GENERATION OF COMPUTATIONAL MESHES FOR BEM SIMULATIONS OF MARINE PROPULSORS USING THE SOFTWARE CATIA

A Diploma Thesis by

Dagkli Vasiliki

Presented to

The Department of Ship and Marine Hydrodynamics

School of Naval Architecture and Marine Engineering

National Technical University of Athens

Athens, February 2016

Table of Contents

Part I. Intro

The quest for efficiency leads the propulsor designers to adopt more complex geometries, leading to ever higher demand for integration of simulations to the design process. Boundary Element Methods (BEM) have already been tried and tested for the first steps in the initial propeller design. However, the propeller design has been a well-set and addressed problem, but more complex geometries pose a difficulty in their representation. An approach to this problem is proposed in the current thesis

Modern propulsors have evolved beyond the simple propeller. Even the propeller's design itself has increased in complexity, with design elements such as extreme skew, tip unloading, contracted tip and so on. Added to that are geometries, such as stators or ducts. Its counterpoint, non-conventional propulsion configurations, such as podded propulsors and ducted propellers are more often than not quite complex in themselves.

Boundary Element Methods, as a formulation, can handle any (within limitation) geometry, but the programs currently available work with mesh geometries produced by in-house programs, based on algebraic meshing methods. These are effective, fast and good at creating known geometries, such as foils, wings and propeller blades. However, input of new geometry to an existing simulation requires new code with which to create the them. This also means that the designer ought to be a programmer, in order to write the new code that will produce the new grid geometry.

On the other hand, CAD programs can produce any geometry the designer wants. Additionally, several of current CAD programs offer built-in meshing capabilities that can deal with any complex geometry. Consequently, the role of the designer could be decoupled from that of the programmer so long as the meshes are produced in a format compatible to the BEM solver. Additionally, some CAD programs also have parametric design capabilities. Considering that the existing geometry generation code is parameterized to a great extent, parametric CAD can be a suitable substitute.

Taking the above under consideration, it is possible to use a CAD software to produce a grid input for the BEM code using the software's built-in mesher, and to do so in a parametric way that will enable rapid geometry creation, thus bridging the gap between CAD and BEM.

In this direction, the following steps were taken:

a) Selection of a program, taking under consideration the BEM requirements and limitations, and design intent.
b) Acquaintance with the program of choice. This includes looking into the format in which the grid is exported and detection of any incompatibilities with the BEM program
c) Creation of parametric models and meshes

d) Creation of an interface program that translates the mesh from the exported format to the input format for the BEM program
e) Utilization of the Animation code, written by V. Tsarsitalidis (Tsarsitalidis 2015)
f) Input in UBEM
g) Execution of test applications

Product of this work is the introduction of a methodology that enables the designer to create complex geometries in CAD and input in UBEM with added complex motions.

For the development of the meshing method, 3 distinct cases were investigated:

i) Creation of a model and mesh with the more traditional approach of provided offsets: for this the Wageningnen B-Series (Oosterveld and van Oossanen 1975) was selected, since it is one of the most well-documented propeller series and is already described in a parametric manner.
ii) Creation of a model and mesh without a fully established geometry: a podded propulsor was created, based on a design concept inspired by existing podded propulsors
iii) Creation of mesh from a third-party surface geometry: an external geometry in the IGES format, provided by SVA for the SMP '15 Cavitating Propeller Workshop, was used

The work on the propeller and pod resulted in the creation of parametric models that can be modified by any user, by changing the parameter values in the accompanying excel files.

In addition, parametric mesh models were created for all cases, that can be easily modified through external excel files. The resulting meshes can then be exported, translated using the developed interface program and input in UBEM.

To test the method, a set of meshes were created for each of the aforementioned cases and run with UBEM. The results were compared to the existing data for evaluation.

Part II. The UBEM program

The Boundary Element solver UBEM, initially created by Politis (2004), is a well-established solver and is the tool that will be used for the simulations.

In order to make a choice regarding the CAD program, its requirements and need to be defined. The method is roughly described below.

II-1 Formulation

The formulation of the boundary element method used in the solver code UBEM is thoroughly discussed by Politis (2011). A general overview is presented here.

II-1.1 Geometry

In the UBEM formulation, the geometry is built of complex systems of bodies using surface patches. Each patch consists of a number of bilinear quadrilateral elements. Two types of patches are allowed: lifting patches and non-lifting patches. By combining patches, lifting and/or non-lifting bodies are built.

Lifting surfaces are distinguished from the non-lifting by supplying to the former a flow separation line, defined by the user. Each separation line is the generator of a free shear layer. On each free shear layer, two distinct parts can be considered: a) the strip directly adjacent to the line (the line is bound to the surface), which is called the "Kutta strip" and b) the remaining part of the shear layer. The free shear layer surface at each time t is expressible as a Boolean union of all the Kutta strips and all the remaining parts of the shear layer.

Let M denotes the number of patches in the system. Introduce an index set M ≡ (1,2,…,M) for all the patches. Let $SB^n(t),\ n \in M$ denotes the defining surface of the *n*th patch at time *t*. The flow separation line is denoted by $L^n(t)$ where $n \in M'$ and $M' \subset M$ denotes the an index subset of M characterising the lifting patches: $L^n(t) \in SB^n(t), n \in M'$. Each separation line $L^n(t)$ is the generator of a free shear layer. Kutta strip is denoted by $SK^n(t), n \in M'$. Thus, each free shear layer surface at time t is expressible as $SK^n(t) \cup SF^n(t), n \in M'$.

Total system surface at time t:

$$SB(t) = \cup_{n \in M} SB^n(t) \qquad (1)$$

Total system Kutta strip surface at time t:

$$SK(t) = \cup_{n \in M'} SK^n(t) \qquad (2)$$

Total system free shear layer surface at time t, excluding Kutta strips:

$$SF(t) = \cup_{n \in M'} SF^n(t) \qquad (3)$$

II-1.2 Velocity and potential representation theorems

For the definition of velocities, an inertial (built in earth surface) frame of reference is used. A corresponding coordinate system (assumed Cartesian-orthogonal) is denoted by OXYZ.

As a result of the unsteady motion of the system of bodies, in the region outer of them, there exists a velocity potential φ. At each timestep, it is expressible through its traces $\Phi, \nabla\Phi$ on the boundary points $Q \in SB(t) \cup SK(t) \cup SF(t)$.

Introduce:

$$F(P) = -\frac{1}{4\pi}\int_{SB(t)} \frac{\vec{n}\cdot\nabla\Phi}{r}dS + \frac{1}{4\pi}\int_{SB(t)} \Phi\frac{\overrightarrow{n}\cdot\vec{r}}{r^3}dS + \frac{1}{4\pi}\int_{SK(t)} \mu\frac{\overrightarrow{n}\cdot\vec{r}}{r^3}dS + \frac{1}{4\pi}\int_{SF(t)} \mu\frac{\overrightarrow{n}\cdot\vec{r}}{r^3}dS \tag{4}$$

$$\vec{H}(P) = \frac{1}{4\pi}\int_{SB(t)} (\vec{n}\cdot\nabla\Phi)\frac{\vec{r}}{r^3}dS + \frac{1}{4\pi}\int_{SB(t)} (\vec{n}\times\nabla\Phi)\times\frac{\vec{r}}{r^3}dS + \frac{1}{4\pi}\int_{SK(t)} \vec{\gamma}\times\frac{\vec{r}}{r^3}dS + \frac{1}{4\pi}\int_{SF(t)} \overrightarrow{\gamma\times}\frac{\vec{r}}{r^3}dS - \frac{1}{4\pi}\int_{L'(t)} \mu\frac{d\vec{l}\times\vec{r}}{r^3} \tag{5}$$

Where P is the evaluation point (or control point) for either F or$\vec{\mathrm{H}}$, $\vec{\mathrm{n}}$ is a unit normal vector at the boundary integration point $Q \in SB(t) \cup SK(t) SF(t)$ showing inside the flow region, $\vec{r} = \overrightarrow{QP}, r = |\overrightarrow{QP}|$, μ is the dipole intensity with support on $SK(t) \cup SF(t)$ and $\vec{\gamma}$ is the corresponding (to μ) vorticity intensity given by

$$\mu = \Phi^+ - \Phi^- \tag{6}$$
$$\vec{\gamma} = \vec{n}\times\nabla\mu \tag{7}$$

Finally, $L'(t) = \partial\big(SK(t) \cup SF(t)\big) - L(t)$, where $L(t) = \cup_{n\in M'} L^n(t)$ (the free part of the line bounding the free shear layers)

With the aid of relations (4) and (5), representations for $\Phi, \nabla\Phi$become

$$\left.\begin{aligned}\Phi(P) &= F(P)\\ \nabla\Phi(P) &= \vec{H}(P)\end{aligned}\right\} P \notin (SB(t) \cup SK(t) \cup SF(t)) \tag{8}$$

$$\left.\begin{aligned}\Phi(P) &= \frac{1}{2}\Phi(P) + F(P)\\ \nabla\Phi(P) &= \frac{1}{2}(\vec{n}\cdot\nabla\Phi)\cdot\vec{n} + \frac{1}{2}(\vec{n}\times\nabla\Phi)\times\vec{n} + \vec{H}(P)\end{aligned}\right\} \Leftrightarrow \left\{\begin{aligned}\frac{1}{2}\Phi(P) &= F(P)\\ \frac{1}{2}\nabla\Phi(P) &= \vec{H}(P)\end{aligned}\right\} P \in SB(t) \tag{9}$$

$$\left.\begin{aligned}\Phi^{+.-}(P) &= \pm\frac{1}{2}\mu(P) + F(P)\\ \nabla\Phi^{+.-}(P) &= \pm\frac{1}{2}\vec{\gamma}(P)\times\vec{n}(P) + \vec{H}(P)\end{aligned}\right\} P \in SF^{+.-}(t) \tag{10}$$

And similarly for$SK^{+.-}$. In relation (10) the superscripts (+ .-) denote the two sides of the free shear (or vorticity) layer surfaces. While the unit normal $\vec{n}$ is directed from (-) to (+)

II-1.3 The integral equation

Let $\overrightarrow{v_A}$ denotes the velocity to the boundary point $A \in SB(t)$ and $\vec{n}$ a unit vector normal to body surface at A with direction pointing into the flow region. Then, the no-entrance conditions at A has the form:

$$\nabla\Phi\cdot\vec{n} = \overrightarrow{v_A}\cdot\vec{n} \tag{11}$$

Substituting (11) to the first of eq. (9) and using (4) we get:

$$\begin{aligned}\frac{1}{2}\Phi(P) - \frac{1}{4\pi}\int_{SB(t)}\Phi\frac{\vec{n}\cdot\vec{r}}{r^3}dS - \frac{1}{4\pi}\int_{SK(t)}\mu\frac{\vec{n}\cdot\vec{r}}{r^3}dS\\ = -\frac{1}{4\pi}\int_{SB(t)}\frac{\vec{n}\cdot\vec{v}_A}{r}dS + \frac{1}{4\pi}\int_{SF(t)}\mu\frac{\vec{n}\cdot\vec{r}}{r^3}dS, P \in SB(t)\end{aligned} \tag{12}$$

This is a second kind Fredholm type Cauchy singular boundary integral equation for the determination of Φ and μ on points of *SB(t)* and *SK(t)*, respectively. In the right hand side of (12) the first term is a known integral (as far as the motion of the system of bodies is known) and the second term is known from the solution of the problem at previous time steps. The unknowns in the left hand side of (12) are the potential Φ on *SB(t)* and the dipole intensity μ on *SK(t)*. For their determination the additional required condition is the Kutta condition at the separation lines (trailing edges in case of wing flow w/o separation).

II-1.4 Kutta condition

Let the point $A \in SB(t)$. Let $(d/dt)|_A$denotes the time derivative for an observer built to the point A of the moving body and let $\overrightarrow{v_A}$ denotes the known velocity of A. Then unsteady Bernoulli equation takes the following form:

$$\frac{p - p_\infty}{\rho} = -\left.\frac{d\Phi}{dt}\right|_A - \frac{1}{2}(\nabla\Phi - \vec{v}_A^2) + \frac{1}{2}\vec{v}_A^2 \tag{13}$$

According to a pressure type Kutta condition, as we approach the trailing edge point from either pressure side (superscript +) or suction side (superscript -), the pressure should be continuous i.e.

$$p^+ = p^- \tag{14}$$

Using (13), this becomes a quadratic (nonlinear) relation between $\Phi^+, \nabla\Phi^+, \Phi^-, \nabla\Phi^-$. Assuming steady linearized flow, Bernoulli equation degenerates to the famous Morino condition

$$\mu_{approachingL(t)fromshearlayer} = (\Phi^+ - \Phi^-)_{approachingL(t)frombodypoints} \quad (15)$$

Which is a linear equation in *Φ, μ.*

II-1.5 Shear layer dynamics

Kinematic and dynamic conditions on a free vortex sheet expressed in terms of the dipole intensity of the sheet results in the following equation Politis (2004):

$$\frac{D\mu}{Dt} = 0 \quad (16)$$

Where D/Dt denotes a material derivative for μ, based on the mean perturbation velocity of the shear layer. Mean perturbation velocity $> \vec{v} <$ on points of the shear layer can be found using (10)

$$> \vec{v} < = \frac{\nabla\Phi^+ + \nabla\Phi^-}{2} = \vec{H}(P) \quad (17)$$

So, relation (16) becomes

$$\frac{D\mu}{Dt} = \frac{\partial\mu}{\partial\tau} + (> \vec{v} < \cdot \nabla)\mu = \frac{\vartheta\mu}{\vartheta t} + \left(\vec{H}(P) \cdot \nabla\right)\mu = 0 \quad (18)$$

II-1.6 Calculation of forces, moments, power and efficiency

Pressure forces on element centroids can be calculated using (13). The developed code also contains a simple subroutine for the calculation of viscous effects, using an elemental drag coefficient provided by the user. We can then integrate the pressure and viscous drag forces to find total body and/or system forces or moments around any given point or axis. In the current computer code a moving coordinate system is introduced, with position and orientation defined by the user, which is used for the interpretation/representation of instantaneous forces and moments.

II-2 Discretization and solution

Subdivide $SB(t), SK(t)$ and $SF(t)$ into N_B, N_K and N_F elements respectively. Four node quadrilateral elements have been used for the subdivision of bodies and shear layer boundaries. Assume piecewise constant Φ and $\vec{n} \cdot \nabla\Phi$ for all elements on $SB(t)$

Assume piecewise constant μ for all elements on$SK(t) \cup SF(t)$. Denote these constant values by $\Phi_i, \sigma_i(= (n \cdot \nabla\Phi)_i), \mu_i$ where the range of index (i) is adapted accordingly

With the aid of the previous assumptions/notation integral Eq. (12) becomes

$$\frac{1}{2}\Phi_i - \sum_{j=1,N_B} B_{ij}\Phi_j - \sum_{j=1,N_K} B_{ij}\mu_j = \sum_{j=1,N_B} A_{ij}\sigma_j + \sum_{j=1,N_F} B_{ij}\mu_j \qquad (19)$$

Where

$$A_{ij} - -\frac{1}{4\pi}\int_{E_j} \frac{dS}{|QP_i|}, B_{ij} = \frac{1}{4\pi}\int_{E_j} \frac{\vec{n}(Q)\cdot\overrightarrow{QP_i}dS}{|QP_i|^3}$$

And E_j denotes the surface of the *j*th element from either $SB(t)$, $SK(t)$, $SF(t)$ and P_i denotes the *i*th control point (centroid ofE_i) on $SB(t)$.

Relation (19) applied at the $N_B$centroids of the body elements, gives N_B linear equations for the determination of element potentials. The N_K additional equations required for the calculation of $\mu_j$are taken from the satisfaction of the Kutta condition on $L(t)$. As already said, the computer code can implement three alternatives for the satisfaction of a Kutta condition, and thus completion of the system of equations:

(i) The first alternative uses a linear Morino condition in the form of (15), which in discretised form becomes: $\mu_i = \Phi_{i^+} - \Phi_{i^-}$, where i^+ and i^- denote element numbers on body, neighboring to trailing edge from different sides (i.e. pressure side and suction side to use terminology from wings) and i denotes element number on Kutta strip, neighbouring to the same point of the trailing edge

(ii) The second alternative uses a pressure type Kutta in the form of Eq. (14). This is a nonlinear equation between the unknowns Φ_j and μ_j. There are two ways to implement numerically the nonlinear pressure type Kutta. The first is to express velocities in (13) as functions of the unknowns Φ_j, μ_j using the representation theorems (9). The second is to evaluate velocities from surface potentials by using finite differences in a body surface curvilinear system. The second method was used, since the computational cost for calculating velocity induction factors needed in the first formulation, is high.

(iii) The third alternative uses a mixed type Kutta i.e. partly Morino and partly Pressure type.

If alternatives (ii) or (iii) have been selected, the resulting system of equations is nonlinear and it is solved by using a Newton iteration method, with starting value taken from a Morino-type Kutta (first alternative). After the system of equations has been solved, the code calculates and prints forces, moments, power and efficiency. It also prepares a number of graphic output files ready to be used by the commercial program TECPLOT to visualize the complex unsteady phenomena in video form.

Solution of the problem is implemented by a time stepping algorithm as follows:

At each time step, the SOLVER-CODE:

1. Reads the next position of the system of bodies, which is calculated by a GPP/MPP (Geometric Preprocessor Program (GPP), Motion Preprocessor Program (MPP)—explained in the next section)
2. Generates corresponding Kutta strips, for the case of lifting bodies, thus introducing the extra unknowns required for the Kutta condition satisfaction.
3. Solves the system consisting of the "no-entrance" and "Kutta" conditions. In case of pressure type Kutta Newton iteration is used at this step.
4. Deforms the free shear layers to their new positions by applying a special filtering technique to calculate $> v <$, Eq. (17).
5. Output results (pressures, forces, velocities and position of free shear layers) for this time step.
6. Proceeds to the next time step and repeats the calculation from step (1).

II-2.1 The GPP's and MPP's

According to UBEM architecture a body (lifting or non-lifting) can be built by a number of patches. Thus, for each case to be modelled, a specialized GPP (Geometry Preprocessing Program) and MPP (Motion Preprocessing Program) has to be developed.

The MPP code used was developed by V. Tsarsitalidis. As shown in Fig. II-1, for the systematic exploration of parameters of movement to performance characteristics, a sequence of simulations is executed. The initial geometry parameters have to be given, along with a set of fixed parameters and the parameters that will be systematically changed in the sequence. For each step, an intermediate subprogram gives the specific parameters to the geometry pre-processing program, which in turn makes the time history (animation) of movement to be input for the solver. The solver in turn, exports result files signified by the parameters of each simulation. Lastly, an insert data collection subprogram gathers user-defined data from all iterations and exports them into a single file (for each set of variables) in order to make it easy to extract summary plots for the whole set of systematic runs performed.

