z then begin
         Maxy[j].x:=LE.x;
         Maxy[j].z:=Zis;
        end;
       end else begin
        IF zis>Maxy[j].z then begin
         Maxy[j].x:=LE.x;
         Maxy[j].z:=Zis;
        end;
       end;
      end;
  end;
  Npots:=200;
end;
{++++++++++++++++++++++++++++++++++++++++++++}
procedure MakeSplitCut(videon:boolean;Blk:Blocksize; thi1,thi2,thi3,MaxCut:real;
              var NCfile:text;CutFlag,MoveFlag:boolean;
             upper:boolean;offset,finito:coord1;tool:tooltype;
              Xstart,Xfinish,Xinc,delZ :real);
 var
  i,count,NoCuts:integer;
  ActX,Xval,offzet,xmax:real;
  oldfinish,finish,start:coord1;
  druf,dsplit,dnew,dprn,dtemp:secty;
  fac,Nspl,Nruf,NoPass,Nnew,tnp,Nnop:integer;
  MaxDepth,deep:real;
  bgn:boolean;
 begin

  finish:=finito;

  NoCuts:=round((Xfinish-Xstart)/Xinc);
  ActX:=(Xfinish-Xstart)/(NoCuts-1);

  FindRoughProfile(upper,thi1,thi2,thi3,Blk,Nruf,druf,xmax);

 For i:=1 to NoCuts do begin
   if i=1 then bgn:=true else bgn:=false;
   Xval:=Xstart + (i-1)*ActX;

      if videon=true then DrawSection(true,true,false,Nruf,druf);

   cutsplit(xval,Blk.min.y,Blk.max.y,thi1,thi2,thi3,dsplit,Nspl);
```

```
        if videon=true then DrawSection(true,false,false,Nspl,dsplit);

    findMaxDepth(upper,Nspl,Nruf,dsplit,druf,MaxDepth);
    if videon=false then writeln(i:4,' maxDepth ',maxdepth:8:2,' maxcut ',maxcut:6:2);
    if MaxCut>=MaxDepth then begin
          reverseinfo(Nspl,Dsplit,Dtemp,oldfinish,finish,start);
      cutterposition(upper,tool.radius,tool.diam,Dtemp,Nspl,tnp);
      readcutdata(tnp,dprn);
      if videon=true then DrawTool(true,false,true,tool.radius,
                                   tool.diam,upper,tnp,dprn);
          movestart(bgn,ncfile,cutflag,moveflag,upper,offset,oldfinish,start);
     writecutdatatofile(tnp,CutFlag,MoveFlag,NCfile,upper,offset);
      end else begin
       NoPass:=round(MaxDepth/MaxCut);

       If NoPass=1 then NoPass:=2;
       deep:=(MaxDepth/NoPass);
       if videon=false then writeln(Nopass:8,' Nopasses ',deep:6:2,' deep');
    For count:=1 to NoPass do begin

      Offzet:=MaxDepth-count*deep;
      IF count< NoPass then begin
       addoffset(upper,offzet,Nspl,dsplit,dtemp);
       FindRedSplitCut(upper,Nspl,Nruf,dtemp,druf,Nnew,Dnew);
       reverseinfo(Nnew,Dnew,Dtemp,oldfinish,finish,start);
       cutterposition(upper,tool.radius,tool.diam,Dnew,Nnew,tnp);
         end else begin
              reverseinfo(Nspl,Dsplit,Dtemp,oldfinish,finish,start);
       cutterposition(upper,tool.radius,tool.diam,Dtemp,Nspl,tnp);
         end;
         readcutdata(tnp,dprn);
         if videon=true then
         DrawTool(true,false,false,tool.radius,tool.diam,upper,tnp,dprn);
         if (nopass=count) and (videon=true) then
        drawboundary(true,false,true,blk,offset);
         if (count=1) then
          movestart(bgn,ncfile,cutflag,moveflag,upper,offset,oldfinish,start);
     writecutdatatofile(tnp,CutFlag,MoveFlag,NCfile,upper,offset);
       end;

      end;


  end;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++}
procedure FindZHeight(upper:boolean;Blk:blocksize;
              Yval,diam:real;Nruf:integer;druf:secty;
              Var Zcut,MaxDepth:real);
 var
 Z1,Z2,Z3,ytest:real;
 i,j,k:integer;
 begin
  ytest:=Yval-0.5*diam;
  j:=0;
  Repeat
      j:=j+1;
```

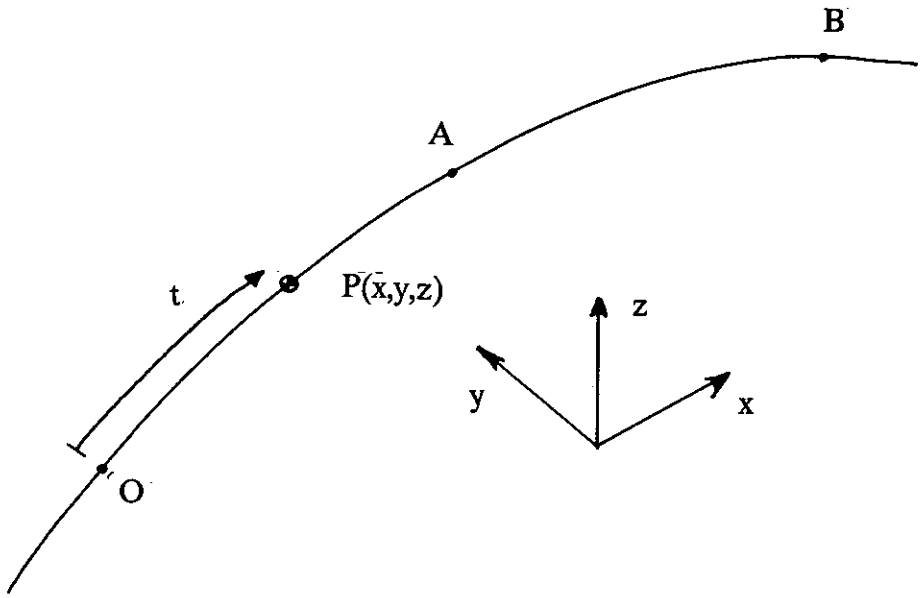
```
  until (Druf[j].y>=ytest) or (j>=Nruf);
  if j=1 then z1:=druf[1].z else
  if j>=Nruf then z1:=druf[j].z else
  z1:=druf[j-1].z+(Ytest-druf[j-1].y)*(druf[j].z-druf[j-1].z)/(druf[j].y-druf[j-1].y);
  Zcut:=z1;

  ytest:=Yval;
  j:=0;
  Repeat
      j:=j+1;
  until (Druf[j].y>=ytest) or (j>=Nruf);
  if j=1 then z1:=druf[1].z else
  if j>=Nruf then z1:=druf[j].z else
  z1:=druf[j-1].z+(Ytest-druf[j-1].y)*(druf[j].z-druf[j-1].z)/(druf[j].y-druf[j-1].y);
  if upper=true then begin
      if z1<zcut then zcut:=z1;
  end else begin
      if z1>zcut then zcut:=z1;
  end;
      ytest:=Yval+0.5*diam;
  j:=0;
  Repeat
      j:=j+1;
  until (Druf[j].y>=ytest) or (j>=Nruf);
  if j=1 then z1:=druf[1].z else
  if j>=Nruf then z1:=druf[j].z else
  z1:=druf[j-1].z+(Ytest-druf[j-1].y)*(druf[j].z-druf[j-1].z)/(druf[j].y-druf[j-1].y);
  if upper=true then begin
      if z1<zcut then zcut:=z1;
  end else begin
      if z1>zcut then zcut:=z1;
  end;

  if upper=true then maxdepth:=zcut-Blk.min.z
  else maxdepth:=Blk.max.z-zcut;

end;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++}
procedure MakeRoughCut(videon:boolean;Blk:Blocksize; thi1,thi2,thi3,MaxCut:real;
              var NCfile:text;CutFlag,MoveFlag:boolean;
             upper:boolean;offset,finito:coord1;tool:tooltype);
 var
  i,count,j,NoCuts:integer;
  ActX,yval,offzet,xmax,Yinc,zcut:real;
  oldfinish,finish,cord1,cord2,start:coord1;
  druf,dtemp:secty;
  fac,Nspl,Nruf,NoPass,Nnew,tnp,Nnop:integer;
  MaxDepth,dipz,zip,deep:real;
  bgn,root:boolean;
  ncord:NCcoord;
 begin
  Yinc:=0.5*tool.diam;
  finish:=finito;
  root:=true;
  NoCuts:=round((Blk.max.y-Blk.min.y)/Yinc);
  ActX:=(Blk.max.y-Blk.min.y)/(NoCuts-1);
```

```
 FindRoughProfile(upper,thi1,thi2,thi3,Blk,Nruf,druf,xmax);
     if videon=true then DrawSection(true,true,false,Nruf,druf);
 For i:=1 to NoCuts do begin
   Yval:=Blk.min.y + (i-1)*ActX;

   FindZHeight(upper,Blk,Yval,tool.diam,Nruf,druf,Zcut,MaxDepth);

   If maxdepth<=MaxCut then NoPass:=1 else begin
        NoPass:=trunc(abs(MaxDepth/MaxCut));
        NoPass:=NoPass+1;
      end;
   dipZ:=MaxDepth/(NoPass);

   For j:=1 to NoPass do begin
       if upper=true then begin
         Zip:=Blk.Min.z + dipZ*j;
       end else begin
         Zip:=Blk.Max.z - dipZ*j;
       end;
       if j=Nopass then zip:=zcut;
       if root=true then begin
         cord1.x:=Blk.min.x-0.5*tool.diam;
         cord2.x:=Blk.max.x+0.5*tool.diam;
         root:=false;
       end else begin
         cord1.x:=Blk.max.x+0.5*tool.diam;
         cord2.x:=Blk.min.x-0.5*tool.diam;
     root:=true;
       end;
       cord1.y:=Yval;
       cord2.y:=Yval;
       cord1.z:=zip;
       cord2.z:=zip;
       translate(upper,offset,cord1,ncord);
       if (i=1) and (j=1) then writecut(NCFile,move,ncord,Txyz,CutFlag,MoveFlag)
       else begin
         writecut(NCFile,move,ncord,Tz,Cutflag,MoveFlag);
         writecut(NCFile,cut,ncord,Ty,CutFlag,MoveFlag);
       end;
    translate(upper,offset,cord2,ncord);
    writecut(NCFile,cut,ncord,Tx,Cutflag,MoveFlag);
    dtemp[1]:=cord1;
       dtemp[2]:=cord2;
       if videon=true then begin
         if (j=NoPass) and (i=NoCuts) then
          DrawTool(true,false,true,tool.radius,tool.diam,upper,2,dtemP)
         else DrawTool(true,false,false,tool.radius,tool.diam,upper,2,dtemp);
       end;
      end;
 end;
 finito:=cord2;
end;
{++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++}
{++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++}
procedure MakeEdgeCut(var NCFile:text;offset:coord1;var finito:coord1;
            CutFlag,MoveFlag:boolean;
```