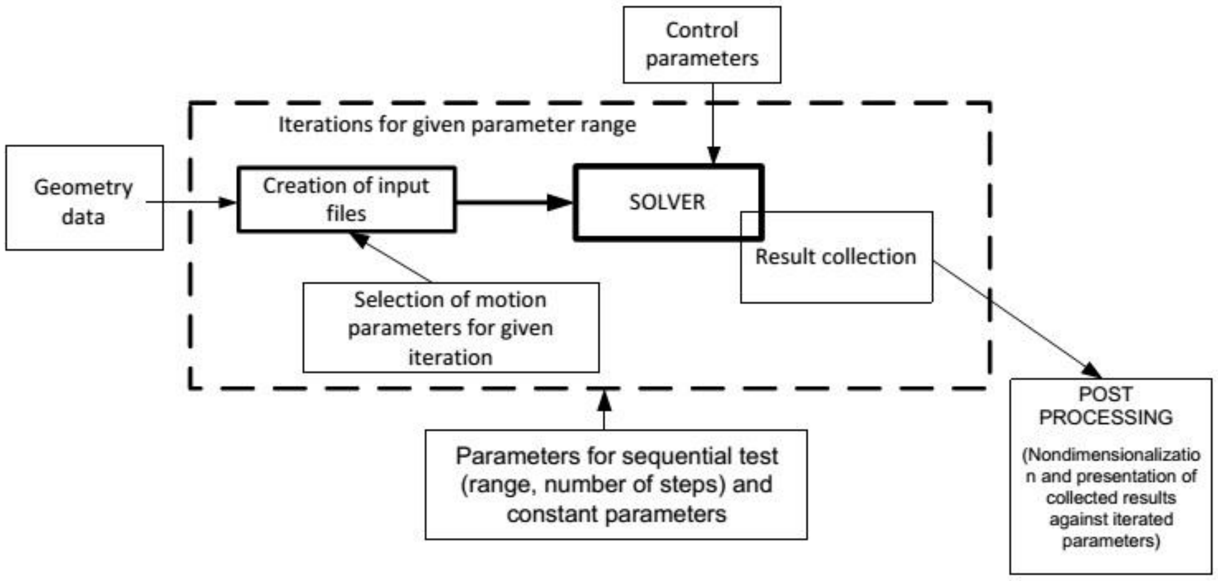

Control parameters
Iterations for given parameter range
Geometry data
Creation of input files
SOLVER
Result collection
Selection of motion parameters for given iteration
Parameters for sequential test (range, number of steps) and constant parameters
POST PROCESSING
(Nondimensionalizatio n and presentation of collected results against iterated parameters)

Fig. II-1 Schematic of used program architecture. Source: (Tsarsitalidis 2015)

<u>Input file generation</u>

A file containing the description of geometry (in animated panel mesh form), is necessary for the solver. Since the architecture is modular, input file generators are built for this purpose. Fig. II-2 depicts the structure of the in-house program that has been created for wing simulations. For the current application, the Geometry Data generation is provided by the developed method, using the meshes created from the parametric models and translated with the interface program.

<u>Animating a given geometry.</u>

The Motion pre-processing subprogram, is actually superimposing user defined rigid (rotations and translations of the whole body) and flexing motions. If flexing motions are employed, they are performed before the rigid motions. In our case, no flexing motions are used, so the rigid body motions calculation will be the only one explained.

Rigid body motions

In the most general case an instantaneous motion is a result of an instantaneous rotation around a given axis and an instantaneous translation along another given axis. For the minimization of errors, instead of incremental motion (i.e. moving the geometry from the position of the previous step to that of the next), a direct approach is taken, where one rotation and one translation is applied to the initial geometry at each step in order to arrive at the desirable position. The movements of each point are done by means of vector translation and rotation relative to given point and axis or the absolute centre of reference using the aforementioned linear algebra/geometry functions. Then, the definition of motions is a matter of defining the rotation angle and vector, and the translation vector in time.

II-3 Resulting mesh requirements

From the UBEM formulation, the following requirements for grid creation arise:

- Mesh curves should be smooth and with smooth derivatives
- Mesh line distribution should be denser in domain areas where higher calculation errors are expected
- Mesh element size distribution should be smooth (no big discrepancies in size between neighbouring grid elements)
- Mesh elements should be as rectangular as possible

However, fulfilment of above requirements is not always feasible. Domain boundary geometry plays a major part and might be prohibiting to achieving smooth curves or rectangular grids. In addition, geometric fidelity or smoothness cannot always be achieved without leading to solutions that are too demanding either in time or data volume.

Consequently, it is obvious that a balance must be struck between requirements and resulting grid.

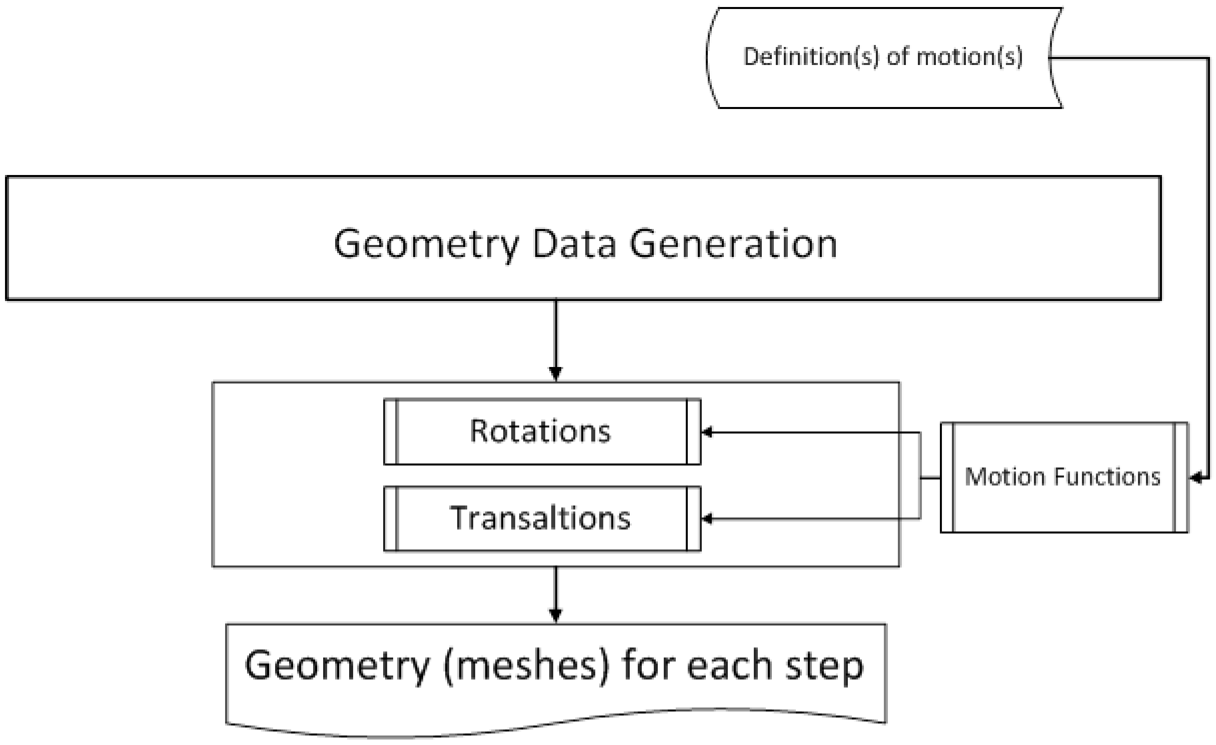


Fig. II-2 Schematic of geometry generation program architecture Source: (Tsarsitalidis 2015)

Part III. CAD software review

As a consequence of the aforementioned requirements, the CAD program needs to:

1) Produce any required geometry, regardless of complexity
2) Have parametrization capabilities
3) Have its own built-in mesher that can create structured grids

Additionally, the software should be available for student use.

The following software is checked against the requirements:

- Rhinoceros 3D
- Autodesk inventor
- Solidworks
- CATIA
- SIEMENS NX

III-1 Rhinoceros 3D

Rhinoceros 3D, developed by Robert McNeel & Associates, is a freeform surface modeller with solid modelling capabilities. It uses NURBS for the representation of curves and freeform surfaces, and feature-based techniques for solid modelling.

Rhino is not primarily a parametric modelling program. However, the software's architecture enables the creation of scripts and add-ons that expand its capabilities. In this spirit, there are add-ons that offer parametric modelling capabilities (for example Grasshopper and Rhinoworks).

The software is available for student use. Its ease of use combined with its very informative manual renders it a program approachable to beginner users. However, its built-in mesher doesn't offer the choice to construct a structured mesh, which makes it unsuitable for the current application.

III-2 Autodesk Inventor

Inventor, developed by Autodesk, is a solid modelling program with parametric capabilities and high programmability using scripts. The program exists in the school's computers and is also offered free as a student licence by the company itself.

Even though Inventor offers some Finite Element Analysis tools, doesn't have a mesher that can specifically create a structured grid. It is mostly meant to be a tool to create the solid objects that will be imported into a separate mesher and FEA program.

III-3 Solidworks

Solidworks, produced by Dassault Systèmes, is the equivalent of Inventor. Just like its counterpart, it is a solid modelling CAD and CAE program with parametric capabilities. Solidworks includes a mesher, however it only offers the choice of unstructured meshes creation.

III-4 CATIA

CATIA, produced by Dassault Systèmes is actually a 3D Product Lifecycle Management software suite. It supports different stages of a product's development, from design (CAD), to engineering (CAE) and manufacturing (CAM). It is a feature-based geometric and solid modelling software, highly parametric. It shares features with Solidworks from the same company (but with extended capabilities). Its built-in FEA solver includes a native mesher, which offers the option of structured mesh creation and modification. CATIA can also integrate with other programs and is programmable using scripts written in a modified version of Visual Basic.

One major downside of the program stems from the very own thing that sets it aside: due to its many features and capabilities, the learning curve is very steep. To reach a level where they can comfortably design something, users need to dedicate a lot of time learning the program both by reading its extensive manual and practicing. CATIA was also available for student use.

III-5 Siemens NX

Siemens NX is a program suite by Siemens, with similar capabilities to CATIA. Since there was not a student licence available, it was not investigated further.

Evidently, CATIA appears to be the most promising choice. Despite the fact that the program has a steep learning curve, it is the most powerful software for the intended use.

Table III-1 CAD Software Comparison

	Parametric capability	Structured meshes Support	Ease of use	Licence Availability
Rhinoceros 3D	Yes, on conditions	No	Easy	Yes
Autodesk Inventor	Yes	No	Medium	Yes
Solidworks	Yes	No	Medium	Yes
CATIA	Yes, highly	Yes	Difficult	Yes
Siemens NX	Yes, highly	Yes	Unknown	No

Part IV. The software CATIA

CATIA is the CAD/CAM and CAE part of Dassault's PLM (Product Lifecycle Management) software suite. This means that it includes the necessary software to assist in the different stages of product development, including conceptualization, design, engineering and manufacturing. It is a feature-based, parametric and solid modeling design tool that can be used to create fully associative 3-D solid models, while utilizing automatic or user-defined relations to capture design intent.

For the creation of the 3D models and meshes in this thesis, CATIA's CAD and CAE capabilities were used, so these will be presented further.

It is suggested to anyone interested in further learning of the software to carefully read the CATIA documentation, since it's extensive, describes tool capabilities in great detail and includes a lot of examples and user tasks to help the prospective user with learning. In addition, "CATIA V5 Essentials" (Kogent 2009), as well as the "CATIA Fundamentals" training course guide are a very good introduction to using CATIA. A great resource for general solid modelling and CAD system functionality is "Solid Modelling and CAD Systems – How to Survive a CAD System" (Stroud and Nagy 2011). Further insight on parametric modelling can be found in "Parametric Modelling" - Ch. 21 of the Handbook of Computer Aided Geometric Design" (Christoph M. Hoffmann 2002), and on the b-rep geometric representation in "Boundary representation modelling techniques" (Stroud 2006).

IV-1 The workbench concept

CATIA's capabilities are subdivided in a set of products, each having a general theme. Each product is further subdivided into the so-called workbenches. In order to create a CATIA document, the user must select one of the workbenches suitable for the document type. Each workbench contains a toolset dedicated to performing specific tasks. These tools can be anything from curve, surface and solid creation tools, dimensioning tools, file organising tools and features. The tools included in a workbench aren't exclusive, with a lot of workbenches sharing tools between them or even properly including whole other workbenches (Kogent 2009).

The most commonly used workbenches are:

- Part Design: provides a solid modeling approach to design model parts
- Sketcher: used for 2Dprofiles creation which are then used to build 3D geometry
- Wireframe and surface: creation of complex parts and features that include a 3D curve wireframe and/or surface elements
- Assembly Design: creation of assemblies by adding constraints, features and specifications to parts

- Generative Shape Design: creation of surfaces and drawings in connection to part and assembly designs

It has to be noted that the Sketcher workbench is contained in most of the other workbenches. It is the most important workbench, since geometries are generally constructed with one or more sketches as a basis and a lot of the features offered are sketch-based. Apart from the Sketcher, for the scope of this thesis, the workbenches used are the Generative Shape Design, Freestyle, Advanced Meshing Tools and Assembly Design.

The tools will not be fully described, since the aim of this thesis is not to teach use of the software, which is quite expansive, with a lot of capabilities and has, as previously noted, a steep learning curve. Rather than this, the creation of the model will be presented in a step-by-step manner, in order to create the specific model, in the aim of giving the reader a first glimpse of the way CATIA works and a starting point for their further involvement with the software.

IV-2 Features

A CATIA document can be composed of a number of primitives (points, lines) and features. When working in a document, the user adds features (such as pads, pockets, holes etc.), that are directly applied to model.

Features represent a specific combination of attributes and relationships of a product's units and provide an essential package of information for various design tasks and performance analyses. A lot of the basic features generally available are borrowed from the manufacturing process. Consequently, they can be linked to a product's manufacturing knowledge, facilitating manufacturing and process planning. They also provide a framework for organizing design and manufacturing information in a way that can be easily reapplied in future designs.

Most features in CATIA CAM/CAD workbenches can be classified either as sketcher-based or dress-up. Sketcher-based features are based on a 2D sketch. Generally, the sketch is transformed into a 3D solid by actions such as extruding, rotating, sweeping, or lofting. Dress-up features are features created directly on the solid model, such as fillets and chamfers. Dress-up features have their roots in the machining operation, examples of which include extruding, which creates a surface by pulling a curve along a direction, and rotating, which creates a surface by revolving a profile around an axis. On the other hand, dress-up features examples include filleting, holes and slots which are similar to their real-life manufacturing namesakes (Fig. IV-1).

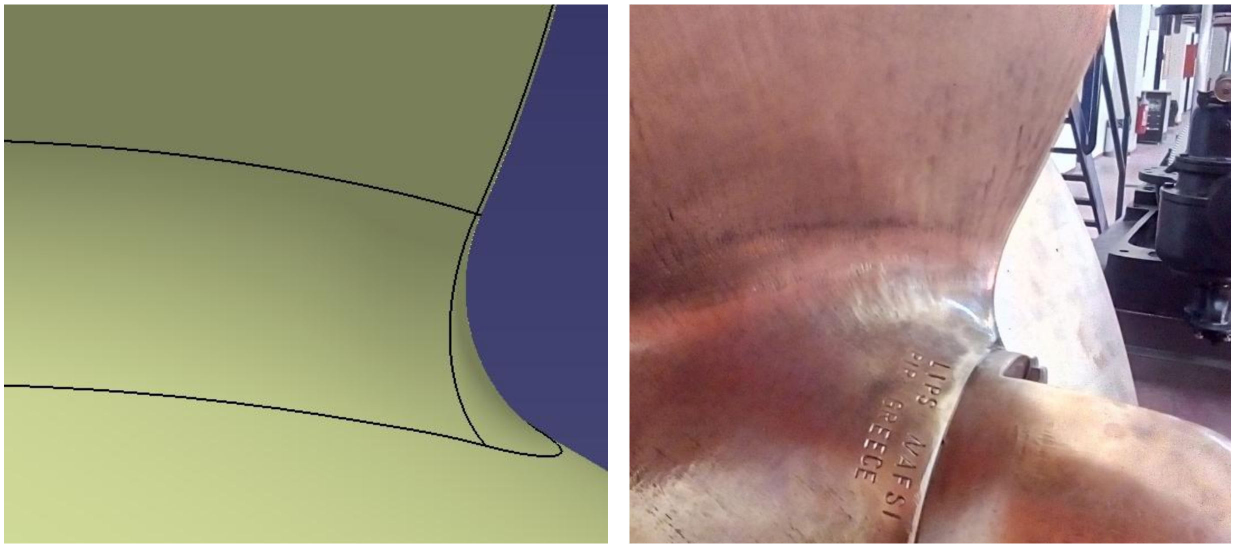


Fig. IV-1 Fillet feature in CATIA (left) and its real-life counterpart, a propeller root fillet (right)

IV-3 Solid modeling

A solid model contains all the wireframe and surface geometry necessary to fully describe the edges and faces of the model. In addition to the geometric information, solid models also convey their topology, which relates the geometry together. CATIA uses the Boundary Representation (b-rep) method to represent a model's topology.

IV-3.1 Boundary representation

In boundary representation (Stroud (2006)), the object is represented in terms of its "skin" - the boundaries between the model and non-model, which is a set of adjacent bounded surface called "faces" . The faces are bounded by sets of curves, their edges. The point where more than one edges meet is called a vertex. B-rep techniques can be used to describe both solid models and a variety of simplified forms called "degenerate models" as well as non-manifold objects.

The "manifold" condition states that at every point of the boundary of an object, a small enough sphere will cut the object boundary in a figure homeomorphic to a disc. A manifold object is essentially an object that can be charted, that can be physically realisable.

IV-3.2 Parametric model

The parametric solid model is more than a representation of a solid. It includes, a meta-structure from which specific solids can be instantiated, which means that a history of the dimensions and relations used to create a feature are stored in the model. This enables the user to capture design intent and to easily make changes to the model by modifying these parameters. So, it might be more appropriate to describe a parametric solid as a class of possible specific solid models.

C. M. Hoffman and R. Joan-Arinyo Christoph M. Hoffmann (2002) propose the following definition for parametric models:

"A parametric solid model is an information structure that permits deriving specific solid models using a deterministic algorithm. Moreover, the specific shape derived depends on parameters that are explicit in the information structure and must be valuated for obtaining a specific solid shape"

Parameters in CATIA are categorized into driving dimensions and relations:

- <u>Driving dimensions</u> are the main dimensions used when creating a feature. They include both dimensions associated with the sketch geometry, and those associated with the feature itself.
- <u>Relations</u> include information such as parallelism, tangency and concentricity.

With a combination of the aforementioned parameters, capturing of the design intent is possible.

IV-4 Design intent

Design intent is the designer's plan of how to construct the solid model in order to properly convey its required form and functionality. For a parametric modeler like CATIA to be used efficiently, the designer must consider the design intent both at the early stages and during the creation of the parametric model. The way the model is set up affects its behaviour during the modifications in the duration of its life cycle, including its flexibility to changes and its stability during updates, including the resources required to compute them. Therefore, taking the design intent under consideration is very important for efficient design and modelling.

The following factors contribute to how the design intent is captured:

- Implicit relations:

Based on the sketch geometry, implicit relations provide common geometric relationships between objects, such as tangency, parallelism, perpendicularity, horizontality and verticality

- Equations

Equations create mathematical relations between dimensions, providing an external way to influence changes.

- Additional relations

Other relations, such as concentricity, coincidence and offset, provide another way to relate model geometries to one another.

- Dimensioning

The way the sketch is dimensioned impacts its design intent. Dimensions are added in a way that reflects the user's intent of making variations of them and controlling the outcome.

IV-5 Constraints

Design intent (and the parameters and relations derived from it) is represented in a solid model using constraints. Constraints establish relationships between features in the model, by determining their relative positions to one another, providing the necessary information on dimensions, size, number of features and so on.

Some examples of constraints that frequently arise are:

- Dimensional: such as length, distance, angle.
- Geometric: such as concentricity, tangency, perpendicularity.
- Equational: expressing the relations between dimensional parameters and/or derivative variables such as an object's moment.
- Semantic: defining validity conditions on a shape
- Topological: relations between entities in a model such as connectivity and incidence.

If the imposed constraints are considered a system of functions (where the constraints themselves are either variables or constants), the parametric problem (the solid) can be solved by identifying a sequence of steps that can provide the solution to the constraint system. When the constraints representing the model's parameters are changed by the designer according to the design intent, this leads to a solution that takes under consideration the rest of the constraints that were established during the design process. If the model is well-constrained, the solution should represent the design intent. On the other hand, if the model is under-constrained or wrongly constrained, the resulting model can be very different to the one expected.

IV-5.1 Using constraints

Constraint-based modelling offers the capability of quickly creating precise drawings and models by applying specific dimensional and relational constraints on a rough sketch or solid. Since the method developed is based on models produced from sketches, the constraints as used in the sketcher are presented.

The most usual approach to using constraints, is to prepare an initial sketch with the required geometry. Then, the sketches are constrained through CATIA's menu. The sketches are converted into precise drawing by solving the constraints. Finally, a solid object or surface is generated by applying operations and features, such as extruding, revolving and cutting, padding, holes etc. The features are also constrained by defining their parameters.

In order for a sketch to be portrayed as per the designer's intent, there should be no grounds for misinterpretation of the constraints by the solver. This entails that the sketch should be, as it is called, well-constrained, meaning that no additional constraints can be added without creating redundancies. A sketch that is over-constrained (has redundant constraints) is usually rejected. On the other hand, a sketch that needs more constraints to be defined is under-constrained. The sketch may still be represented, but under-constrained elements might take values not in accordance to the designer's intent, when solving the constraints.

In CATIA, unconstrained or semi-constrained elements in a sketch are coloured white; fully-constrained elements are coloured green; over-constrained elements are coloured magenta (an example can be seen in Fig. IV-2). Apart from the colour-coded geometry, the sketch analysis tool can also provide further info on the constraint status of the sketch geometries.

When an already constrained model is edited, for example by changing some of the constraints, a new instance of that model is automatically constructed by solving the constraint problems with the changed values. Complex editing may make radical changes to the whole model, for example by automatically adding features or changing the definitions of them.

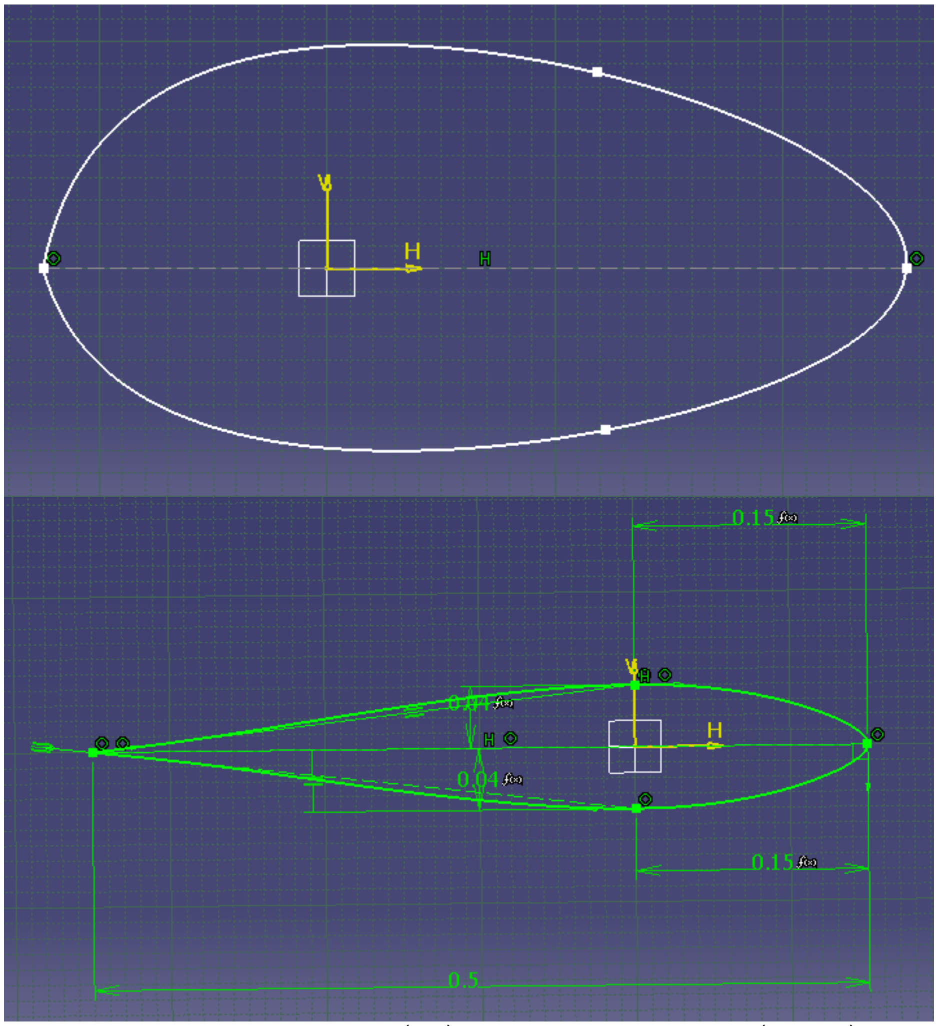


Fig. IV-2 Example of an unconstrained (top) and a fully constrained (bottom) sketch of a foil section. The original shape is created with little resemblance to the final sketch on purpose, to show the impact that constraining has on the sketch.

IV-6 Associations

A CATIA model is fully associative with the drawings and parts or assemblies that reference it. Changes to the model are automatically reflected in the associated drawings, parts and/or assemblies. Likewise, changes in the context of the drawing or assembly are reflected back in the model. In addition to that, CATIA documents can also be linked with and receive input from other types of files, such as excel worksheets, import geometry from other programs in different file formats (such as IGES, DXF, STEP, STL etc.) and receive scripts written in its scripting language that is based on Visual Basic.

IV-6.1 Managing Links

An important tool that helps manage the CATIA modularity is the document Links management. This provides a list of the currently active direct links and their status. Direct links are external documents directed pointed to by the active document. The command can be found under Edit>Links.

The Links dialog box provide link-related information, such as the pointing and pointed element, the pointed document, the link type, the status, when it was last synchronised and so on. Replacement of a document can also be performed within the dialog window, under the "Pointed documents" tab.

Replacement can also be done for any types of documents, either non-CATIA or CATIA. For example, a linked design table can be exchanged for another version of the table; or in an Analysis product, the pointed CATIA part document can be exchanged for another using the same command.

IV-6.2 The Design Table

The design table is software capability designed to serve as the interface between the parameters of a CATIA document and external values. The external values are stored in the form of a table, either in a Microsoft Excel file, or in a tabulated text file. A design table can be created from a CATIA document and then export the data to a design table. It can also be importing to a document if it is pre-existing, with its data saved into CATIA parameters.

The design table columns may not correspond completely to the CATIA document's parameters. In any case, the user needs to declare which documents parameters are linked to the required design table columns. This is achieved by creating associations between the design table cells and CATIA parameters.

For ease of use, the design tables used for this thesis are saved in an Excel file and this is what will be described below.

<u>Excel sheet format</u>

The values in the sheet cells need to be expressed in the appropriate units. If no unit is mentioned, then the unit is either taken into account as mentioned in the first row, or as the relevant SI unit of the parameter, if no unit is mentioned.

The table should be saved in the Excel 97-2003 version (.xls extension) and not the more recent one (.xlsx), which is not supported.

Design table creation

No matter where the existing data is saved, the design table must be created in CATIA. The two ways to do that are using current parameter values or using a pre-existing file.

In the first case, a new file to save the values is created. The user names the table, selects its orientation (horizontal or vertical), the sheet index in case of an Excel file and the tree destination of the table. Then, the parameters to be inserted in the table are chosen and last, a new file is created.

In the second case, the user names the table, selects the orientation of the existing table, the sheet index where the data can be found (in case of an Excel file) and the tree destination of the table. The existing file is then selected and a choice is offered of automatically creating associations between existing parameters and table columns that have the same name. The associations can also be manually defined under the associations tab. Existing parameters can be selected to be associated with a table column. New parameters can also be created in case they are needed. It is not required to associate all of the tables columns with a parameter. The current design table, as existing in CATIA, is presented under the configurations tab.

Value update

The design table is automatically updated (synchronized) every time the external file is altered and saved. The associated parameters take on the modified values and, depending on the settings, the solid model is updated accordingly

IV-7 Surface analysis

CATIA includes tools to analyse the created geometry. One of these is the surface curvature analysis tool, included in the freestyle shaper workbench. The tool provides a quick visual representation of the changes in surfacic curvature, colouring areas with different curvature according to a user-specified colour chart. Options on the type of analysis are offered (Gaussian, mean, maximum and minimum curvature), as well as the option to display values on a spot using the cursor. An example of the surfacic curvature analysis on the third-party propeller geometry that was meshed is presented in Fig. IV-3

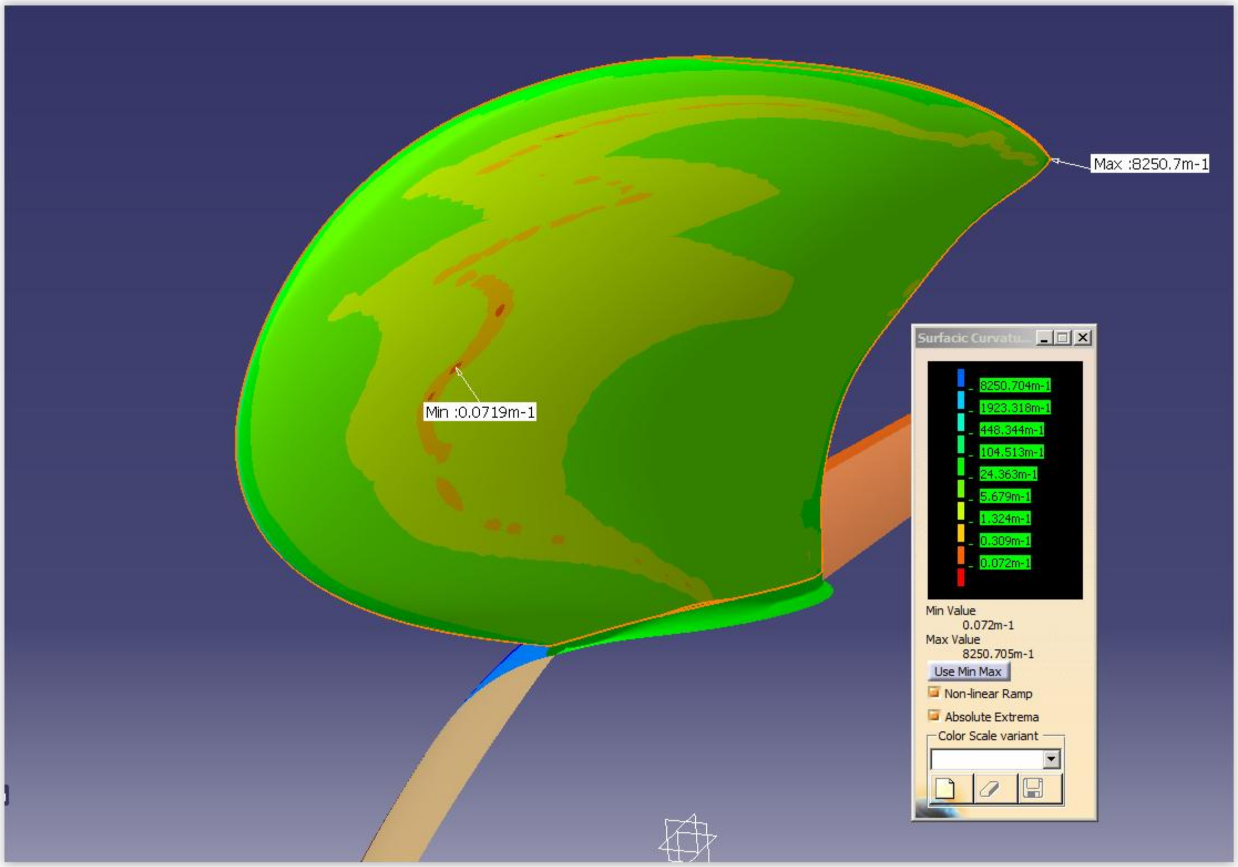


Fig. IV-3 Surface curvature analysis of the SMP propeller using the Gaussian type. The colour legend is also presented: Red(minimum): 0.072m-1, Green:24.363m-1, Blue(maximum): 8250.7 m-1

Part V. Meshing in CATIA

The CATIA mesher can be found in the Advanced Meshing Tools workbench of the Analysis product. It allows the rapid creation of a finite element model for complex parts, whether they are surface or solid. It will generate an associative mesh from complex parts, with advanced control on mesh specifications. The mesher contains two functionalities:

- The FEM Surface Mesher (FMS), used to generate a finite element model from surface parts
- The FEM Solid Mesher (FMD), to generate a finite element model from solid parts.

Since the aim is the creation of surface meshes, the Surface Mesher is used and it is what will be described in this chapter. Further details on CATIA's mesher can be found in the CATIA User's manual – Advanced meshing tools. A step-by-step example for the creation of meshes can be found in the Appendix.

There are two surface meshing capabilities: the simple surface mesher and the advanced surface mesher, offering similar functionalities. The simple surface mesher was chosen due to its being simpler in use and modification.

The basis of the mesher operation is the following:

1. Pick a mesh support surface
2. Define Global meshing parameters
3. Define meshing specifications
4. Execute meshing

The surface support to be meshed is picked upon entering the mesher. The mesh can be considered a feature that is applied to the specific surface by defining a domain. Each support may have more than one domain and subsequently have more than one sub-meshes that together make up the final mesh. However, for the current method, it is strictly required that each support is meshed using only one domain. For this, support is specified during the model creation phase, by splitting the model into the necessary surfaces, so no other action needs to be taken within the mesher workbench.

V-1 Global meshing parameters

The Global parameters are defined upon support selection, through a window. They include the Mesh parameters and the Geometry parameters, and pertain to the supporting surface meshing parameters.

V-1.1 Mesh parameters

For the Mesh parameters, the following can be defined

- the shape of mesh elements (triangular or rectangular)
- their type (linear or parabolic)

- The mesh size
- What the default meshing method will be
- Whether the mesh distribution will be automatically captured by neighbouring meshes or explicitly given.

For the current method, the type of the elements is very important. The mesh translation program is written with the assumption that the meshes exported have linear elements. Parabolic elements include extra information that is not necessary for the current intended use of the mesh, whereas both types result in the same mesh. So, it is imperative that all meshes are created and exported with linear elements.

The mesh size is generally used for automatic meshing. It is a guideline for the mesher for the final size of the meshes. However, it also plays a part in the mesh update time. A small mesh size will make updating slower, whether the mesh is automatic or with imposed specifications. It has also been noticed that in case of complex geometry, setting a smaller element size leads to the generated mesh following the geometry better and being smoother. So, a general rule would be to use larger element sizes in simpler geometries that don't have a lot of detail and use smaller element sizes in more complex geometries, so that a balance between meshing time and geometry fidelity is achieved.

The meshing method choice is irrelevant, since the mesh type will be explicitly imposed later on. However, the automatic mesh capture option is best left unused, since it sometimes creates problems with the mesh update.

Geometry parameters

The geometry parameters tab defines constrains regarding the surface geometry. Of all the options given, constraint sag is what plays a part in the current meshing procedure:

Constraint sag

A constraint is created along the edge of a face to avoid creating elements across this edge (the element sag would be higher than the specified value). This does not guarantee that the whole mesh respects the sag value but helps with constraints creation. For a given mesh size, the lower the constraint sag value, the more numerous the created constraints, and vice versa.

The constraint sag can be controlled either by input of the value in the box, or using a slider below it. The requirement is to avoid having inner edge constraints, so the constraint sag slider is set to 0. Depending on element sizes, the constraint sag box above it gets a different value, which should be higher than the element size in case inner constraints are undesired.

The other options in the geometry parameters tab should be left unchecked, since automatic capturing of the internal geometry must be avoided.

V-2 Meshing specifications

The local meshing specifications can be defined using tools inside the mesher workbench. The tools deal with the meshing parameters for the domain(s) of which the support consists.

The meshing specifications are divided in the following three toolbars, according to their use:

-Geometrical Specifications

- 1D Mesh Specifications

- 2D Mesh Specifications

V-2.1 Geometrical Specifications toolbar

Hole suppression

The suppressing holes toolbar enables the user to define holes in the geometry to be ignored. The mesher will deal with the suppressed holes as if they're non-existent, creating a mesh over them.

Edge Constrain

A surface's outer edges are always constrained (they are marked in a green colour). However, sub-domains can be defined within the supporting surface, by specifying inner edges to be constrained or unconstrained. A constrained inner edge takes on a yellow colour, whereas an edge specified as unconstrained is blue. In order for an edge to be constrained, it must be a b-rep geometry. This practically means that every edge that appears in any "with edges" rendering option can be constrained.

The edge constraint tool creates similar constraints as the automatic global parameters one. Their main difference is that the global setting may constrain more or less of the inner geometry than needed, whereas the edge constrain tool creates constraints in a more specific way. In addition, the tool can override the global parameters, turning an edge that is constrained due to the global rule into an unconstrained one.

It is reminded, that for the developed method, creating more than one domain must be avoided.

Vertex Constrain

Same as with the edges, vertices on edges can be constrained. Defining a vertex as constrained tells the mesher where the edge's boundaries lie. The mesher deals with any vertex where there is an edge junction as a constrained one by default. This means that in order to remove a vertex constrain, there have to be only two constrain edges that touch that vertex – a vertex cannot be removed when there are more edges passing through it. However, a constrained edge can be split, by adding a vertex constrain without adding an extra constrained edge. A constrained vertex has red colour and a constrained vertex that splits an edge is yellow, whereas an unconstrained vertex is blue.

External curves and points projection

Edges and vertices can also be created by projecting external curves or points. These projections can then be constrained in a similar way as the b-rep ones, using the constrain tools.

V-2.2 1D Mesh Specifications toolbar

Elements Distribution

Elements distribution is defined by imposing nodes on a supporting edge. To define an element distribution, the supporting edge is picked, then the type of distribution and, depending on the distribution type, a set of parameters to define the number of elements, size, distance between them etc.