```
                upper,ledge:boolean;tool:tooltype;
                Depth,Xmin,Xmax,thi1,thi2,thi3:real);
  var
    CutLe,CutTe,Dib:secty;
    NTe,Nle,Nock,Nib:integer;
  begin
    findprofile(tool.radius,Xmin,Xmax,thi1,thi2,thi3,
                cutle,cutTe,NTe,Nle);
    drawplan(true,true,false,NTe,CutTe);
    drawplan(true,false,false,Nle,CutLe);
    if ledge=true then
      findedgetool(tool.radius,tool.diam,depth,upper,Ledge,Cutle,Nle,Nock)
    else
        findedgetool(tool.radius,tool.diam,depth,upper,Ledge,Cutte,Nte,Nock);
    readcutdata(Nock,Dib);
    drawplan(true,false,true,Nock,Dib);
    movestart(true,ncfile,cutflag,moveflag,upper,offset,finito,Dib[1]);
    writecutdatatofile(Nock,CutFlag,MoveFlag,NCfile,upper,offset);
    finito:=Dib[Nock];
  end;
  end.
```

Appendix 2

Quadratic Parametric Spline (QuadSpline; Unit PropMenu)



An approximation to a three-dimensional space curve can be defined in terms of a parameter t measured along its length. The order of the approximation will depend on the number of points defining the curve and/or the slope conditions at the two ends (e.g. gradient,curvature). The determination of the coordinates of a point lying on the approximation to the space curve are defined in terms of the parameter t. If t is unknown but at least one of the coordinates of the point P is known a method of solving the approximation equations for the space curve is required to uniquely determine t. The first order (linear) and