The types of elements distribution that the mesher offers is the following:

- Uniform: The distance between all distributed nodes is the same. The offered options are either number of elements or element size.
- Arithmetic: the distance between the distributed nodes will be defined by a common difference computed with two of the following parameters:
  - Number of elements
  - Difference between first and last distribution edge (Size 2/ Size 1)
  - Length of first element in distribution
  - Length of last element in distribution
- Geometric: same as arithmetic, however the distance between distributed nodes is defined by a common ratio. Parameters are similar to the arithmetic distribution. The only difference is that the Size 2/ Size1 option calculates a ratio instead of a difference. Both this and the Arithmetic distribution offer the option to create a symmetric distribution
- User Law: The distance between the nodes is defined by a knowledge law created by the user.

The type of distribution used in the current meshing method is either the geometric or the arithmetic one, since they offer the option to decrease nodes distance in place where required. In addition, the only used defining parameters are the number of elements and the size 2/size1 ratio/difference. For the uniform meshes, the ratio is assigned a value of 1

The element distribution can be modified using parameters. More specifically, the number of elements, mesh size and ratios can receive values from formulas. This is a main component in the way the proposed method updates the meshes, since it enables the use of Excel to facilitate modifying the elements distribution.

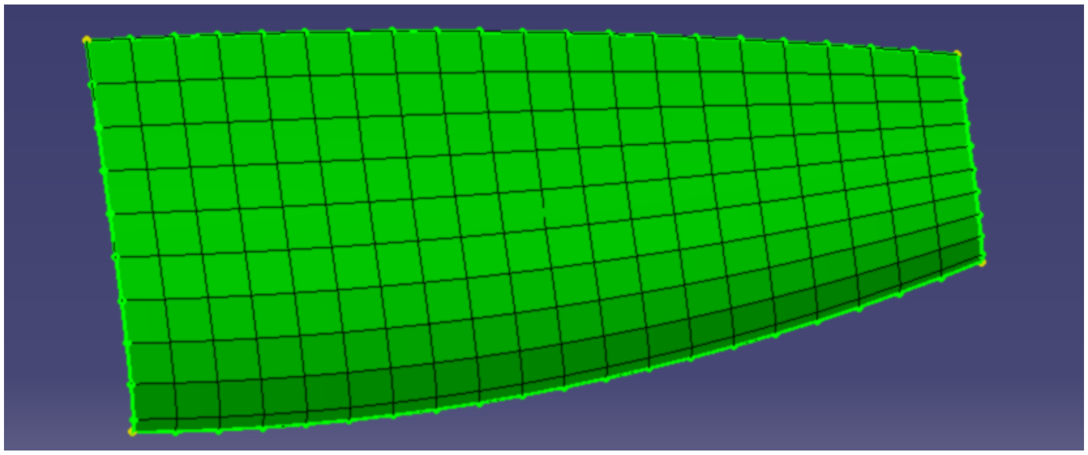


Fig. V-1 Uniform elements distribution mesh

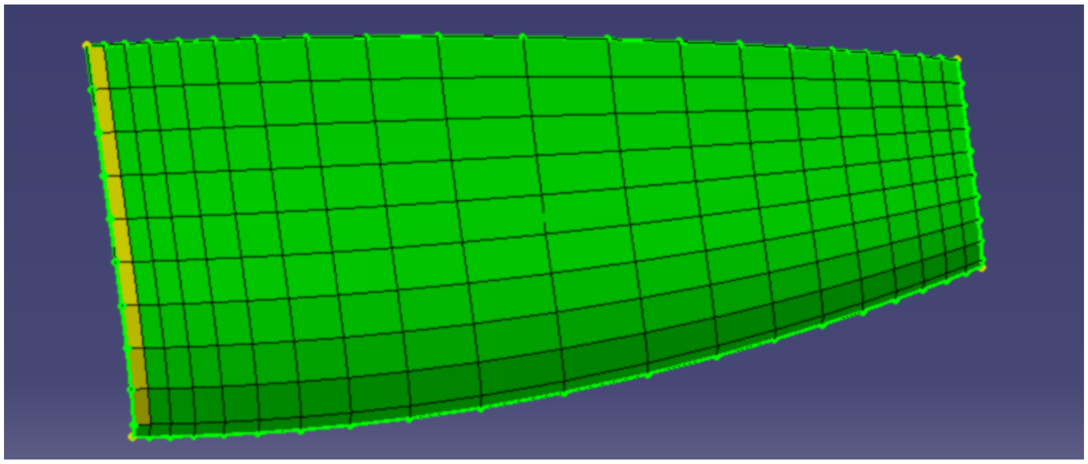


Fig. V-2 Geometric elements distribution, denser towards right edge

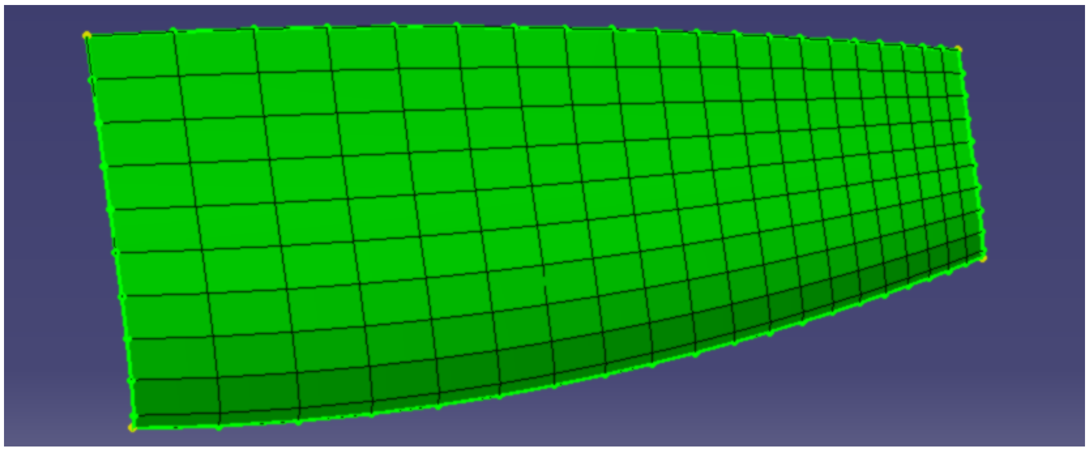


Fig. V-3 Geometric elements distribution, symmetric with denser distribution on edges

<u>Elements Capture</u>

Apart from explicitly defining the elements distribution, the mesher enables the capturing of a mesh distribution from the edge of an adjacent mesh. The mesh edges do not need to be exactly matched, since the nodes source can be automatically found within a distance tolerance that the user defines. Apart from the source edge selection (from which the distribution will be derived), two extra nodes options are offered:

Coincidence: if checked, the nodes will be superimposed on the common edge. This leads to pairs of nodes with the same coordinates.

Condensation: the coinciding nodes for either mesh are “merged”. There is only one node per set of coordinates and it is shared by both meshes.

For the proposed meshing method, elements capture is advised against, since it is frequently problematic with the model updating. However, in case it is used, it must be used with only the coincidence option checked so that individual nodes are created for each mesh.

<u>Elements distribution around hole</u>

As its name suggests, this tool enables the user to define the elements distribution around a hole.

The structured mesh domain is not compatible with meshing around holes, so this is not a tool used in the current method.

<u>Distribution propagation</u>

With the distribution propagation tool, a node distribution of an edge (source) can be reproduced on another edge (support) of the same domain. The reproduction can be:

- Proportional: the relative distance between nodes is preserved
- Normal: the source-target vector direction is normal to the supporting edge
- Directional: the source-target vector is parallel to a given direction

Distribution propagation is not a tool preferred, mostly because by explicitly defining the nodes distribution for each edge using the elements distribution tool, the chance for errors is minimized.

V-2.3 2D Mesh Specifications toolbar

The 2D mesh specifications toolbar contains tools to define the type of mesh a domain will receive. In general, the mesh defined by selecting the required method and then the supporting domain. All of the meshing methods have an “impact neighbour domains” option, which re-meshes the neighbouring meshes to match the edge nodes of current domain’s mesh. One thing to keep in mind is that in order to be available for selection, the domain geometry must be visible, not hidden.

Mapped Mesh Method

The Mapped Mesh Method creates a structured mesh, which is the one required (Fig. V-4). To create a structured mesh per the current requirements, the domain must always be meshed with the "Split quadrangles" option unchecked, so that no triangular meshes are created such as in Fig. V-5. In addition, the impact neighbour domains, as well as the mesh size options have no effect in a fully constrained mesh.

Other meshing methods

No other meshing methods are used in the current thesis, since they don't produce a structured mesh. Examples of each method can be seen below (Fig. V-6 - Fig. V-11).

V-1 Exporting the mesh

After the required meshes have been created, they need to be exported. The export can be done either in a bulk data file with the .dat file extension or in a CATIA V4 file with the .model file extension. For the current requirements, the bulk data file type is selected.

Every active mesh will be exported, and in the order that it appears in the specification tree. Thus, every mesh that is undesired for a particular export, must be deactivated via the specification tree. In addition, the meshes can be rearranged inside the specification tree, according to the user's needs.

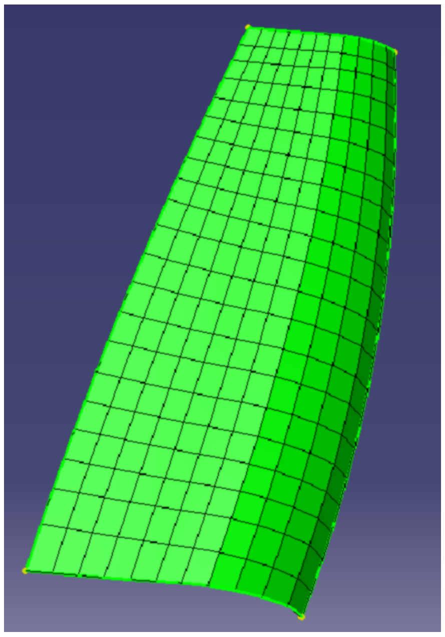


Fig. V-4 Mapped Method - Split quadrangles unselected

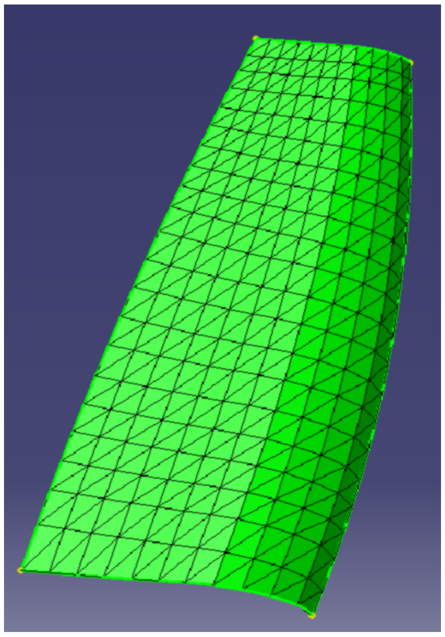


Fig. V-5 Mapped Method - Split quadrangles selected

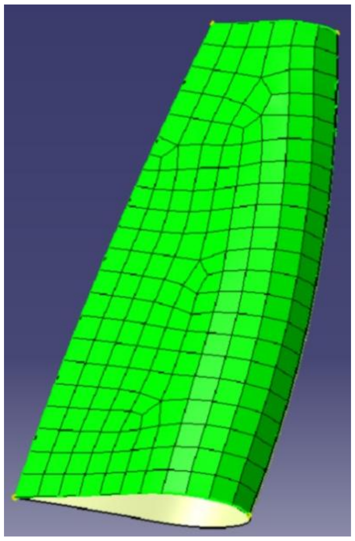


Fig. V-6 Frontal Quadrangle Method

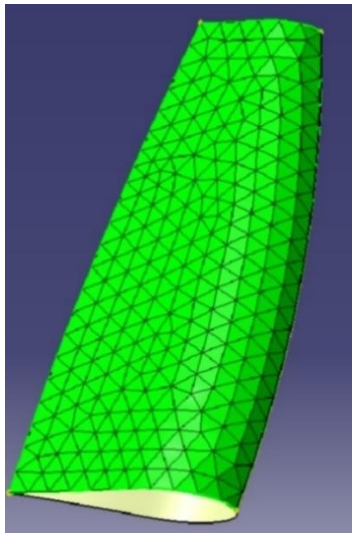


Fig. V-7 Frontal Triangle Method

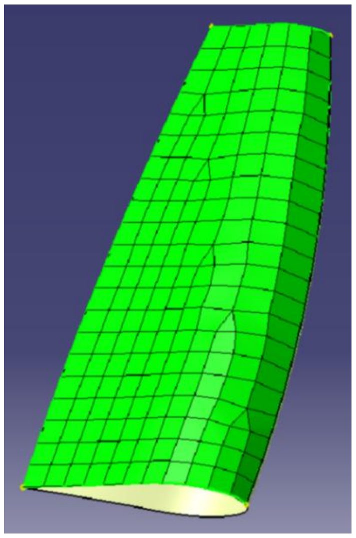


Fig. V-8 Mapped Free Method

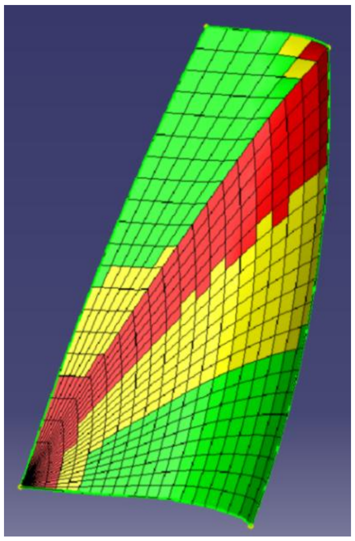


Fig. V-9 Triangular Domain Mapped Method

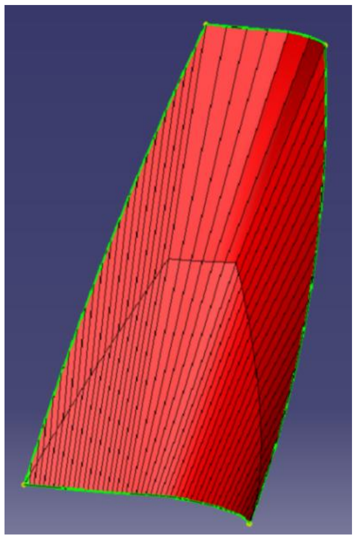


Fig. V-10 Bead Method

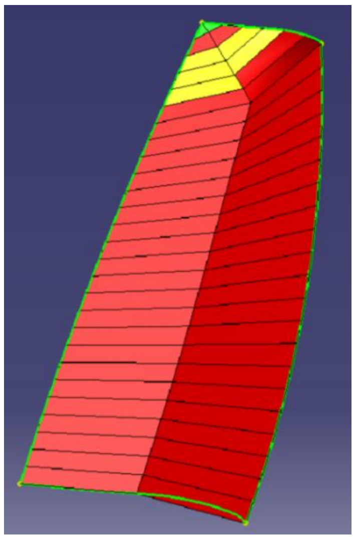


Fig. V-11 Half Bead Method

V-2 Mesh quality analysis

The CATIA mesher offers a set of tools to analyze the quality of the elements in the mesh, both visually and in report form.

The "Quality Analysis" tool in the Mesh Analysis workbench contains a set of criteria for element analysis, such as skewness (and skew angle), warp factor, min. and max. angle of the quadrilateral elements, aspect ratio etc. Each analysis criterion can toggled on or off and receive three threshold values that define the three categories (best value, poor and bad – see Fig. V-13). Using the criteria, an analysis report can be produced and saved, showing the percentage of elements contained in each of the quality categories. This report can also be presented in the form of a bar diagram.

The visual analysis is carried out using the Quality Visualisation option in the Mesh Visualisation toolbar. The elements are colour-coded according to the thresholds set in the Quality Analysis tool and are presented in Table III-1

Table V-1 Quality analysis colour codes

Green	Best value
Yellow	Poor
Bad	Red

An example of a visual analysis based on the elements aspect ratio can be seen in Fig. V-12

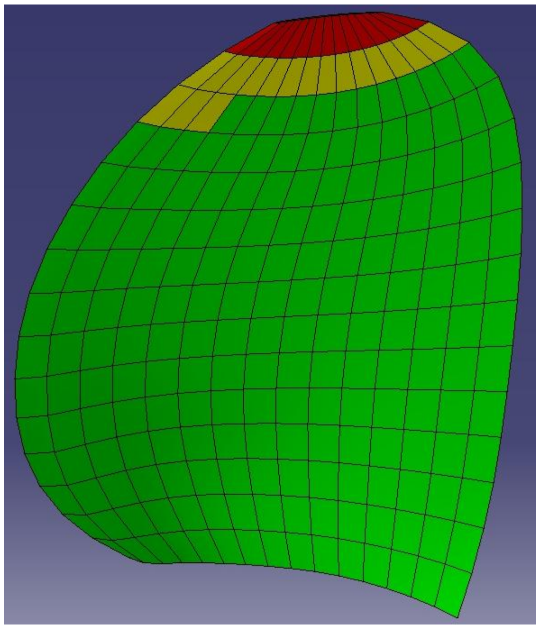
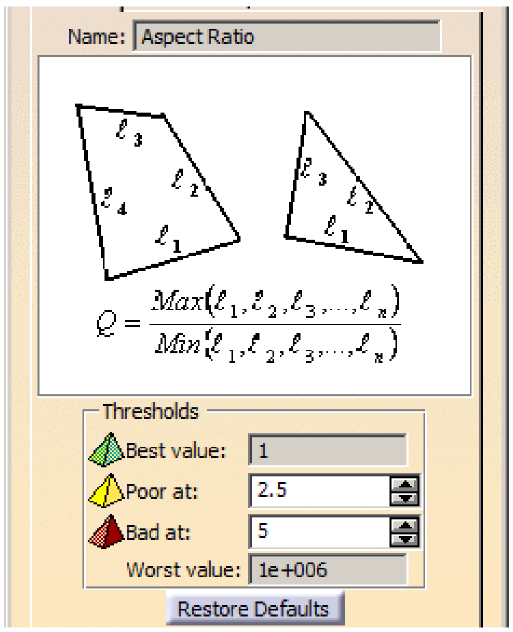


Fig. V-12 1 Quality analysis of a B4-60-D5P1 blade, 15X15 uniform mesh, using the aspect ratio criterion (green: best at 1, yellow: poor at 2.5, red: bad at 5)



Name: Aspect Ratio

$$Q = \frac{Max(\ell_1,\ell_2,\ell_3,\ldots,\ell_n)}{Min(\ell_1,\ell_2,\ell_3,\ldots,\ell_n)}$$

Thresholds

Best value: 1

Poor at: 2.5

Bad at: 5

Worst value: 1e+006

Fig. V-13 Aspect ratio definition and thresholds as defined in the Quality Analysis tool

Part VI. Reformatting the meshes for use in UBEM

The exported meshes from CATIA contain the necessary information, but not in the correct format for input in UBEM. For this reason, a separate interface program was written in FORTRAN, which translates the exported mesh into the format needed.

VI-1 CATIA's exported format

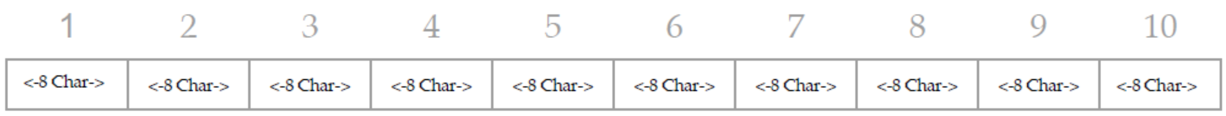
CATIA exports the mesh in a Bulk Data file to be used by software NASTRAN. The nodes are presented in a global numbering with coordinates scheme and the elements in relation to the nodes. The data is presented in two kinds of formats, the small field and the large field format.

VI-1.1 Small field format

With the small field format, the software separates an 80 column line of Bulk Data entry into ten equal fields of eight characters each (Fig. VI-1):

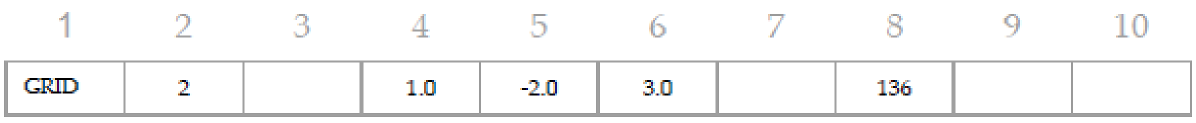
The small field format rules are as following:

- Fields 1 and 10 must be left-justified.
- Fields 2 through 9 don't need to be either right- or left-justified. However, aligning the data fields is a good practice.
- Embedded blanks can't be included in a character field



1	2	3	4	5	6	7	8	9	10
<-8 Char->	<-8 Char->	<-8 Char->	<-8 Char->	<-8 Char->	<-8 Char->	<-8 Char->	<-8 Char->	<-8 Char->	<-8 Char->

Fig. VI-1 Small field format (Source: NX Nastran - User's Guide)



1	2	3	4	5	6	7	8	9	10
GRID	2		1.0	-2.0	3.0		136		

Fig. VI-2 Example of small field format for use in NASTRAN (Source: NX Nastran - User's Guide)

VI-1.2 Large field format

A high degree of numerical accuracy is required in some applications. The large field format can be used when the small field format doesn't provide enough significant digits. Note: A minus sign, decimal point, and the "E" in scientific notation count as characters.

The large field format rules are as following:

The large field format requires (at least) two lines for each entry:

- the first and last field of each line contains eight columns
- the fields in between contain 16 columns

The large field entries are denoted by an asterisk (*). The asterisk must:

- immediately follow the character string in field 1A of the first line of the entry
- immediately precede the character string in field 1B of the second line of the entry

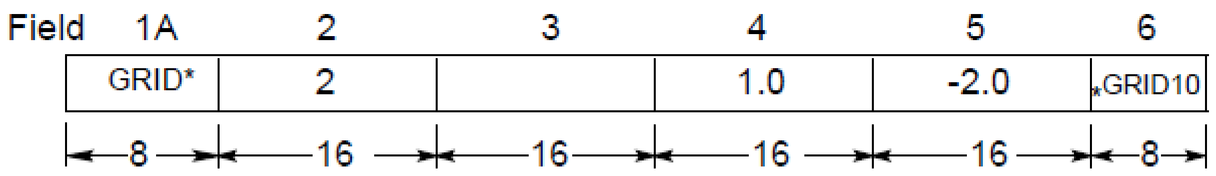


Fig. VI-3 Example of large field format, showing the number of characters in each field (Source: NX Nastran - User's Guide)

VI-1.3 The exported Data

The data in the export file is contained in two groups: the Nodes group and the Elements group. The nodes receive global numbering and their coordinates; the elements are separated by surface, with the numbering carrying on from the group of one surface to the elements of the next one.

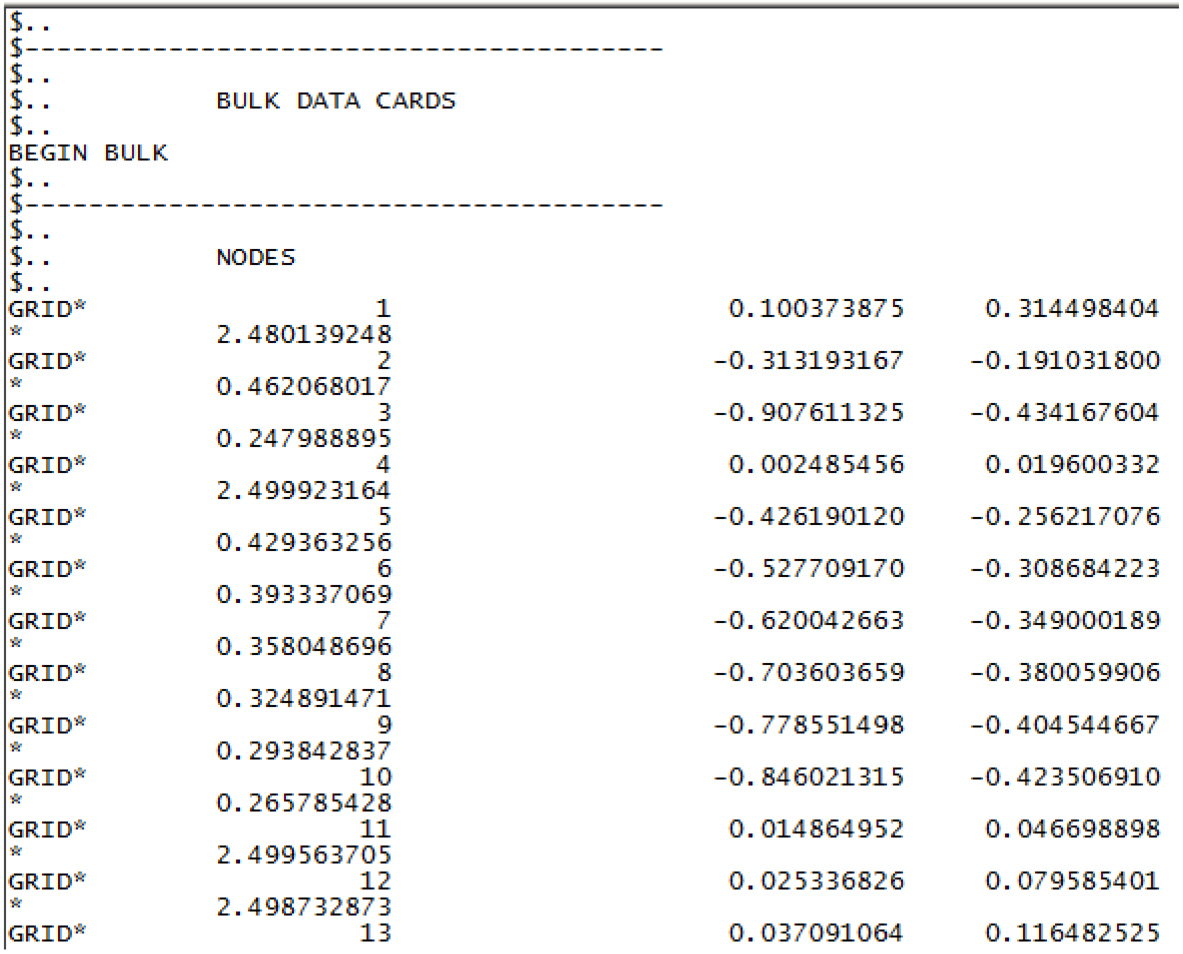
<u>**The Nodes**</u>

The Nodes group contains the information for all of the meshes nodes, using the large field format, as follows (Table VI-1):

Table VI-1– Nodes export format (large field) in CATIA

GRID*		Node number	X coordinate	Y coordinate	
*	Z coordinate				

As it can be seen in Fig. VI-4, each node is assigned a number (in a way that will be described further down), and is followed by its X, Y and Z coordinates.



```
$..
$---------------------------------------
$..
$..         BULK DATA CARDS
$..
BEGIN BULK
$..
$---------------------------------------
$..
$..         NODES
$..
GRID*                      1                   0.100373875     0.314498404
*           2.480139248
GRID*                      2                  -0.313193167    -0.191031800
*           0.462068017
GRID*                      3                  -0.907611325    -0.434167604
*           0.247988895
GRID*                      4                   0.002485456     0.019600332
*           2.499923164
GRID*                      5                  -0.426190120    -0.256217076
*           0.429363256
GRID*                      6                  -0.527709170    -0.308684223
*           0.393337069
GRID*                      7                  -0.620042663    -0.349000189
*           0.358048696
GRID*                      8                  -0.703603659    -0.380059906
*           0.324891471
GRID*                      9                  -0.778551498    -0.404544667
*           0.293842837
GRID*                     10                  -0.846021315    -0.423506910
*           0.265785428
GRID*                     11                   0.014864952     0.046698898
*           2.499563705
GRID*                     12                   0.025336826     0.079585401
*           2.498732873
GRID*                     13                   0.037091064     0.116482525
```

Fig. VI-4Nodes data in CATIA bulk file. For each node its number and X,Y and Z coordinates are presented in two lines.

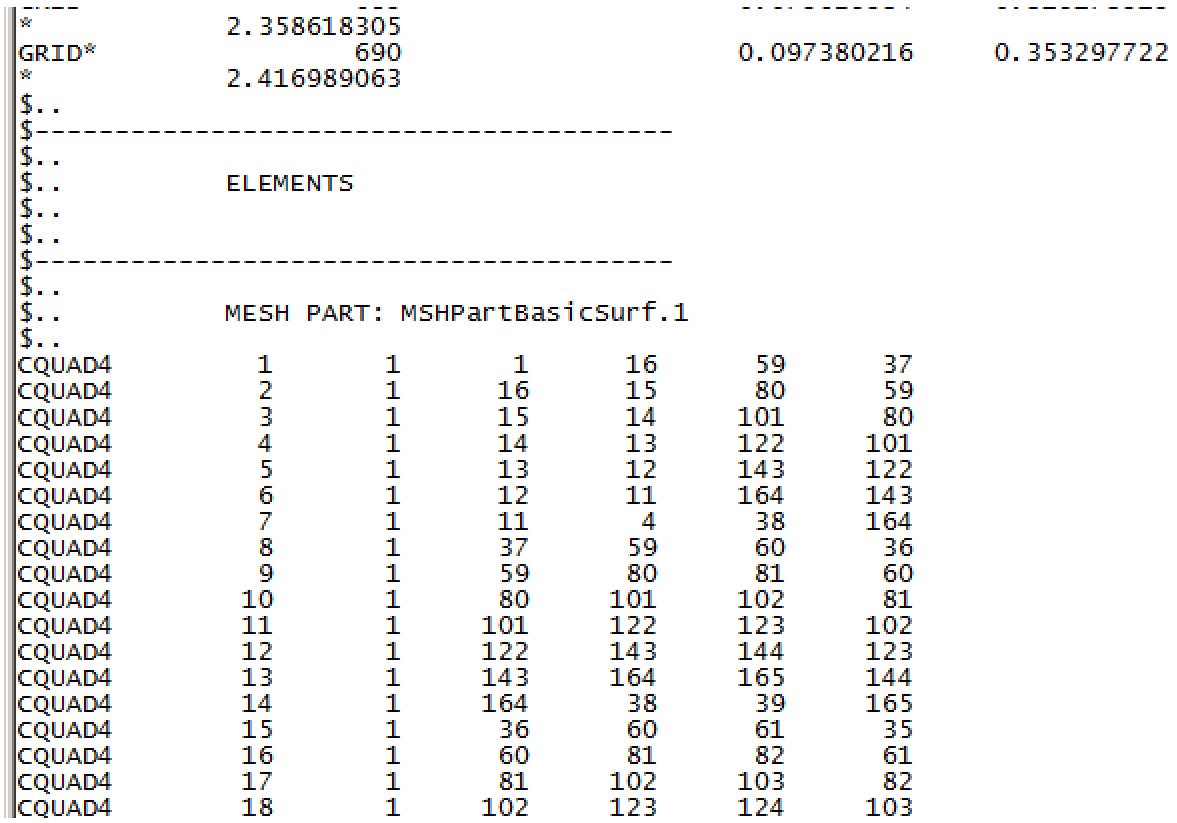
<u>The Elements</u>

The Elements group is presented by surface. For each surface, the Elements are written in the small field format. For linear elements, the format is as presented in Table VI-2.

Table VI-2 Linear quadratic element export format (small field) in CATIA

CQUAD4	Element number	Srfc number	Node A	nodeB	nodeC	nodeD

The elements are presented in relation to their nodes; for each element, its index and the four nodes that define it are given. A screenshot of a bulk data file can be seen in Fig. VI-5



```
*          2.358618305
GRID*              690                     0.097380216     0.353297722
*          2.416989063
$..
$---------------------------------------
$..
$..        ELEMENTS
$..
$..
$---------------------------------------
$..
$..        MESH PART: MSHPartBasicSurf.1
$..
CQUAD4            1        1        1       16       59       37
CQUAD4            2        1       16       15       80       59
CQUAD4            3        1       15       14      101       80
CQUAD4            4        1       14       13      122      101
CQUAD4            5        1       13       12      143      122
CQUAD4            6        1       12       11      164      143
CQUAD4            7        1       11        4       38      164
CQUAD4            8        1       37       59       60       36
CQUAD4            9        1       59       80       81       60
CQUAD4           10        1       80      101      102       81
CQUAD4           11        1      101      122      123      102
CQUAD4           12        1      122      143      144      123
CQUAD4           13        1      143      164      165      144
CQUAD4           14        1      164       38       39      165
CQUAD4           15        1       36       60       61       35
CQUAD4           16        1       60       81       82       61
CQUAD4           17        1       81      102      103       82
CQUAD4           18        1      102      123      124      103
```

Fig. VI-5 Linear elements data example – CQUAD4 denotes a linear quadrangle element, Second column is the element number, third is the surface number and last four columns are the nodes that define the element

VI-1.4 Matrix arrangement

Each element consists of 4 nodes (here designated as nodeA, nodeB, nodeC and node D), which are printed in the file as described above and seen in Fig. VI-5. The nodes go clockwise around each element, with nodeA of the first element being the node not shared by any other of the patch's elements. The rest of the elements follow the same arrangement. This is true regardless of the patch's orientation.

CATIA indexes each patch of the mesh in a different direction, without allowing manual specification. In some patches the elements are indexed in a chordwise manner (chordwise oriented) and in others in a spanwise manner (spanwise oriented). Also, the orientation might differ even in adjacent patches. The orientation can be determined from the CATIA model after the mesh creation and is important for the proper printing of the final translated mesh.

VI-2 The interface program

The interface program's inputs are: the CATIA exported mesh bulk data file and an input.txt file. The input.txt file contains the following information:

- filename of input mesh bulk data file
- number of patches constituting the mesh
- total number of mesh elements
- type of surface: face-back, modified face-back (for the hub) and ordinary
- output print direction (spanwise, chordwise or hub modified)
- output filename
- 3 orientation markers for each patch and the patch's number of elements (chordwise, spanwise)

VI-2.1 Types of surfaces

The program handles two types of surfaces: the face-back type and the modified face-back.

An example of face-back type of surface (Fig. VI-6) is the propeller blade. The surface patches are indexed in the chordwise direction (they share a spanwise edge). The surface is first defined by the Face edge, starting with a direction from the LE to the TE, followed by the Back edge, also with a LE-to-TE direction.

The modified face-back (Fig. VI-7) is a surface where the patches are divided in two groups being indexed in one main direction, having a common edge on the other direction (for example, when elements are indexed in a chord-wise manner, the surfaces share a spanwise edge). However, the groups between them also share a main direction edge (in a chordwise direction, they share a chordwise edge). The main shared edge will be handled as the equivalent of the Face-Back's Leading Edge when reformatting. This type of surface is used for the propeller hub.

VI-2.2 Orientation markers

When the mesh is exported, the elements are frequently indexed in an order other than required. In order to allow the user to easily change the indexing orientation of a surface patch, the orientation markers feature was added to the interface program, and works as follows:

- A value of 0 keeps the corresponding direction as is, while 1 reverses it
- The first marker controls the patch's chordwise direction
- The second marker controls the patch's spanwise direction
- The third marker controls the patch's orientation (inward- outward)
- To assign the values, compare the patch in CATIA with the required final mesh

A set of markers for each patch is located at the end of the interface program's input file.

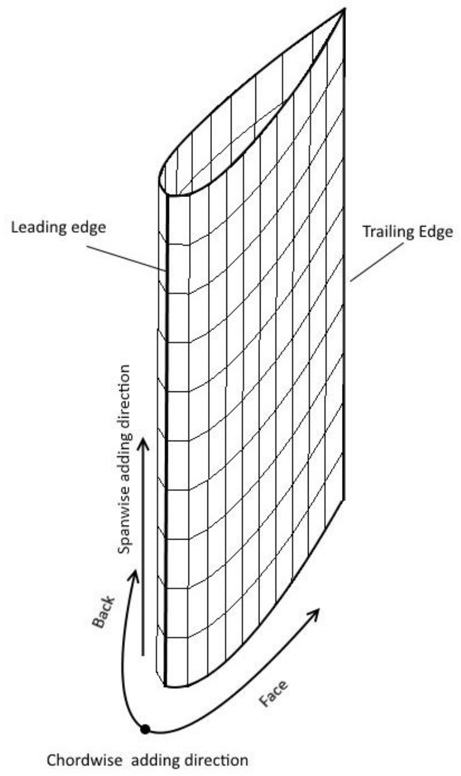


Fig. VI-6 Face-Back surface type example

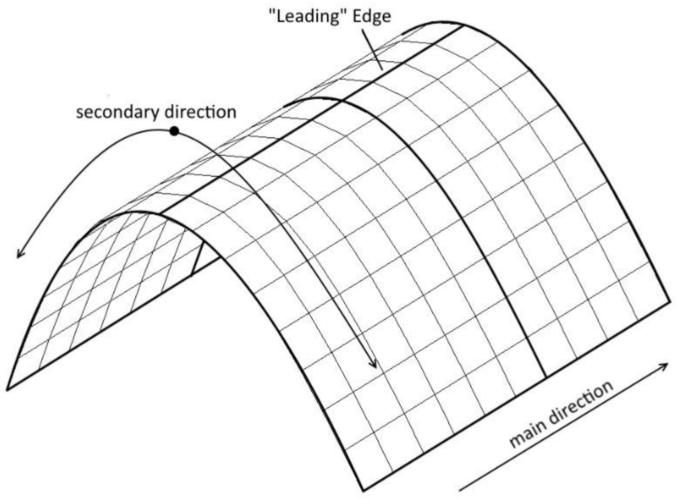


Fig. VI-7 Modified Face-Back surface type example

VI-2.3 Program

In a nutshell, the program works as following:

[1] Reads input file of surface

[2] Reads through mesh bulk data file and indexes the data location inside the file

[3] Checks if number of nodes in bulk data file is the same as number in input file

[4]Re-reads mesh file, saving the data in arrays, one for the elements and one for the nodes

[5] Constructs a temporary array (patches not properly oriented) using the information saved in the nodes and elements array

[6] Constructs the final mesh array by reading from temporary array in proper orientation

[7] Prints final mesh array in direction of choice

The interface program's source code with comments can be found in Appendix G.