second order(quadratic) approximations can be solved directly, whereas an iterative method is required for the third order(cubic) approximation. Insufficient data is available to define a propeller surface using linear interpolation but the concave and convex curves present are sufficiently regular that a quadratic approximation was considered adequate. To define the second-order approximation, the position of three points (O, A an B) were required. The unknown P(x,y,z) is defined in terms of the parameter t as follows.

$$x = a_1 . t^2 + b_1 . t + c_1 \qquad 1\text{-}1$$
$$y = a_2 . t^2 + b_2 . t + c_2 \qquad 1\text{-}2$$
$$z = a_3 . t^2 + b_3 . t + c_3 \qquad 1\text{-}3$$

The values of the a, b and c coefficients are determined from the three specified points O, A and B by setting t=0 for Point O, t = 1 for Point B, and t = |OA|/|OB| for point A, where |OA| and |OB|are the straight line lengths between the respective points. These values of t for the three known points allow the three sets of coefficients a, b and c to be directly determined.

If the value of x of P is known,(or y, or z) then 1-1 or (1-2, or 1-3) is solved to determine t.

$$t = \frac{-b_1 \pm \sqrt{b_1^2 - 4 . a_1 . c_1}}{2 . a_1}$$

This value of t is then used to determine the remaining coordinates of Point P. If the a coefficient is zero a straightforward linear interpolation is used. As t has to lie between 0 and 1 the correct solution for t can then be chosen.

Table 1 Unmodified Wageningen B4 -40 Profile Definition

X	P	c/D	h_d/D	t/D	h_t/D
1.00	1.000	0.000	-0.044	0.0025	0.500
0.95	1.000	0.119	0.026	0.0049	0.500
0.90	1.000	0.158	0.055	0.0072	0.500
0.80	1.000	0.197	0.091	0.0120	0.479
0.70	1.000	0.214	0.112	0.0168	0.443
0.60	1.000	0.218	0.122	0.0215	0.389
0.50	1.000	0.215	0.126	0.0262	0.355
0.40	1.000	0.205	0.123	0.0310	0.350
0.30	1.000	0.188	0.115	0.0358	0.350
0.20	1.000	0.166	0.103	0.0405	0.350

		-100	-80	-60	-40	-20	0	20	40	60	70	80	90	95	98	100
0.95	BACK	0.000	0.448	0.720	0.888	0.972	1.000	0.972	0.888	0.720	0.595	0.448	0.295	0.216	0.117	0.000
	FACE	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.90	BACK	0.000	0.451	0.700	0.870	0.970	1.000	0.970	0.870	0.700	0.585	0.451	0.301	0.220	0.116	0.000
	FACE	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.80	BACK	0.000	0.409	0.678	0.853	0.967	1.000	0.970	0.863	0.700	0.605	0.470	0.316	0.224	0.135	0.000
	FACE	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.70	BACK	0.000	0.394	0.669	0.849	0.966	1.000	0.976	0.891	0.732	0.625	0.490	0.329	0.230	0.139	0.000
	FACE	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.60	BACK	0.051	0.402	0.671	0.854	0.968	1.000	0.981	0.906	0.754	0.635	0.516	0.348	0.253	0.175	0.102
	FACE	0.051	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.005	0.019	0.050	0.102
0.50	BACK	0.097	0.419	0.671	0.854	0.968	1.000	0.981	0.906	0.757	0.655	0.533	0.388	0.306	0.256	0.170
	FACE	0.097	0.017	0.000	0.000	0.000	0.000	0.000	0.000	0.003	0.008	0.017	0.044	0.072	0.105	0.170