Part VII. Application: A Wageningen B-Series Propeller

VII-1 The propeller geometry

The description of the propeller geometry and the one used for the development of the model uses the following reference frame: a rectangular, Cartesian system, with the X-axis coincident with the propeller shaft axis, aftwards positive; the Y-axis positive to starboard; and the Z-axis positive in the vertical upwards direction.

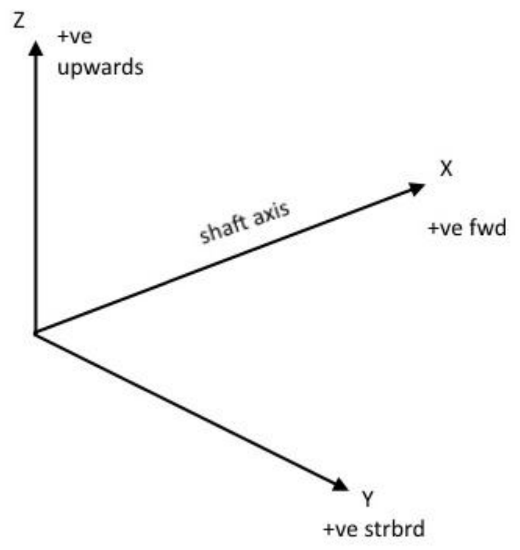


Fig. VII-1Propeller reference system. Source: (Carlton 2012)

VII-1.1 Reference lines

The propeller blade is defined about the *propeller reference line* or *directrix*, a line normal to the shaft axis, which is frequently defined starting at the origin of the Cartesian reference system (Carlton (2012)

The blade of a propeller is comprised of hydrofoil sections defined on the surfaces of cylinders whose axes are concentric with the shaft axis. Due to their pitch, the sections lie obliquely on the cylinder surface, so their nose-tail lines form helixes. The points where those helixes intersect the plane defined by the directrix and the X-axis form the propeller's *generator line*.

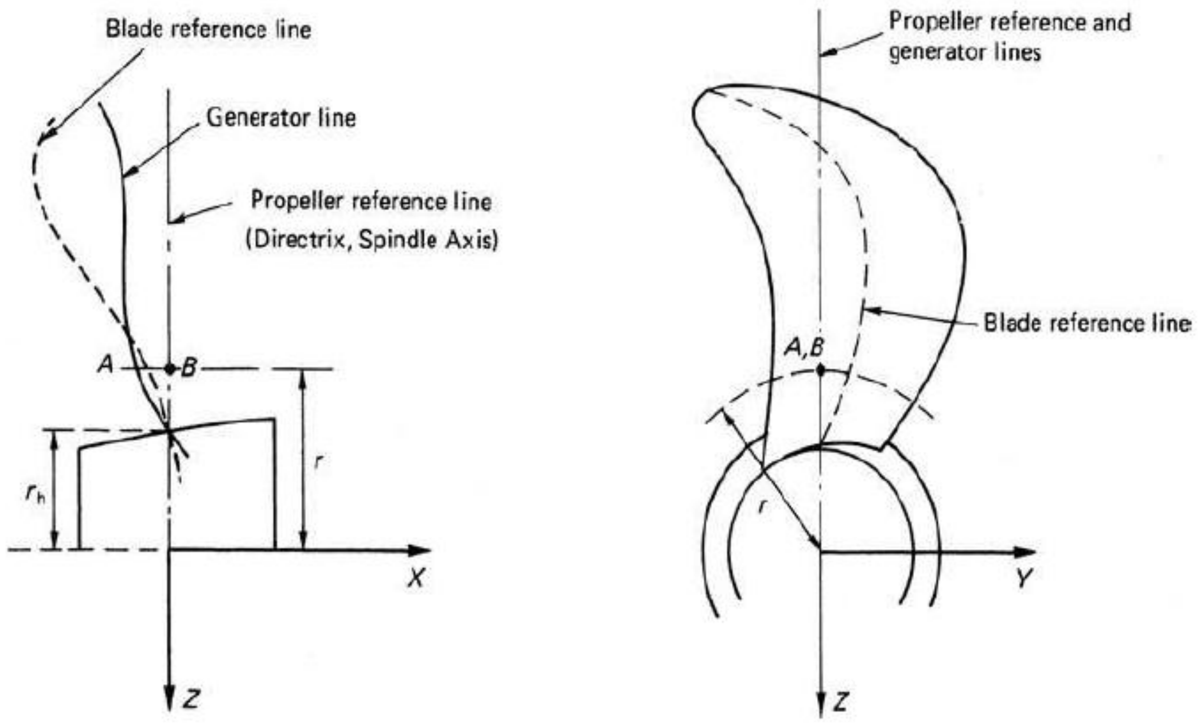


Fig. VII-2 Blade reference lines (Carlton 2012)

The propeller geometry is a complex one. However, it can be (and traditionally is) broken into simpler geometries which are easily defined and which, combined, will result in the required propeller geometry. These are: the hydrofoil section and the helical line.

VII-1.2 The hydrofoil Section

The general definition of the hydrofoil is presented in Fig. VII-3. The camber line is the locus of the mid-points between the face and back surface. The extremities of the camber line are the leading and trailing edge and the straight line joining them is the chord line. The length of the chord line is termed chord length (c) of the section. The camber of the section is the maximum distance between the mean camber line and the chord line. The hydrofoil thickness is the maximum distance between the face and back edge, usually measured perpendicularly to the chord line. The leading edge is usually a circle about a point on the camber line.

In most cases, the sections of a propeller are described by their interpolating points, which are given in the form of their offsets from the section's reference line. Chordwise on the hydrofoil, the offsets are measured either from one of the two reference line extremities, or from another reference point (for example the chord's midpoint). Then, each point's other spanwise offset is defined by its distance from the reference line. It's become common practice to use the section's chord or nose-tail line as a reference line. However, this is not always the case, especially for older propeller series, such as the Wageningen B-series; a line tangent to the pressure face (face pitch line), with length same as the chord, is used as the section's reference line (Fig. VII-4).

The point where the generator line intersects the section reference line is also an important parameter, since it defines the location of each section.

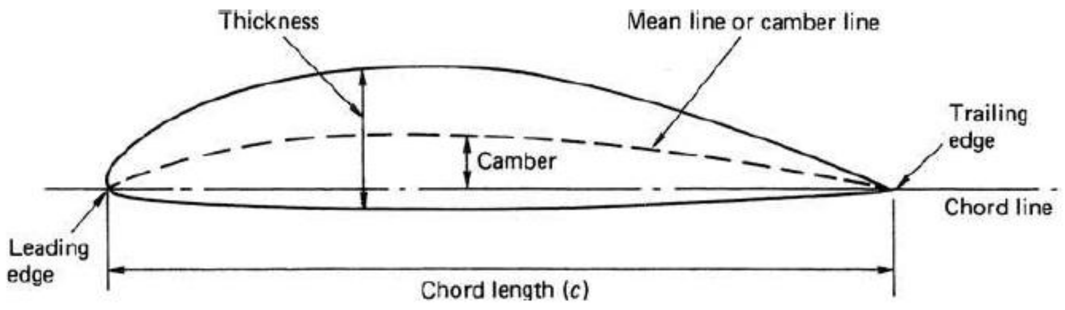


Fig. VII-3General definition of a hydrofoil section

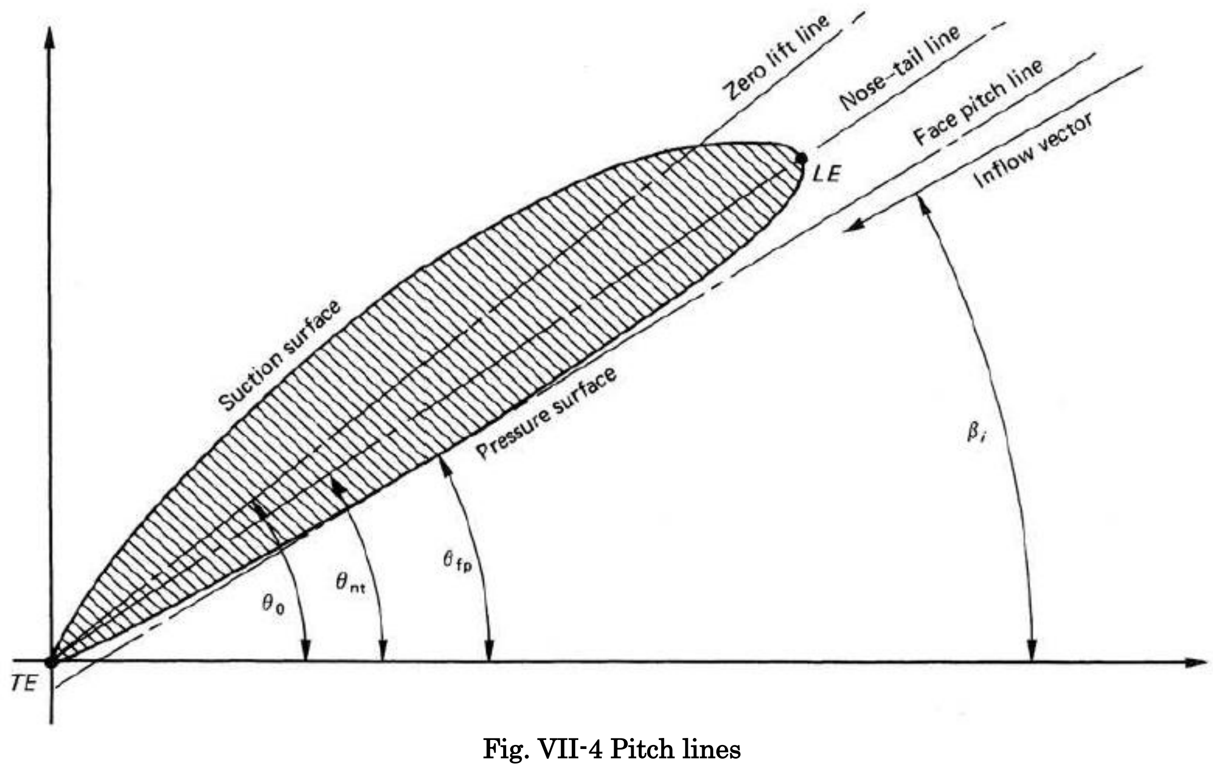


Fig. VII-4 Pitch lines

VII-1.3 Helical line

The helical line is the product of the propeller's pitch. Each point on the propeller surface leaves a helical track while the propeller is in operation, due to the combination of the rotation and the straight movement of the propeller. For a point on the cylindrical section with radius r (r<R =the propeller's radius), the following is true:

$$\theta = \tan^{-1}(\frac{p}{2\pi r})$$

Where, p is the pitch length and θ the helix angle.

Considering the above, a section's reference line will be a helix lying on the corresponding cylindrical surface and its length would be the same as the section's chord length; it will also pass by the generator line-cylinder intersection point.

VII-2 Propeller Model in CATIA

For the purposes of this thesis, the propeller models are based on the Wageningen B-Series for 4 blades or more. The propeller is left-hand (counter-clockwise) rotating. The accompanying files are presented in Table VII-1

Table VII-1 B-Series propeller model files

Parametric model file	'B-Series_Propeller.CATPart'
Model parameters file	'Wageningnen.xls'
Mesh model file	'B-Series_Analysis.CATAnalysis'
Mesh parameters file	'Wageningen_elements.xls'

VII-2.1 Design Intent

The B-series is a fixed-pitched, non-ducted propeller series that is widely used and extensively studies. Its geometry is given as a series of section offsets for sections at certain percentages of the propeller's Radius. Each propeller's geometry is defined by a set of equations that use the propeller's diameter (D), expanded blade area (A_E/A_O) and number of blades (Z). The pitch ratio (P/D) is also a main defining parameter. Subsequently, these are used as input parameters to create the B-series propeller model in CATIA.

In addition to the B-series parameters, a few other parameters are used for the propeller definition: rake angle (θrake), the hub's diameter (Dhub), the hub's slope (θhub) and the thickness at the tip (tip thickness). Especially for the last one, the tip, as defined by the B-series, is originally a zero-thickness section. However, the BEM program can't deal with zero-thickness sections. In order to solve the problem, the blade tip can either be designed as a section with thickness, or be designed according to the B-series and then trimmed. The former was selected, in order to have more control over the thickness of the blade tip. In any case, the effect on the propeller performance is negligible, since the tip contributes very little to the generated thrust.

The propeller design intent, general geometry and axes can be seen in Fig. VII-5

Taking the design intent into consideration, the propeller model parameters are saved in the Excel file "Wageningen.xls", a description of which can be found in Appendix C.

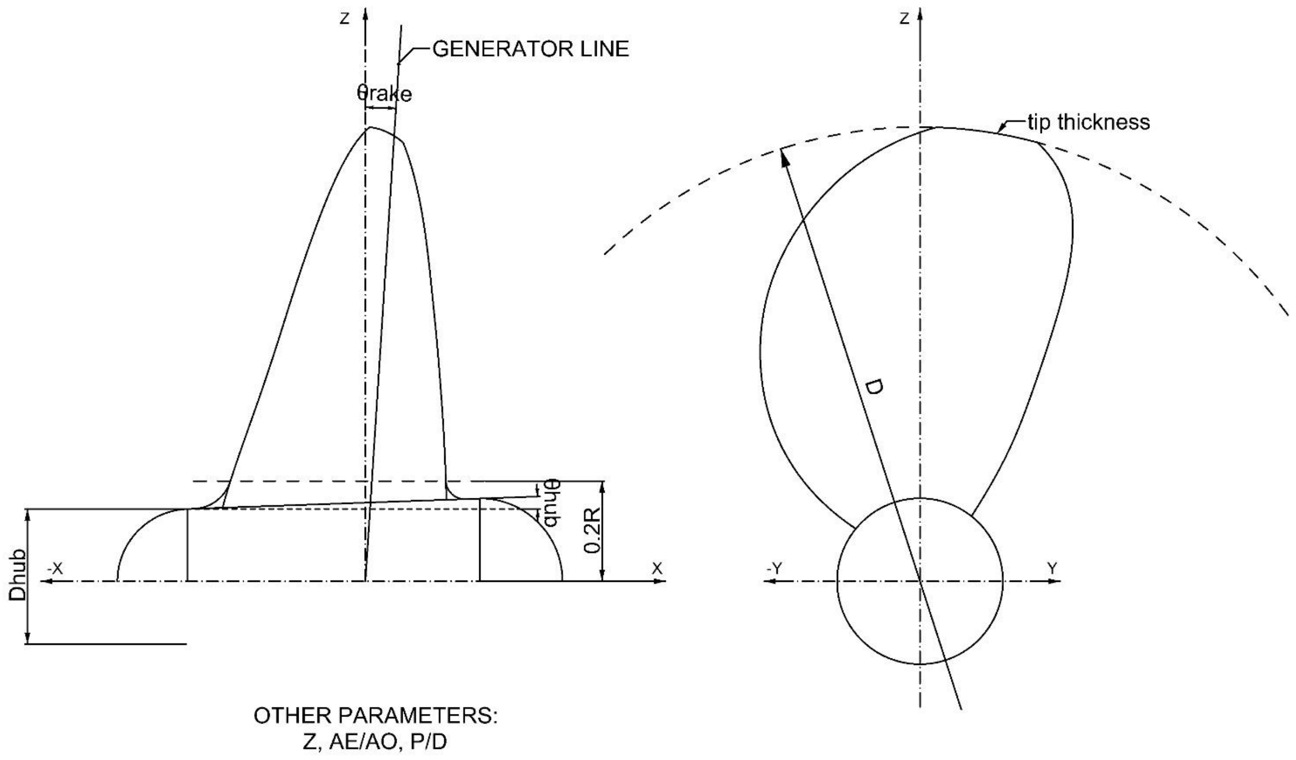


Fig. VII-5 Propeller design intent. See Table VII-2

Table VII-2 Propeller parameters

Z	Number of blades	Tip thickness	Max thickness of the tip section
D	Diameter	Θrake	Generator line rake angle
A_E/A_O	Expanded area ratio	Θhub	Hub slope angle

VII-3 Propeller Model creation overview

The propeller model consists of the blade and the hub, on which the root fillet is added as a surface-based feature. A general overview of the model creation is given, with a step-by-step description presented in Appendix A

VII-3.1 The Blade

For the propeller blade, the wireframe is first built, on which the surface is subsequently added. The wireframe consists of the cylindrical sections and the outline. To create the cylindrical sections, the flat sections are first created and constrained, according to the Series' offsets, in sketches supported by their respective planes. The flat sections are then deformed according to helical lines that have the same pitch as the respective section. Two outline splines are created by interpolating the endpoints of the cylindrical sections.

The propeller's Face and Back surfaces are built upon the created wireframe. The leading edge is completed with a strip and all the surfaces are joined. The final surface is created by adding a tritangent fillet feature on the joined surfaces to round off the leading edge strip.

Each section is constructed using its offsets, as calculated by the B-series formulas and saved in a design table. In the following description, the **Sketcher workbench** is used for each 2D drawing. For the wireframe and surface creation, work is done in the **Generative Shape Design workbench.** Exceptions will be explicitly noted. Face denotes the propeller's pressure or face side, whereas face implies the b-rep's geometry subdivision. LE and TE are Leading Edge and Trailing Edge respectively and GL is Generator Line.

VII-3.1 The Hub

The hub is a much simpler surface to build, since it's essentially a truncated cone. One simple way to create the hub surface is by revolving the hub profile around the X-axis. However, this may create problems with the updating for certain expanded area ratios, due to the way CATIA handles the revolved surface. To avoid those problems, a helix is used as the profile to create the revolved surface. The hub is completed with spherical caps at each end.

VII-3.2 The Fillet

The root fillet is created as a dress-up feature. Before creating it, the blade and the hub surfaces are joined into one object.

The propeller's trailing edge needs to close at a sharp edge, which presents a problem with the fillet around the edge. As a solution, separate fillets are created for the LE, Face and Back edges using the face-to-face fillet tool. The Face and Back fillets don't go all the way towards the TE, stopping a set distance from the edge. The remaining fillet portion is then built manually, constructing a wireframe on which a surface is created.

The root fillet starts from the 0.2r radius on the blade to the hub.

VII-3.3 Preparation for meshing

The Motion Preprocessing Module can be set to use multiple instances of the same body and give initial position and orientation to each of them. This allows only 1/zth (where z the number of blades) of the propeller to be exported as a mesh and multiplied within the MPP with the suitable initial angle. Consequently, the surface that is finally exported to the mesher consists of one blade, its root fillet and the hub portion between it and the next blade on its face side. The surface is further separated into the blade, fillet and hub, since this is how it will be input in UBEM.

VII-4 B-series meshes and runs

VII-4.1 Meshes

Using the developed method, a variety of Wageningen propeller meshes can be created.

Representative meshes from the Wageningen propeller model are shown in Fig. VII-6 - Fig. VII-19

Fig. VII-12 show different element distributions for a B4-70 propeller. A set of meshes for different geometries is shown in Fig. VII-13 - Fig. VII-19. The presented meshes are instances from the MPP program, so they include the Kutta strip at the blade's TE.

The B-series naming convention is used, to which the diameter and pitch ratio is added. For example, "B4-70 – D5P1" denotes a 4-bladed propeller with A_e/A_o=0.7, D=5m and P/D=1. Element numbers in distributions are denoted per blade face, in a chordwiseXspanwise manner.