0.40	BACK	0.178	0.477	0.702	0.865	0.970	1.000	0.975	0.904	0.769	0.715	0.616	0.487	0.407	0.350	0.243
	FACE	0.178	0.062	0.015	0.000	0.000	0.000	0.000	0.003	0.017	0.035	0.059	0.099	0.134	0.175	0.243
0.30	BACK	0.253	0.509	0.716	0.868	0.968	1.000	0.981	0.913	0.804	0.740	0.648	0.537	0.465	0.445	0.310
	FACE	0.253	0.122	0.058	0.017	0.000	0.000	0.004	0.013	0.046	0.071	0.109	0.162	0.196	0.235	0.310
0.20	BACK	0.277	0.521	0.721	0.869	0.966	1.000	0.981	0.919	0.814	0.738	0.661	0.508	0.554	0.486	0.339
	FACE	0.277	0.152	0.083	0.036	0.008	0.000	0.004	0.020	0.060	0.096	0.132	0.161	0.189	0.229	0.339

Table 2 Final Modified Wageningen B4-40 Profile Definition

X	P	c/D	h_d/D	t/D	h_t/D
1.00	0.927	0.000	-0.043	0.0025	0.500
0.95	0.957	0.136	0.025	0.0049	0.500
0.90	0.986	0.181	0.054	0.0072	0.500
0.80	1.039	0.225	0.089	0.0120	0.478
0.70	1.084	0.245	0.109	0.0167	0.442
0.60	1.117	0.250	0.119	0.0215	0.389
0.50	1.133	0.246	0.122	0.0262	0.355
0.40	1.124	0.225	0.120	0.0310	0.349
0.30	1.074	0.169	0.083	0.0409	0.350
0.25	1.024	0.111	0.062	0.0512	0.350

		-100	-80	-60	-40	-20	0	20	40	60	70	80	90	95	98	100
0.95	BACK	-0.418	0.030	0.302	0.470	0.554	0.582	0.554	0.470	0.302	0.177	0.030	-0.123	-0.202	-0.301	-0.418
	FACE	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418
0.90	BACK	-0.414	0.038	0.286	0.456	0.556	0.586	0.556	0.456	0.286	0.171	0.038	-0.113	-0.194	-0.298	-0.414
	FACE	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414
0.80	BACK	-0.412	-0.002	0.266	0.441	0.555	0.588	0.558	0.451	0.288	0.193	0.058	-0.095	-0.187	-0.277	-0.412
	FACE	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412	-0.412
0.70	BACK	-0.414	-0.020	0.255	0.435	0.553	0.586	0.562	0.478	0.318	0.211	0.076	-0.084	-0.184	-0.275	-0.414
	FACE	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414	-0.414
0.60	BACK	-0.365	-0.014	0.255	0.438	0.552	0.584	0.565	0.490	0.338	0.219	0.100	-0.068	-0.163	-0.241	-0.314
	FACE	-0.365	-0.416	-0.416	-0.416	-0.416	-0.416	-0.416	-0.416	-0.416	-0.416	-0.416	-0.411	-0.397	-0.366	-0.314
0.50	BACK	-0.321	0.002	0.254	0.436	0.550	0.582	0.563	0.489	0.340	0.237	0.116	-0.030	-0.112	-0.162	-0.247
	FACE	-0.321	-0.400	-0.418	-0.418	-0.418	-0.418	-0.418	-0.418	-0.414	-0.410	-0.401	-0.374	-0.345	-0.313	-0.247
0.40	BACK	-0.251	0.048	0.273	0.436	0.541	0.571	0.546	0.475	0.360	0.286	0.187	0.058	-0.022	-0.079	-0.186
	FACE	-0.251	-0.367	-0.414	-0.429	-0.429	-0.429	-0.429	-0.426	-0.412	-0.394	-0.370	-0.330	-0.295	-0.253	-0.186
0.30	BACK	-0.189	0.067	0.273	0.425	0.525	0.557	0.539	0.471	0.362	0.297	0.206	0.094	0.023	0.002	-0.133
	FACE	-0.189	-0.321	-0.385	-0.426	-0.443	-0.443	-0.438	-0.430	-0.396	-0.372	-0.334	-0.280	-0.245	-0.208	-0.133
0.25	BACK	-0.173	0.072	0.272	0.419	0.517	0.550	0.532	0.469	0.364	0.288	0.212	0.158	0.105	0.036	-0.111
	FACE	-0.173	-0.298	-0.366	-0.414	-0.442	-0.450	-0.445	-0.429	-0.389	-0.353	-0.318	-0.289	-0.260	-0.221	-0.111

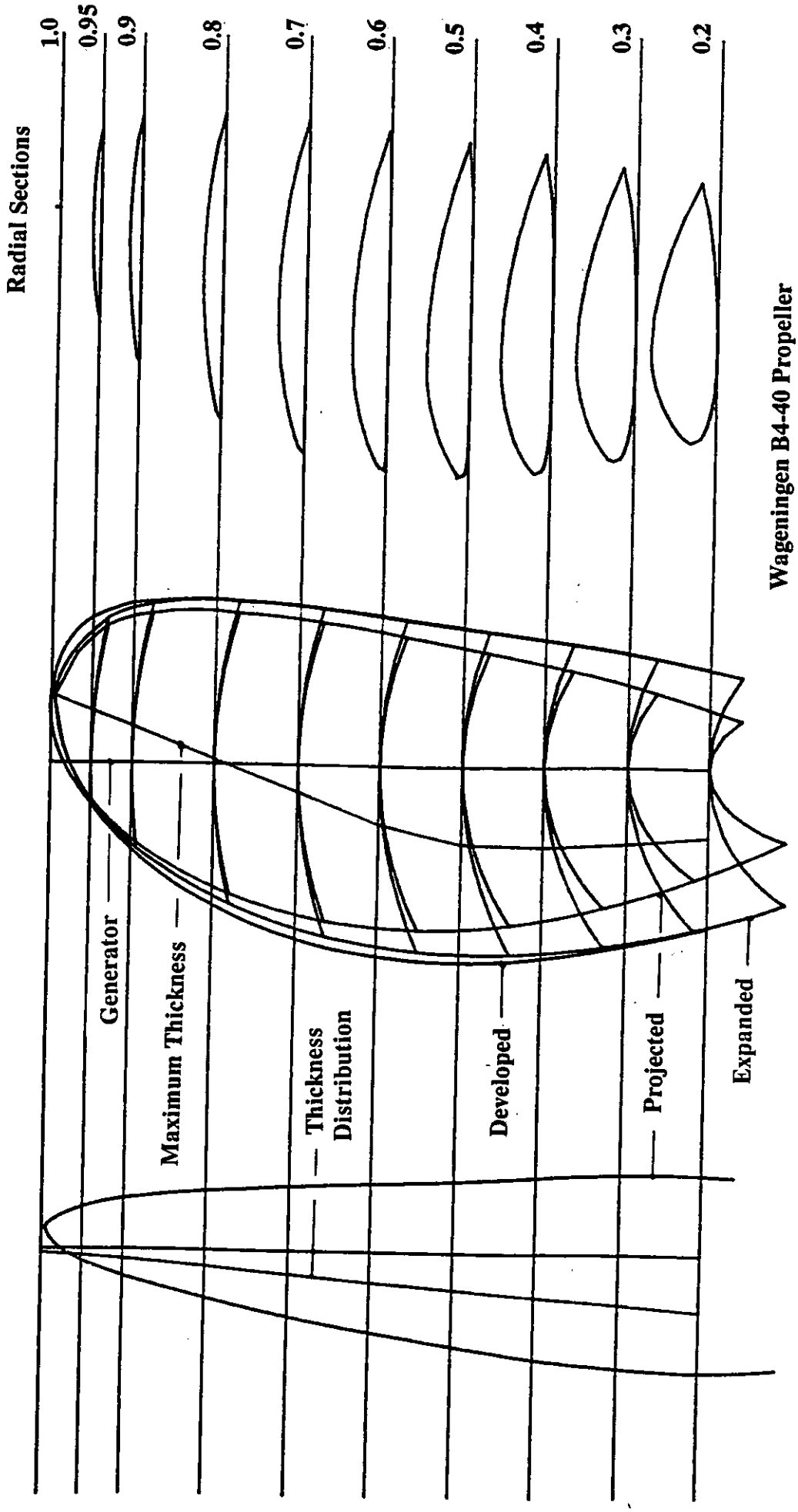


Figure 1 Definition Of Ship Propeller Geometry