B4-70-D5P1 geometry with different element distributions:

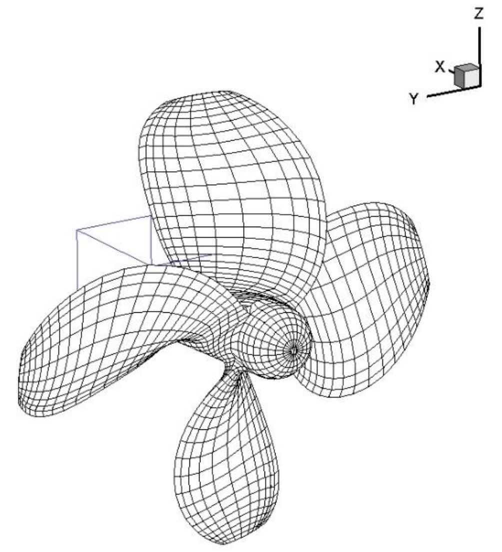


Fig. VII-6 A B4-70-D5P1 propeller, 10X20 distribution with denser mesh towards edges

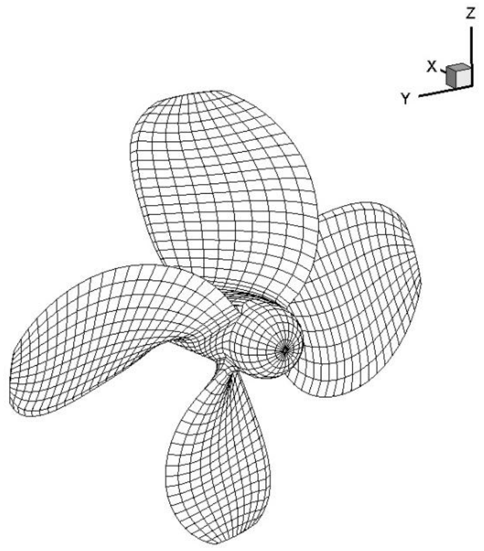


Fig. VII-7 A B4-70-D5P1 propeller, 10X20 uniform distribution

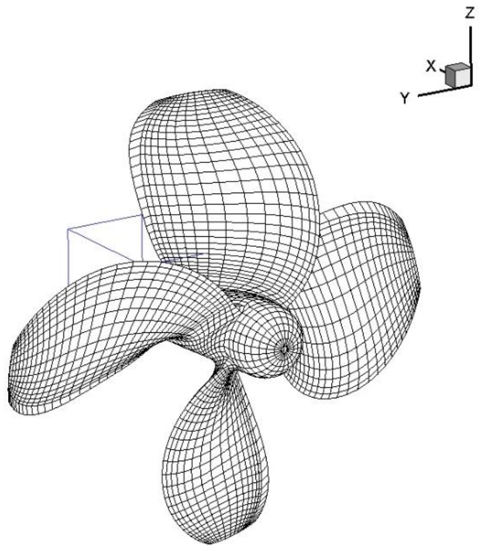


Fig. VII-8 A B4-70-D5P1 propeller, 15X30 distribution with denser mesh towards edges

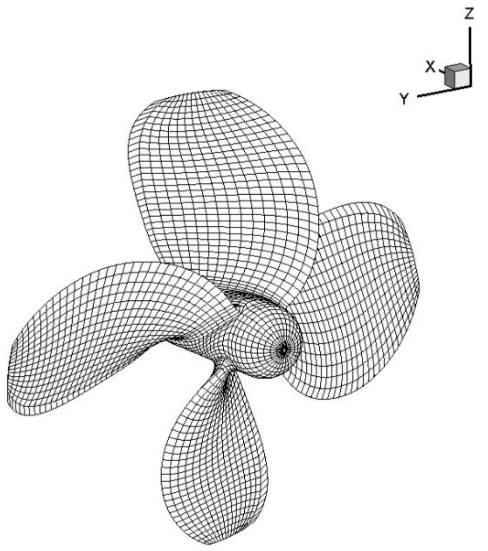


Fig. VII-9 A B4-70-D5P1 propeller, 15X30 uniform distribution

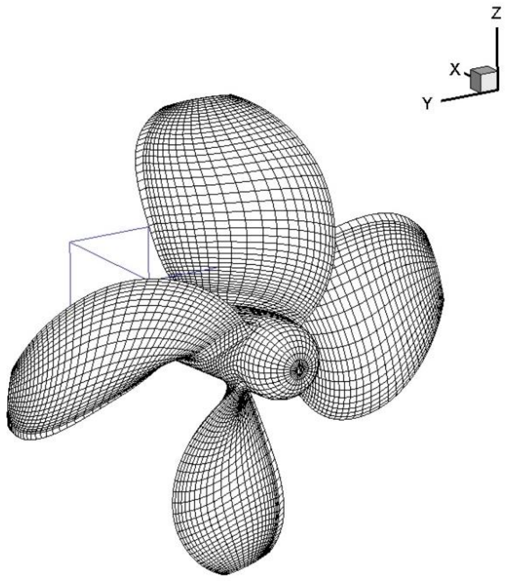


Fig. VII-10 A B4-70-D5P1 propeller 22X40 distribution with denser mesh towards edges

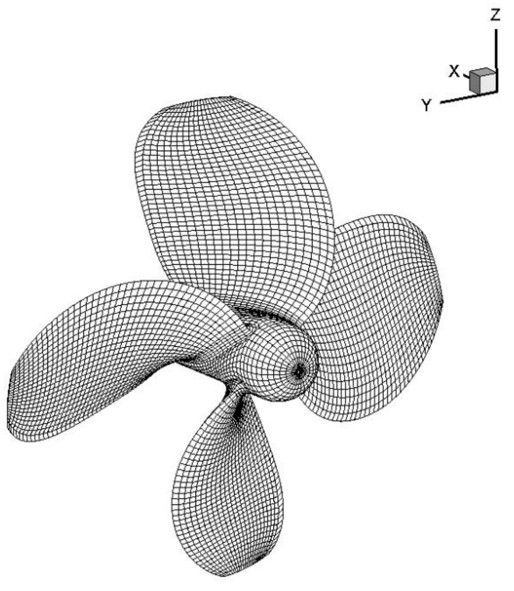


Fig. VII-11 A B4-70-D5P1 propeller, 22X40 uniform distribution

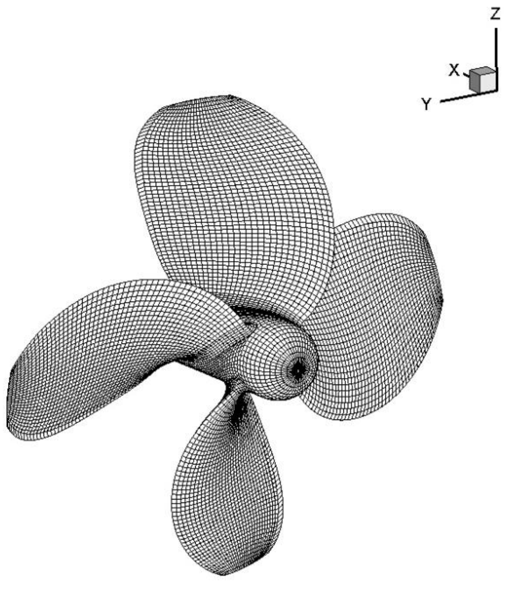


Fig. VII-12 A B4-70-D5P1 propeller, 30X50 uniform distribution

Other geometries:

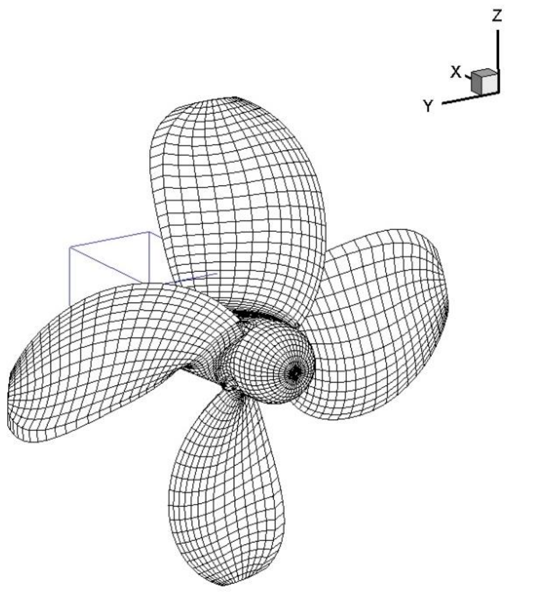


Fig. VII-13 A B4-60-D5P07 propeller, 14X22 distribution with denser mesh towards LE

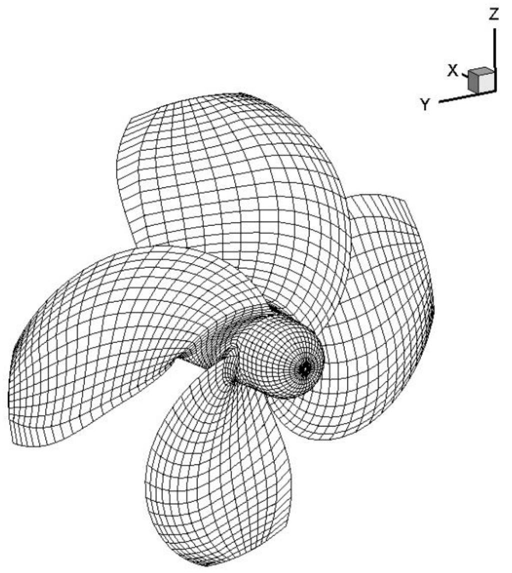


Fig. VII-14 A B4-85-D5P08 propeller, 23X15 distribution with denser mesh towards LE

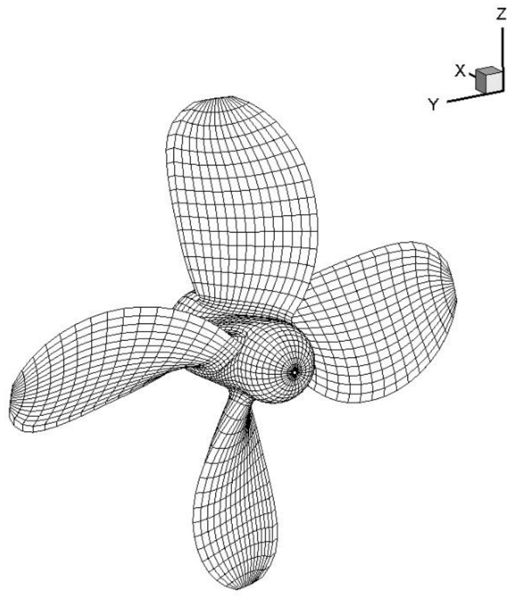


Fig. VII-15 A B4-50-D5P1 propeller, 15X15 uniform distribution

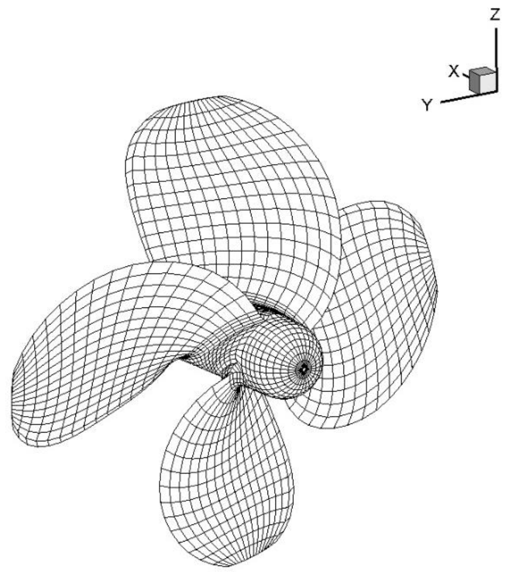


Fig. VII-16 A B4-80-D5P08 propeller, 15X15 uniform distribution

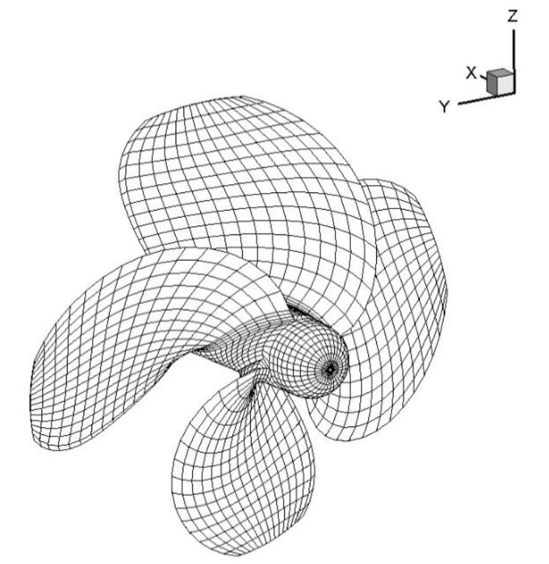


Fig. VII-17 A B4-100-D5P1 propeller, 15X15 uniform distribution

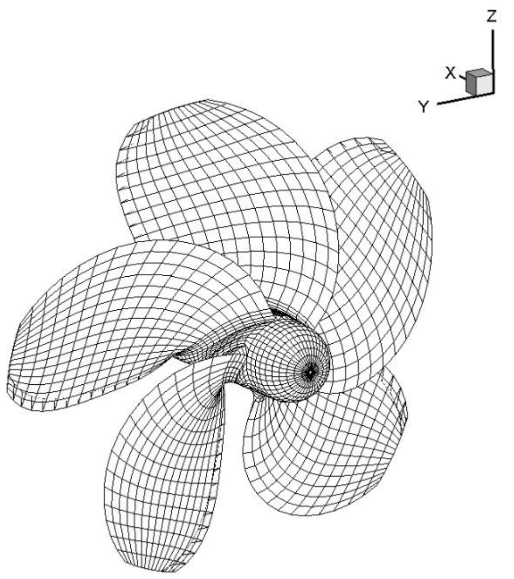


Fig. VII-18 A B5-80-D5P1 propeller, 15X15 uniform distribution

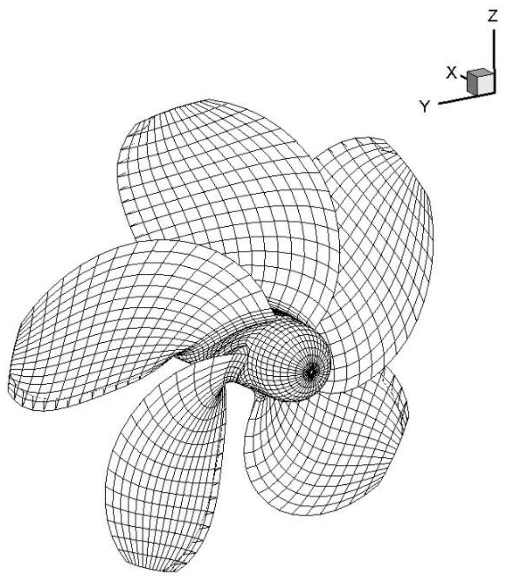


Fig. VII-19 A B5-100-D5P1 propeller, 15X15 uniform distribution

VII-4.2 Runs

In order to verify proper interaction of the produced geometry with the UBEM program, a series of runs were carried out.

The run results, some of them presented in Table VII-4, show some deviation from the test results but is within accepted margin. It is expected that further fine-tuning of the meshes will eliminate it.

2 runs of a B4-70-D5P1 propeller, with a 10X20 and a 15X30 (per blade face) distribution are presented in Fig. VII-20 and Fig. VII-21. Instances are taken at the beginning, mid and end of the run (after 3 periods).

Table VII-3 Propeller Data

D=	5	[m]
Ae/Ao=	0.7	
P/D=	1	
z=	4	
Va=	7	[m/s]

Table VII-4 Run Results

instance	10X20 uniform	15X30 uniform	10X20 denser at edges	15X30 denser at edges
J	0.6	0.6	0.6	0.6
Fx	-826.51	-754.04	-887.01	-800.73
KT	0.24	0.22	0.26	0.24
Kt(exp)	0.22555	0.22555	0.22555	0.22555
diff%	7.90	-1.55	15.8	4.54
Mx	-651.11	-592.85	-695.87	-630.83
KQ	0.03835	0.034916	0.04098	0.03715
KQ(exp)	0.03727	0.03727	0.03727	0.03727
diff%	2.89	-6.32	9.96	-0.32
η exp	0.578	0.578	0.578	0.578
η res.	0.607	0.608	0.609	0.607
diff%	4.87	5.08	5.31	4.87

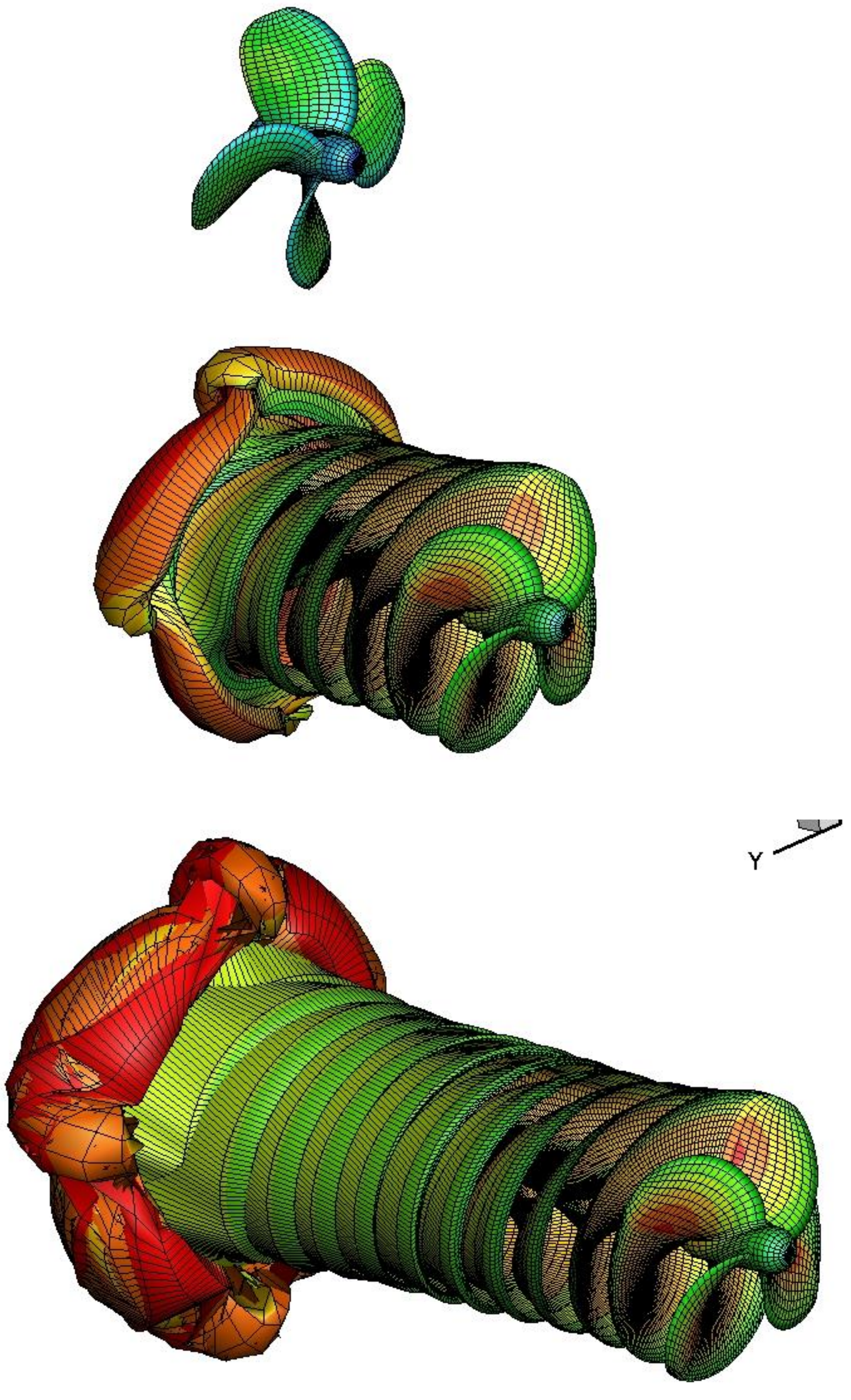


Fig. VII-20 A B4-70-D5P1 propeller, with a uniform 10X20 element distribution (per blade face) at t=0, 0.66 and 1.37s