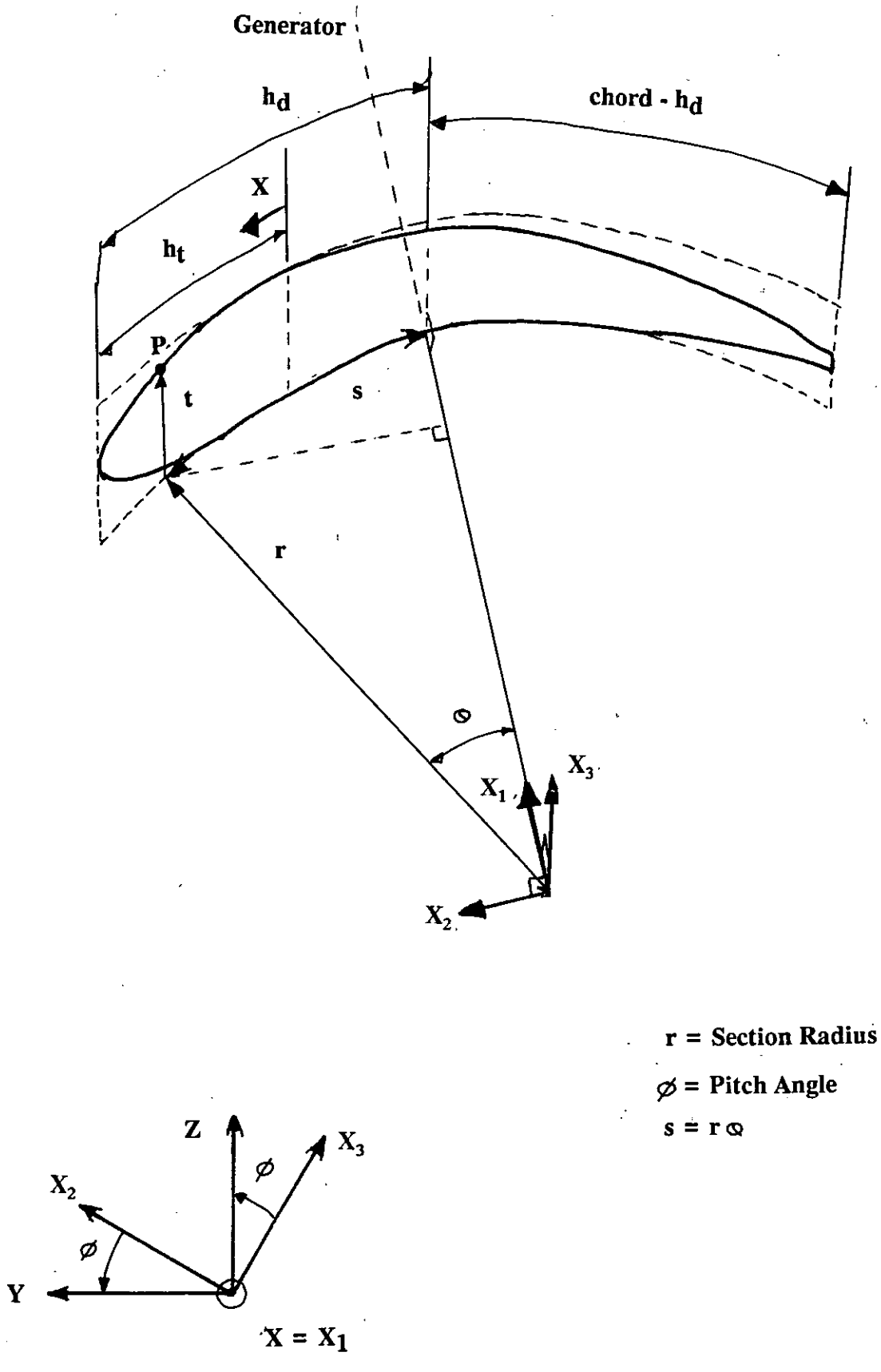


Figure 2 Transformation from Propeller Definition to Cartesian Coordinates.

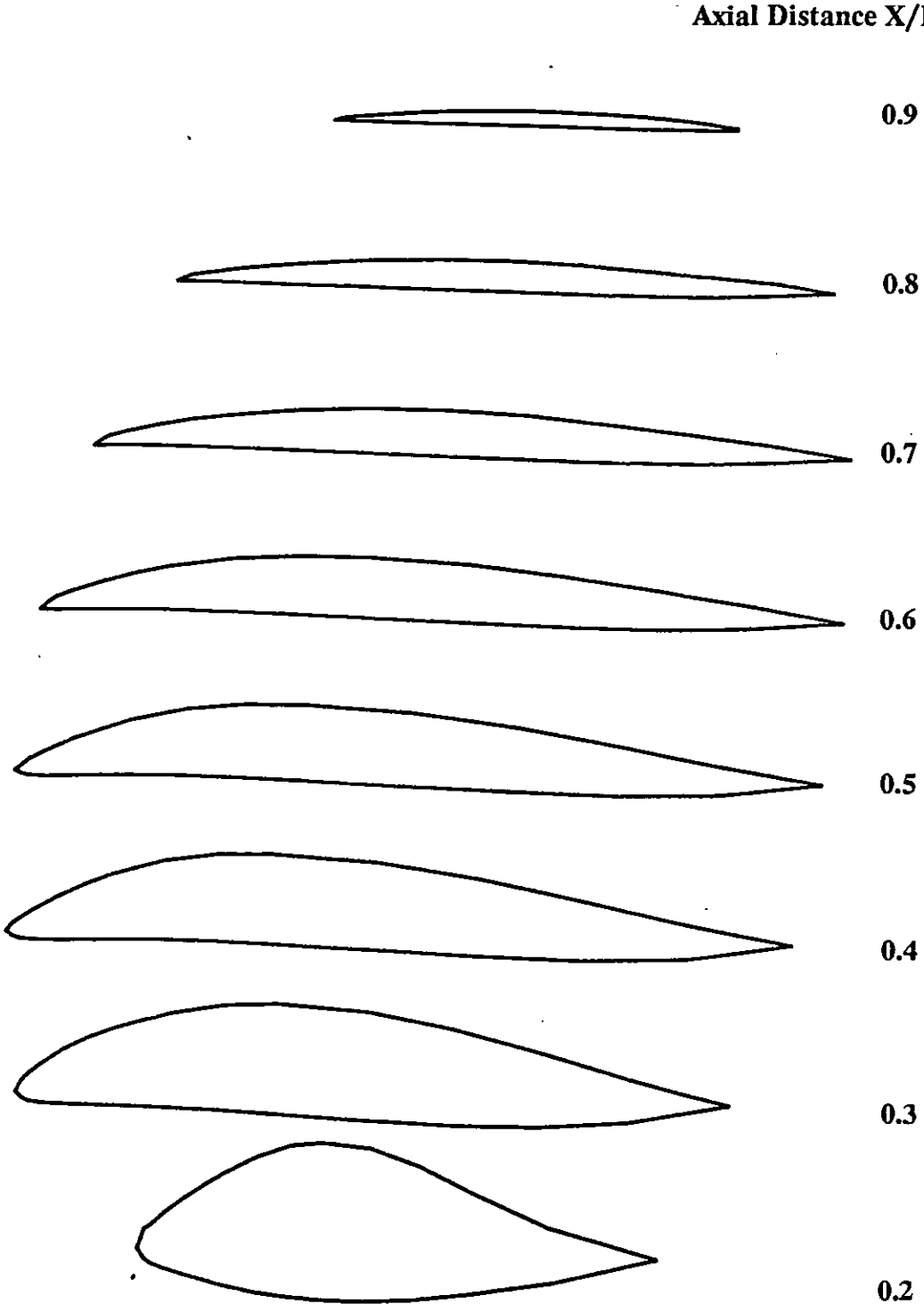


Figure 3 Two-Dimensional y-z Sections Of the Wageningen B4-40

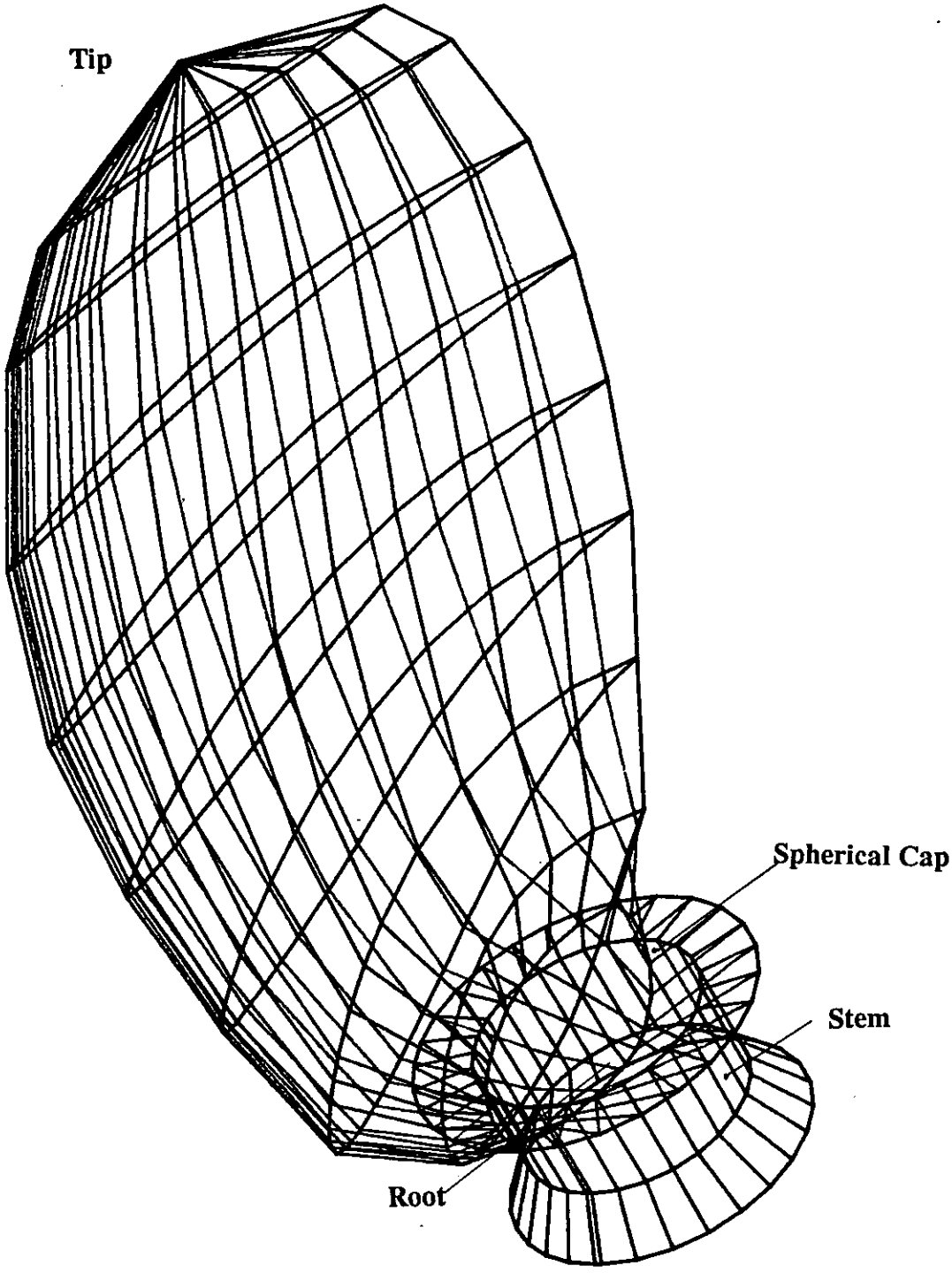


Figure 4 SPLOT output of Single Propeller Blade

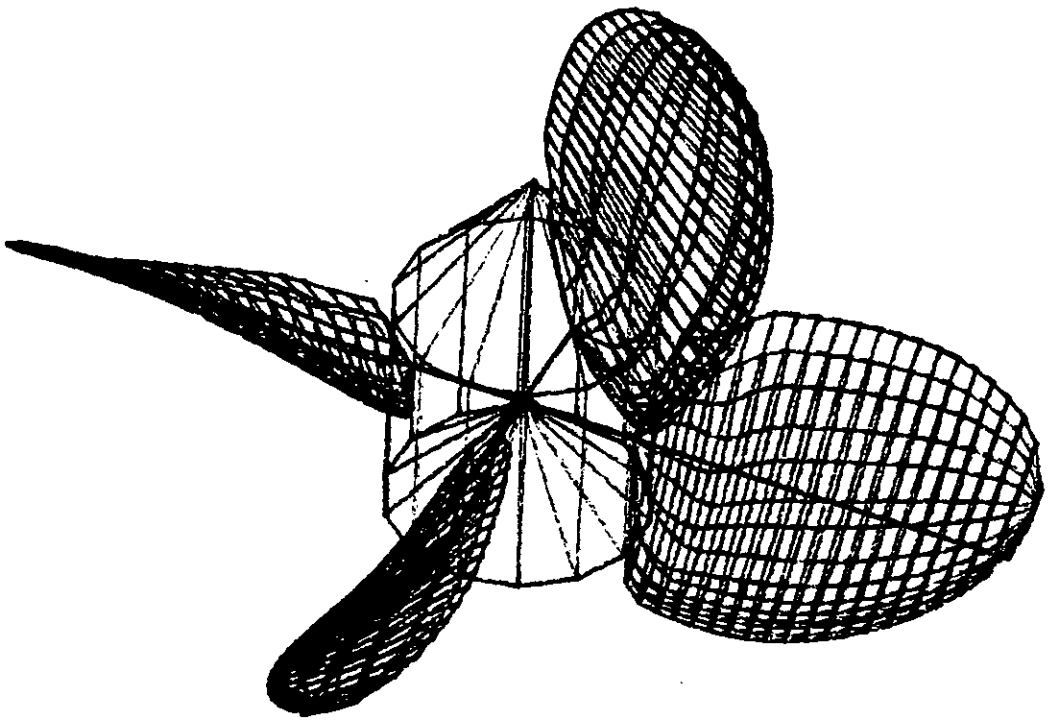


Figure 5 SPLOT output of Four-Bladed Propeller and Hub

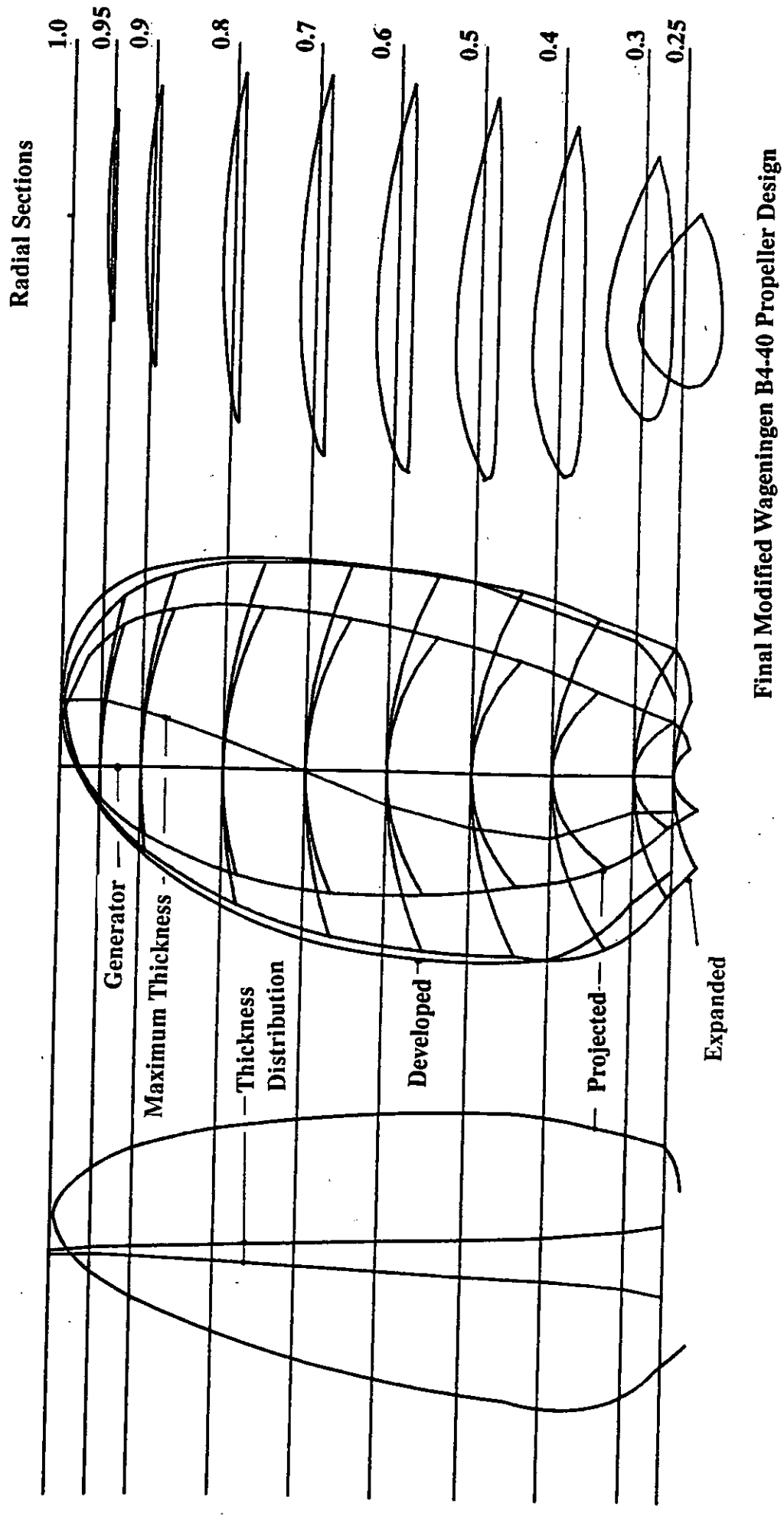


Figure 6 Final Propeller Design Definition

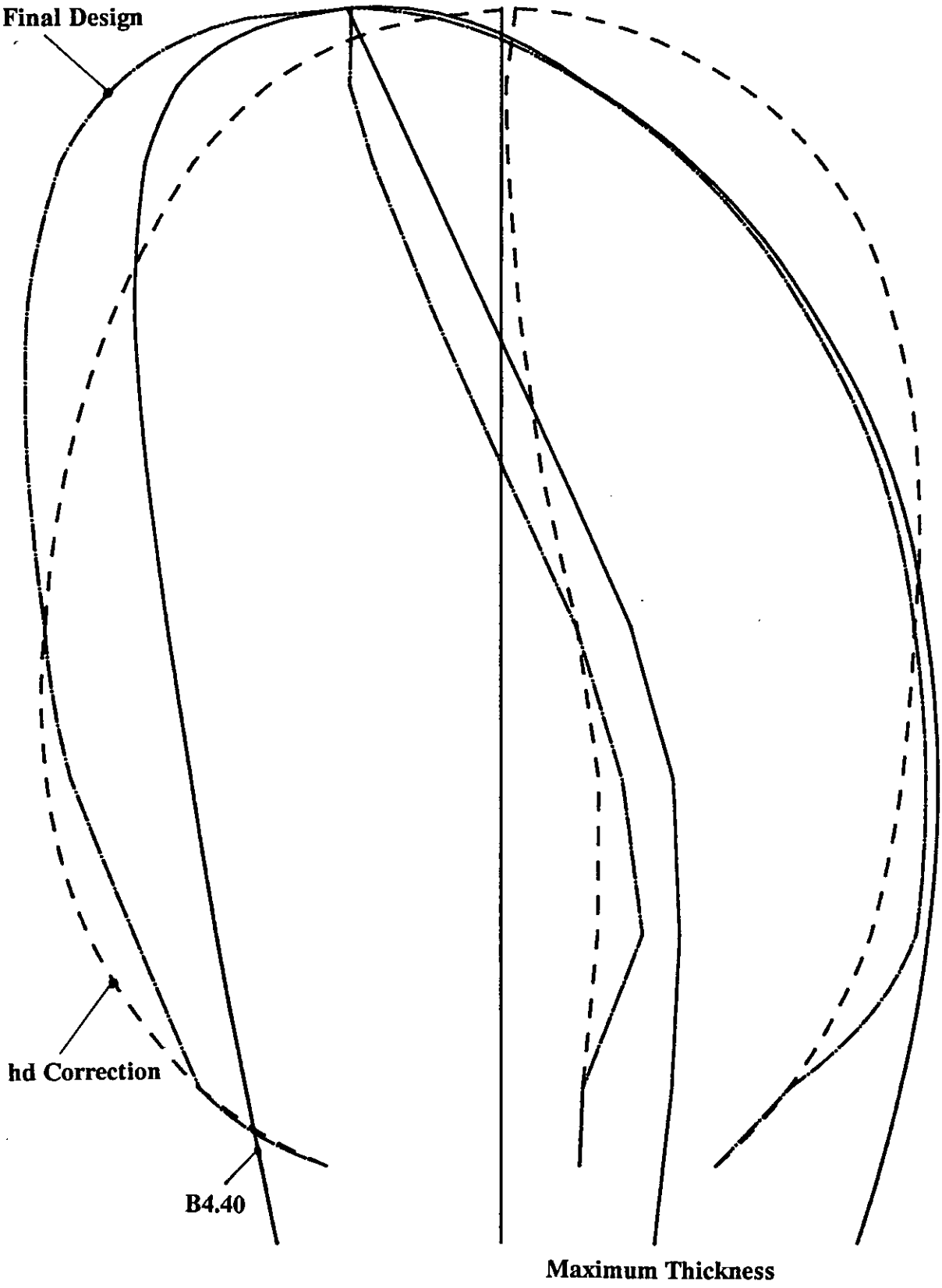


Figure 7 Comparison of Various Modified Propeller Profiles

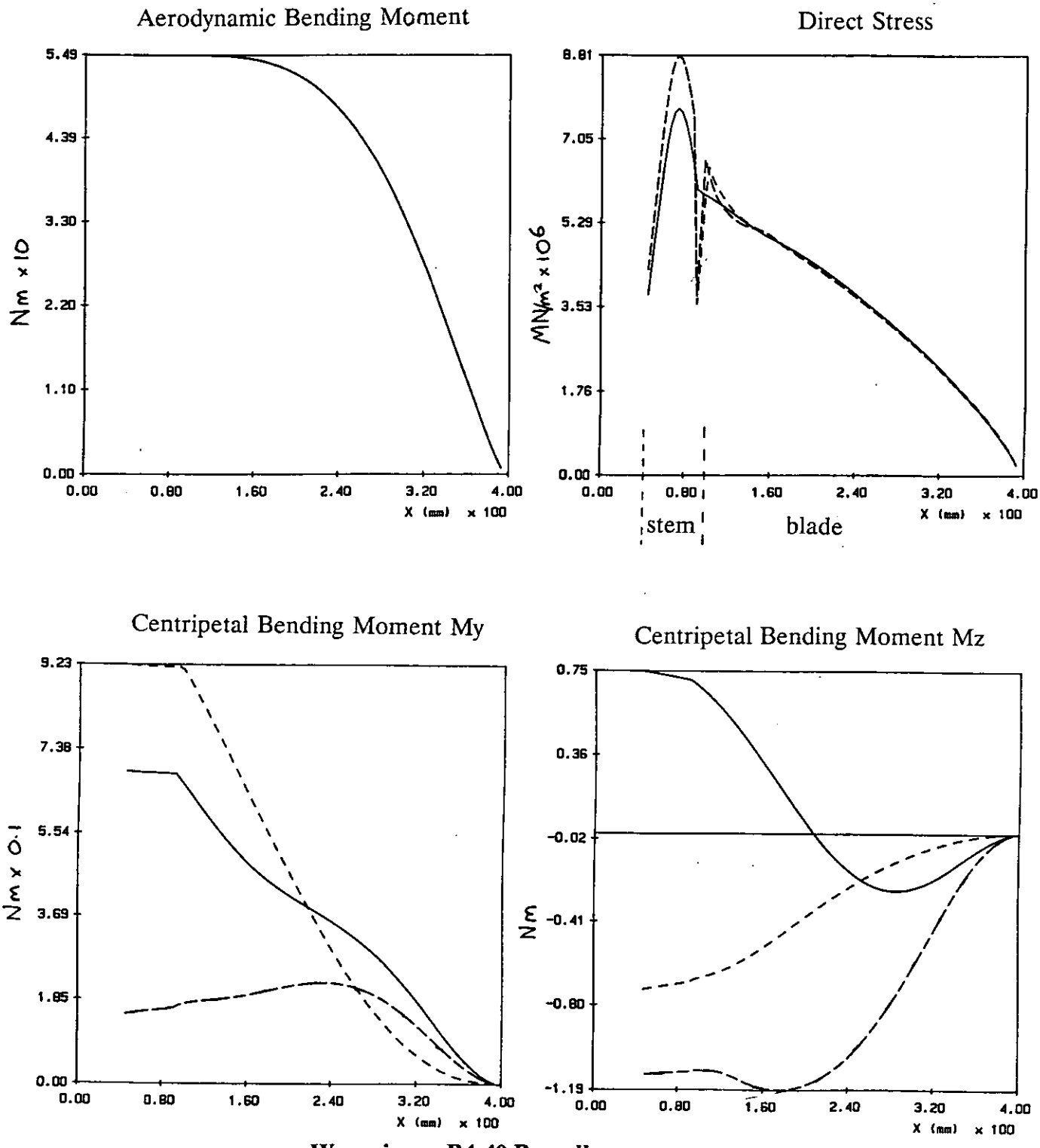
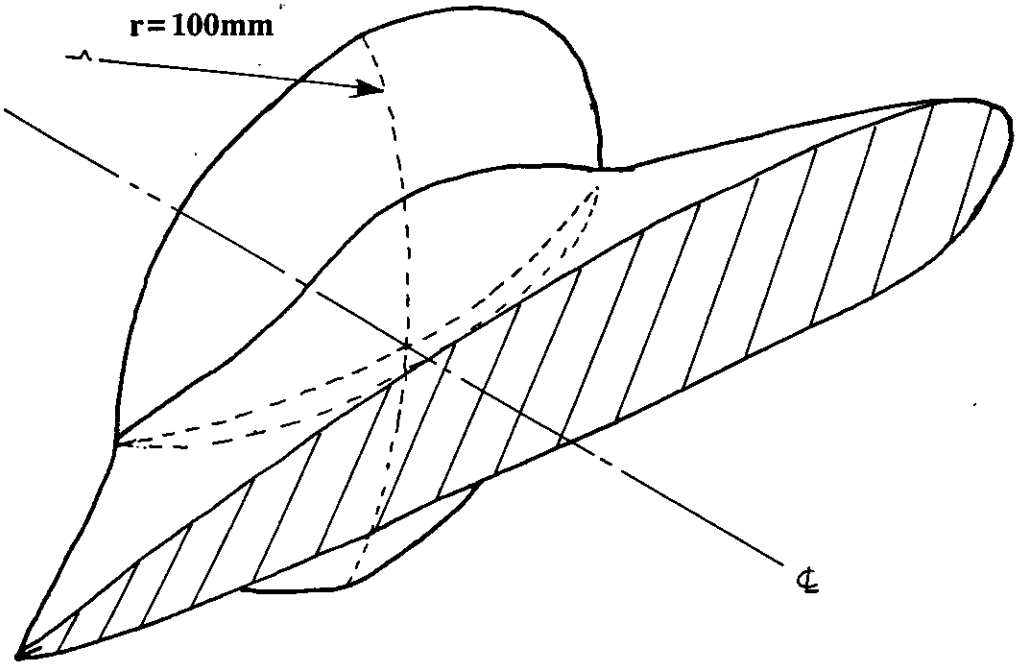


Figure 8 Spanwise distribution of Stress and Bending Moment



Isometric Sketch of Root Detail

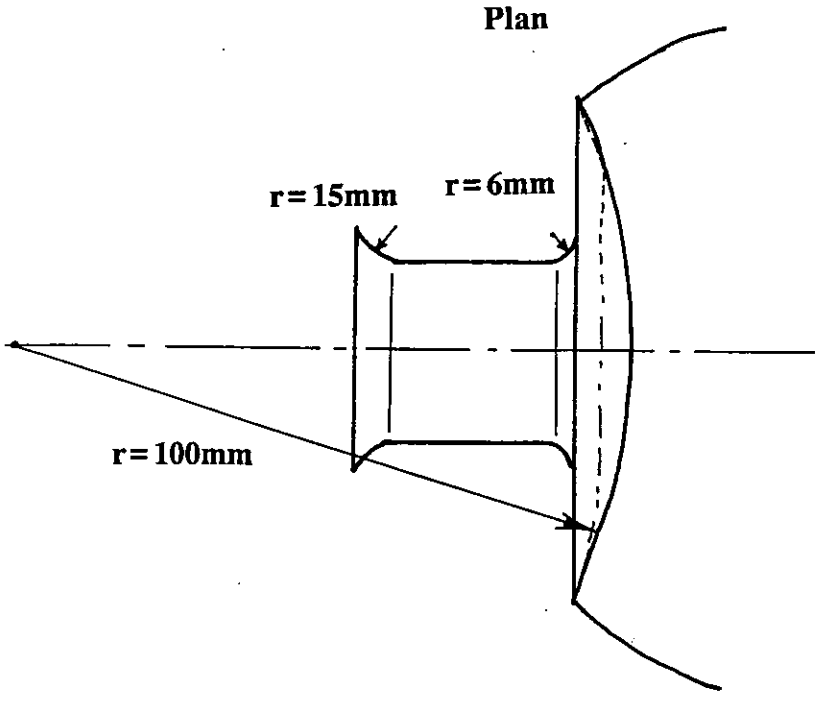


Figure 9 Detail of Blade Root Shape

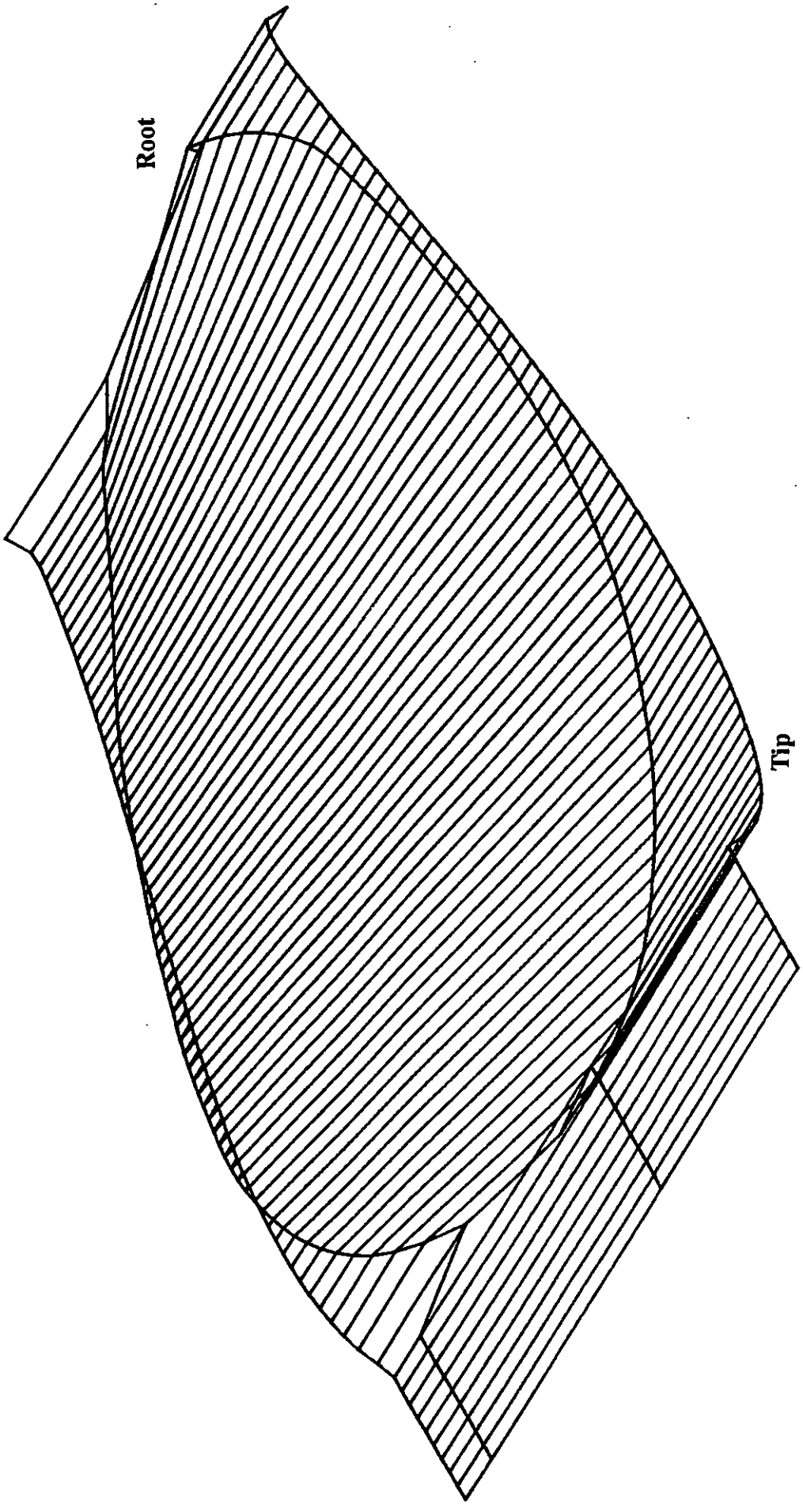


Figure 10 Isometric View of Propeller Mould Split Plane

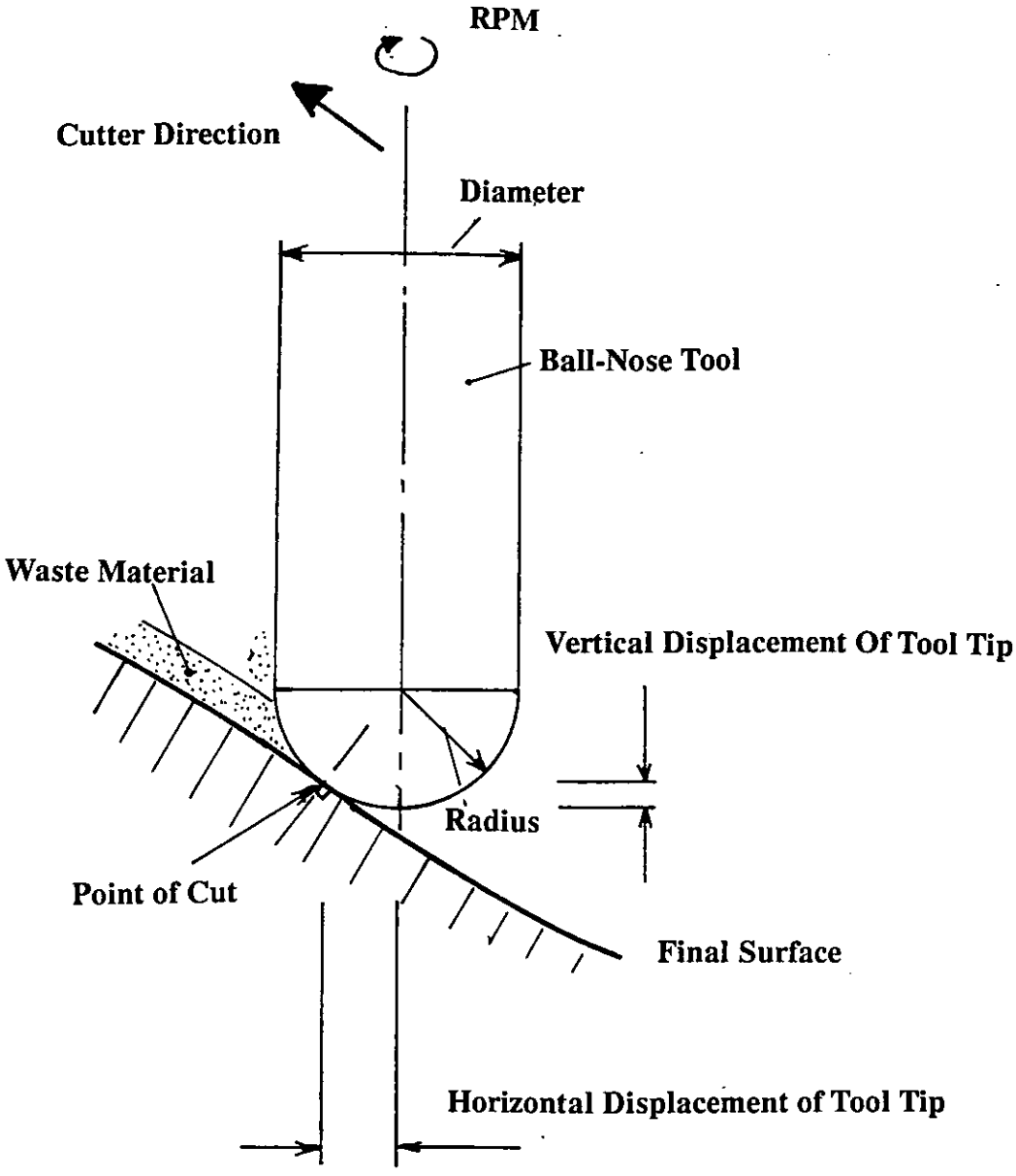


Figure 11 Offset Distance of Cutter from Cutting Surface

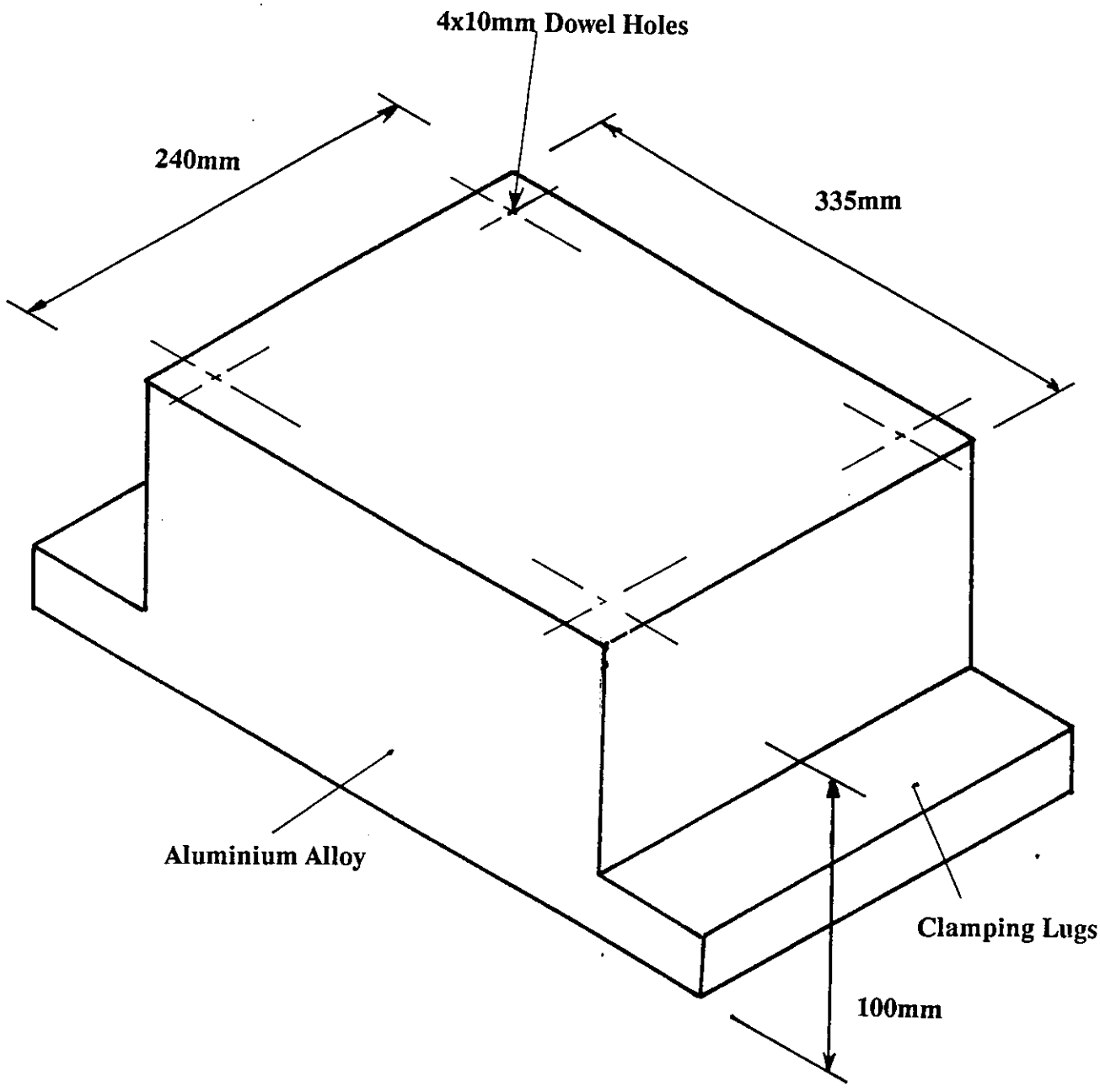


Figure 12 Mould Aluminium Block

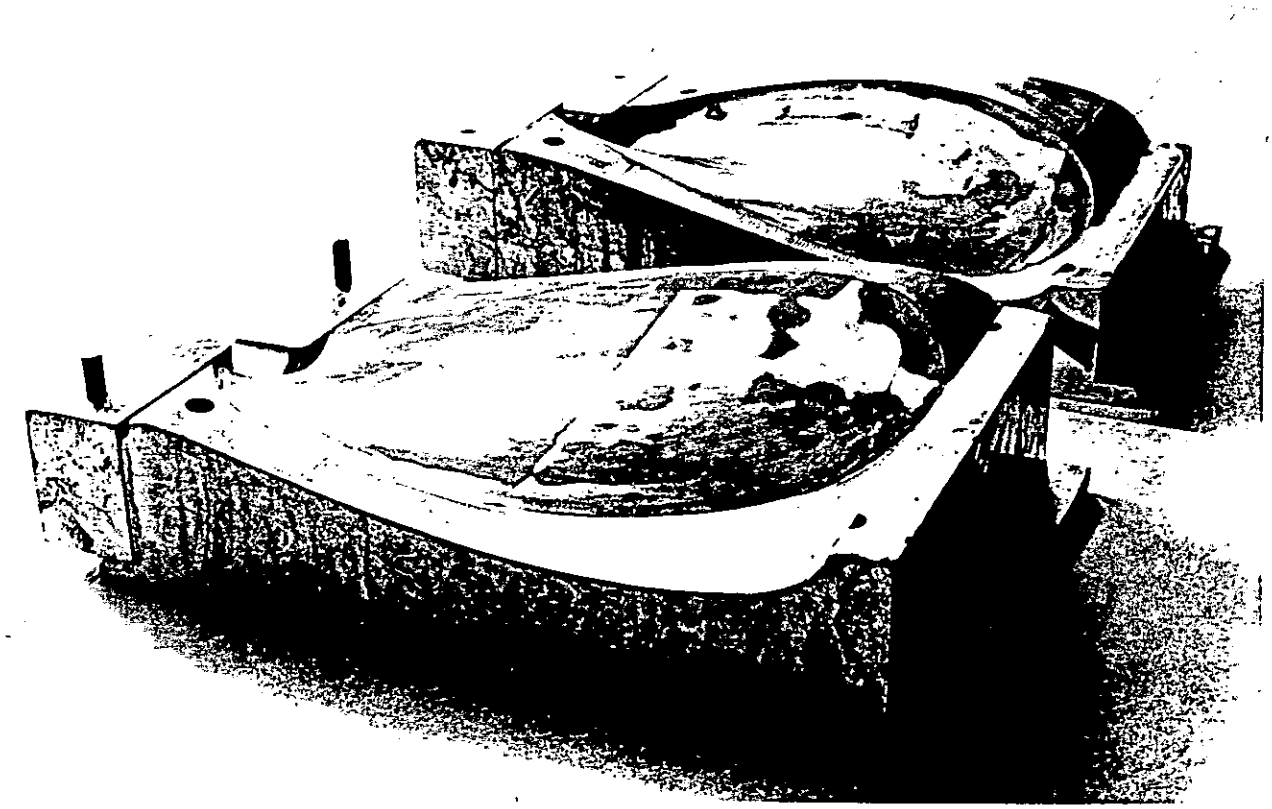


Figure 13 View looking towards Propeller Tip of Finished Moulds

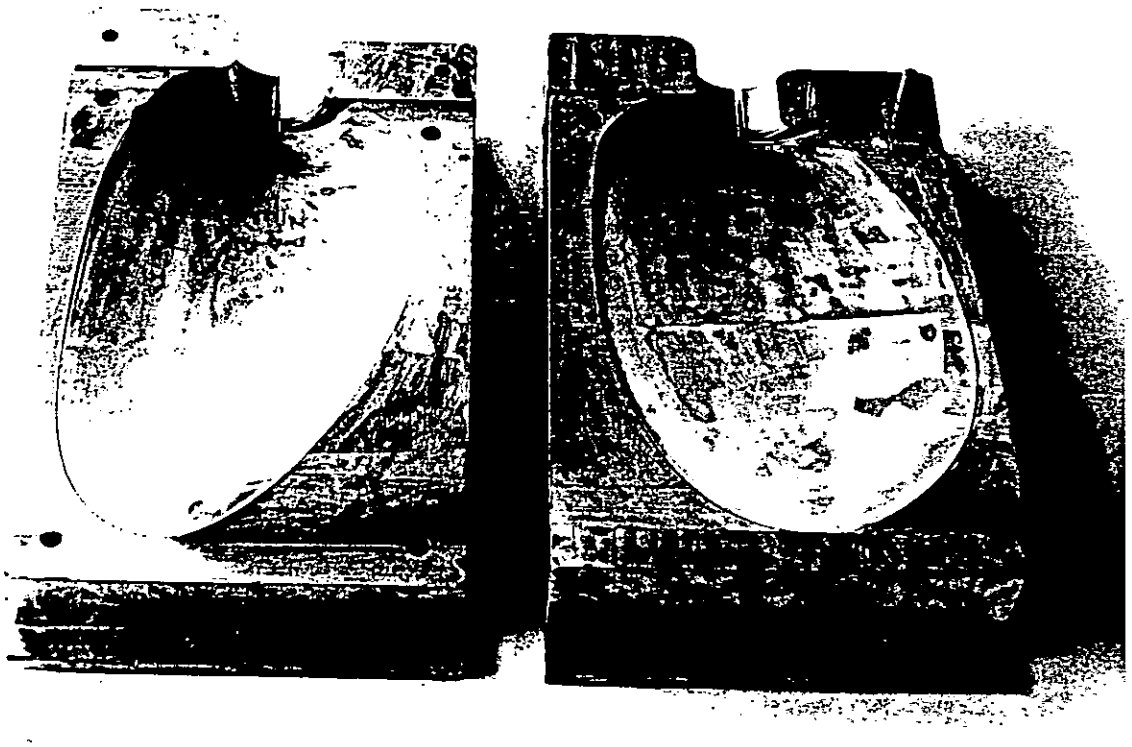


Figure 14 View Looking towards Propeller Root of Finished Moulds

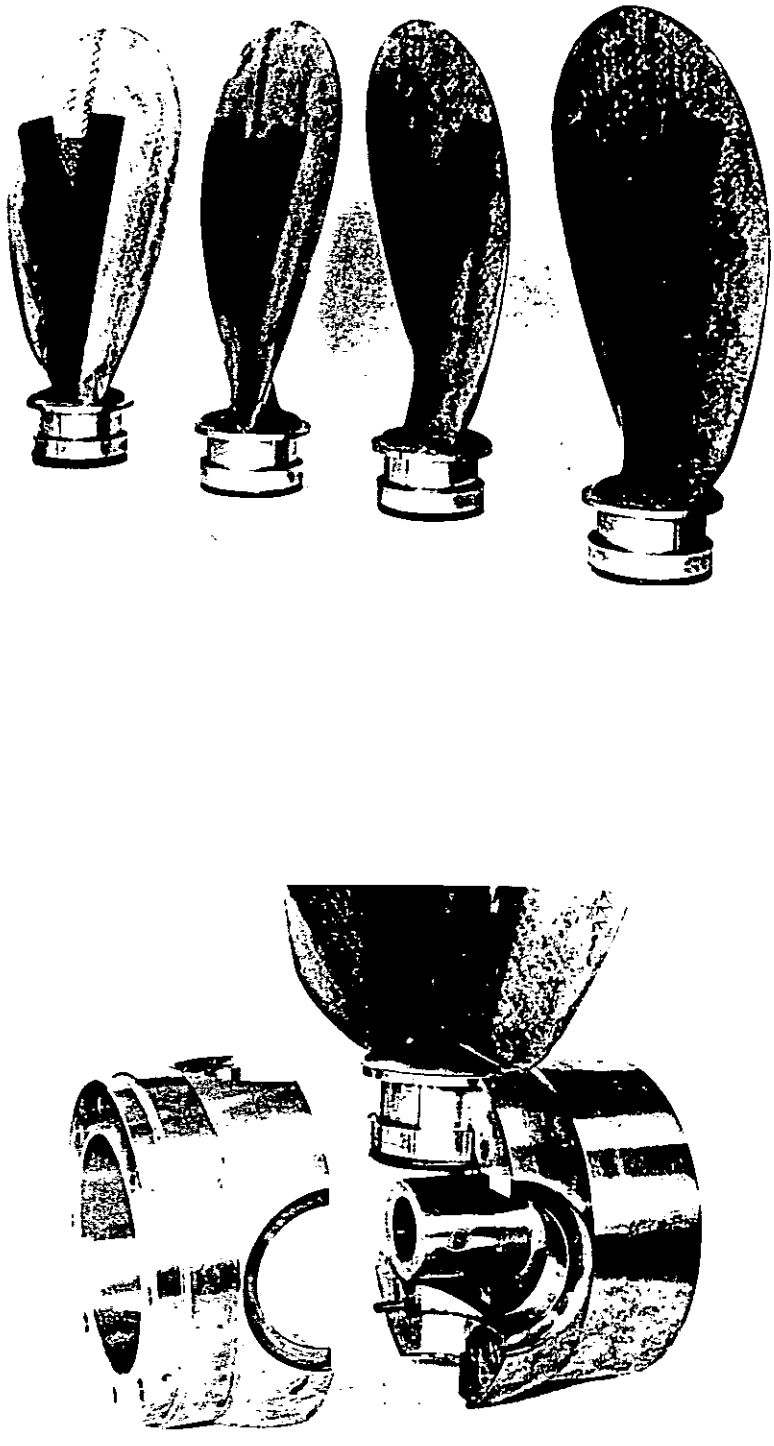


Figure 15 Photograph of Finished Composite Blades.