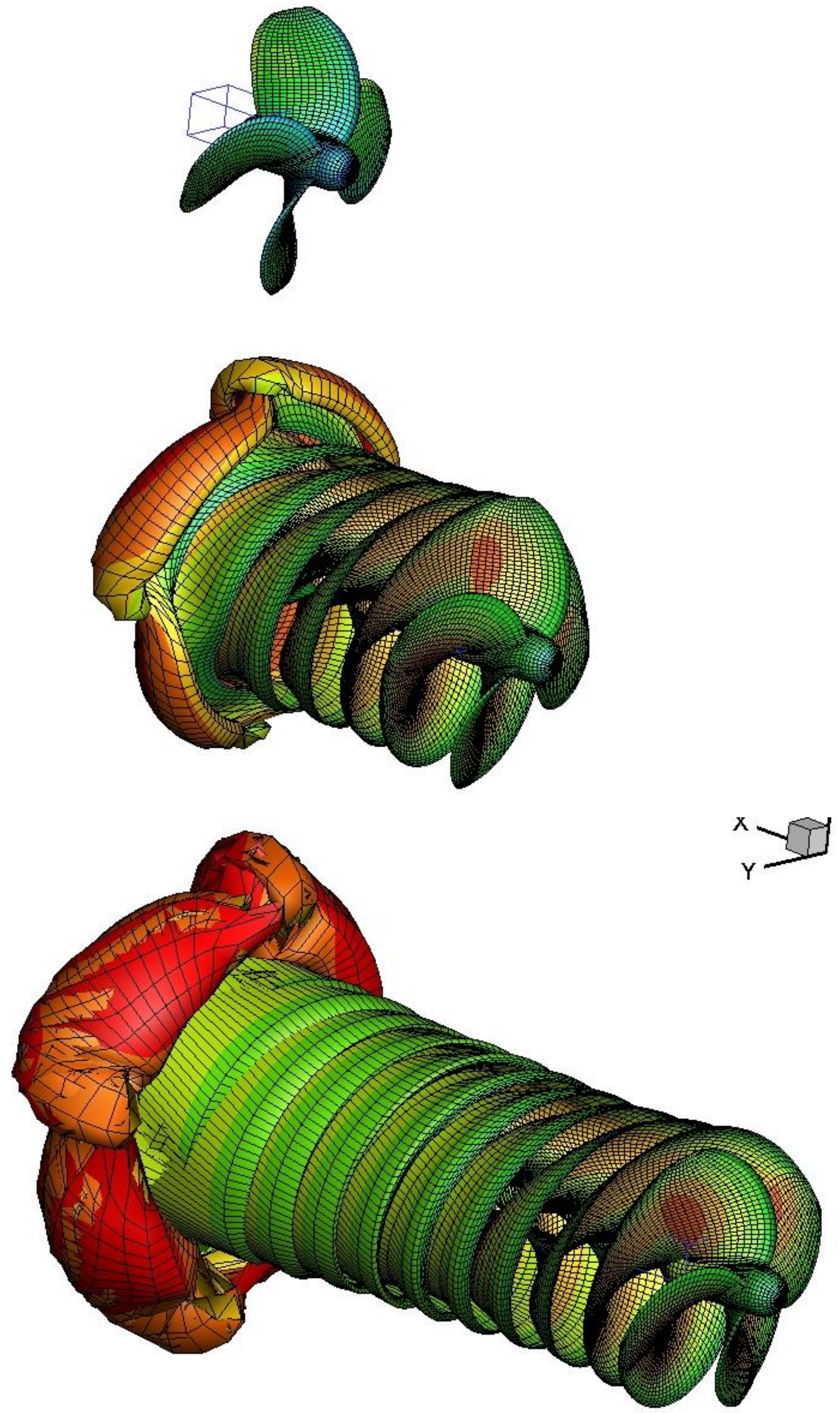


Fig. VII-21 A B4-70-D5P1 propeller, with a uniform 15X30 element distribution (per blade face) at t=0, 0.66 and 1.37s

Part VIII. Application: an Azimuthing propulsor

The azimuthing or podded propulsor model that was created is modelled after ABB's Azipod® XO and Rolls-Royce's Mermaid® propulsors. This type of propulsor consists of the motor module (pod) that attaches on the ship's hull via the strut. These specific models include one pulling propeller.

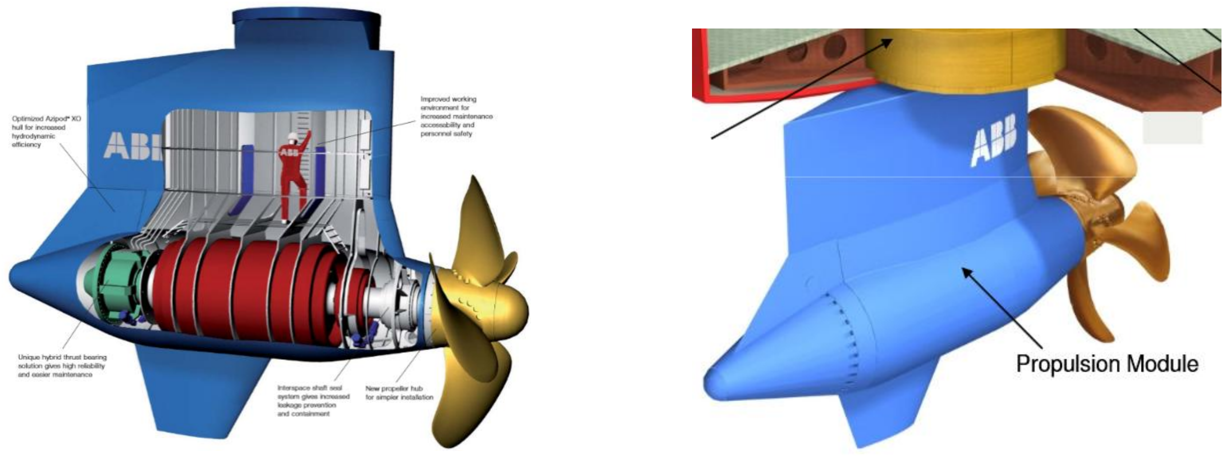

Optimized Azipod® XO hull for increased hydrodynamic efficiency
Improved working environment for increased maintenance accessability and personnel safety
Unique hybrid thrust bearing solution gives high reliability and easier maintenance
Interspace shaft seal system gives increased leakage prevention and containment
New propeller hub for simpler installation
ABB
Propulsion Module

Fig. VIII-1 ABB Azipod XO (source: www.abb.com)

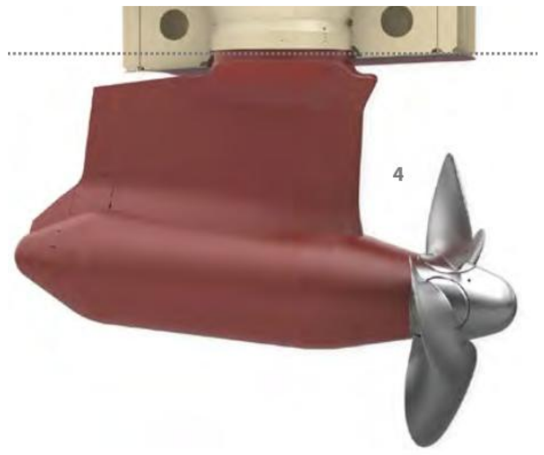


Fig. VIII-2 Rolls Royce Mermaid

(source: www.rolls-royce.com)

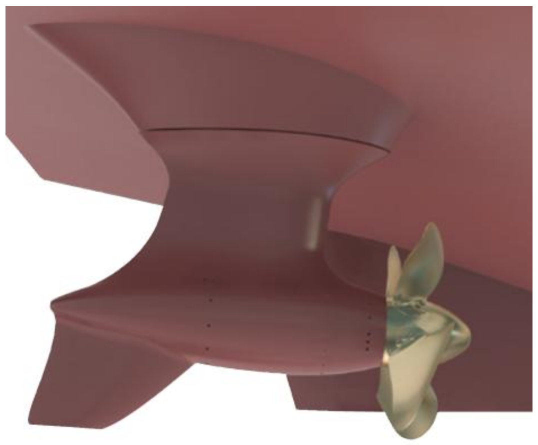


Fig. VIII-3 Rolls Royce Azipull

(source: www.rolls-royce.com)

VIII-1 The propulsor segments

The propulsor unit can be broken down into 3 main segments: the strut, the pod and the propeller, as seen in Fig. VIII-4

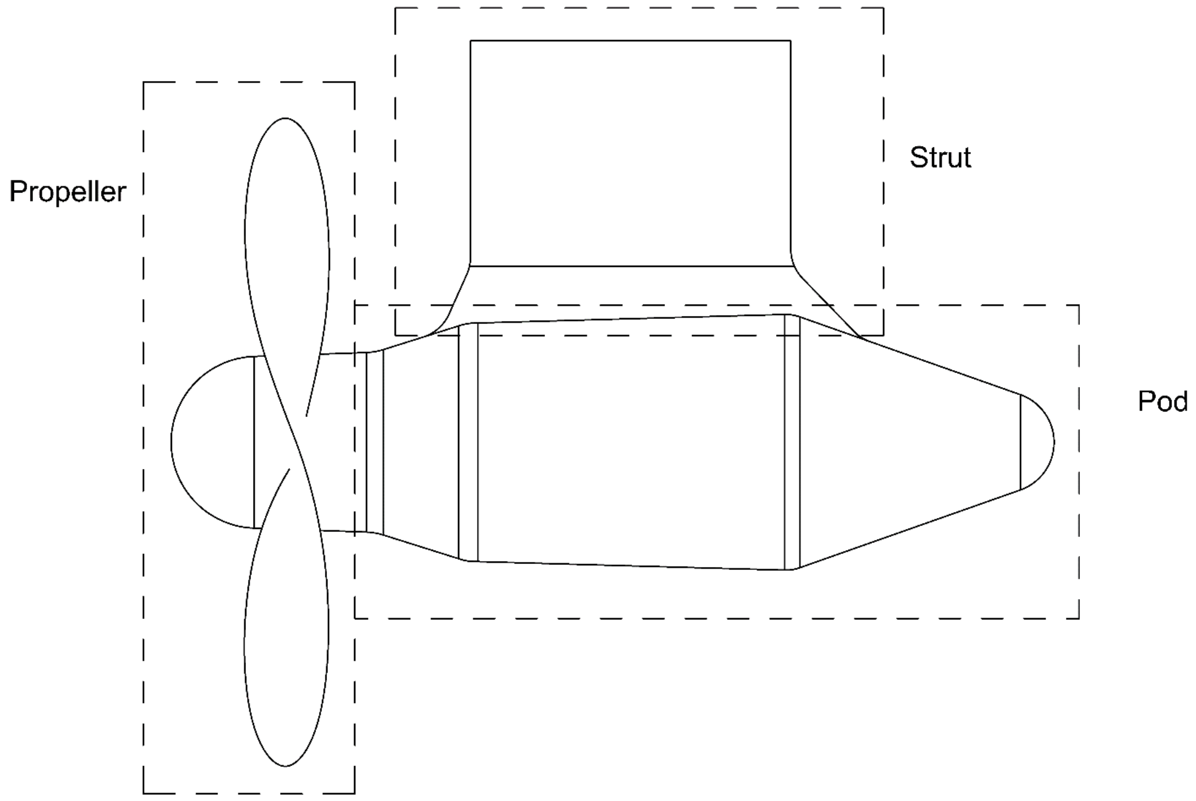


Fig. VIII-4 Podded propulsor main geometric segments

VIII-1.1 The propeller

For the propeller segment, the Wageningen B-series propeller model described in the previous chapter is used. However, CATIA's intrinsic modularity enables swapping out this propeller for another, so a designer can easily create another propeller model and use that with the propulsor's main body.

VIII-1.2 The strut

The propulsor's strut is its connection with the steering unit installed on the ship, and subsequently with the ship's hull. It is a wing-like structure, with a hydrofoil cross-section. Spanwise, two zones can be distinguished: the upper zone, towards the ship and the lower, where the pod attaches. The lower zone shows a slightly different geometry, especially on the rear end of the strut, where its profile changes angles to attach to the pod. This is more notable on the Azipod model.

VIII-1.3 The motor module (pod)

The pod's geometry can roughly be described as a (tapered) cylinder with truncated cones on either end. The front end attaches to the propeller and the rear end closes off with a cap. The edges are rounded off with fillets.

VIII-2 Azimuthing propulsor model in CATIA

As was described in the propulsor's geometry paragraph, the main body's geometry can be constructed using a set of different solids: (i) a wing with foil cross-section for the strut and (ii) a combination of three truncated cones and a hemisphere for the pod. On that, the propeller with the front cap is attached.

VIII-2.1 Model Axes

The X-axis coincides with the model's longitudinal axis, propulsor looking towards the negative, the Y axis coincides with the model's transverse axis and Z axis with the model's vertical axis. The model is created symmetrical, with the XZ plane as the plane of symmetry. The model's Axis origin is located so that it coincides with the centre of the pod's front (circular) edge.

VIII-2.2 Design intent

The design intent and selected driving parameters for the pod can be seen in Fig. VIII-5. An Excel file is created, containing geometric the parameters for the propulsor body, as they appear in Fig. VIII-5. In addition the hydrofoil section design parameters, seen in Fig. VIII-6, are included in the same file. The parameters are input in CATIA, in the form of design tables.

VIII-3 Model creation overview

VIII-3.1 Strut

To create the upper section of the strut, the foil cross-section is swept along the outline. The same goes for the lower strut section. For this, a wireframe that consists of the upper foil sketch, the lower foil sketch and the outline is created for half of the propulsor. The surface is subsequently created and then mirrored by the plane of symmetry to create the other half.

Foil Section

The foil section is defined using 3 points and the chord: The Leading Edge (LE) point, maximum thickness point and Trailing Edge (TE) point. The LE point coincides with the chord's LE. The maximum thickness point is defined by its offsets from the LE and the chord. Both of them are given as a percentage of the chord's length. The TE point lies on the a line perpendicular to the chord that passes from the Chord's TE and is offset by a length also determined as a percentage of the chord's length.

The section is symmetric about the chord. The trailing edge closes off with a tangent circle that has its centre on the chord (Fig. VIII-6)

<u>Wireframe</u>

The upper foil section is defined by its maximum thickness, location of maximum thickness and trailing edge thickness, all of which are percentages of the chord's length (determined by the outline).

For the lower section, a choice was made to keep the same thickness in the same model-relative X position. Its chord length also is determined by the strut outline.

The outline is created as a sketch on the plane of symmetry. Splines were used for the upper curves, so that a propulsor similar to the ABB mermaid model can be created from the same 3D model. If this is not desired, lines can be used instead, as was used in the lower outline.

<u>Surface</u>

To create the strut surface, two multisection surfaces are created, using the respective hydrofoil sections and outlines as profiles and guides respectively. A fillet is created between the upper and lower strut surface to create a smooth join.

VIII-3.2 Pod

The pod is created with a series of truncated cones joined together. Fillets are created on the joint edges. The rear is completed with a spherical cap tangent to the rear truncated cone.

Each cone is created by blending between two circles. The circles are created on planes parallel to the XY plane, offset by the respective distances LP1, LP2, LP3, LP4

For the cap, a sketch of a circular arc tangent to the truncated cone is created after the surface creation, which is then revolved around the X axis to produce the surface.

The final (mirrored) strut surface and the pod surface are joined together and a fillet is applied on the join edge.

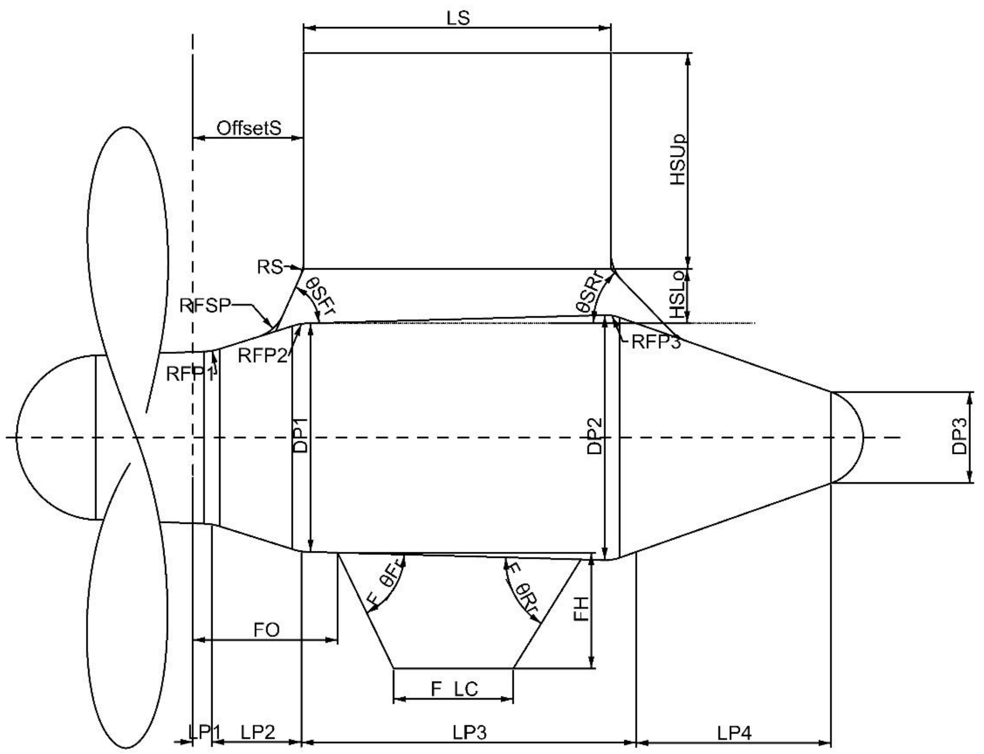


Fig. VIII-5 Podded propulsor Side - Design Intent. The geometric parameters used to create the model be seen in Table VIII-1

Pod body		Strut	
Parameter	*Description*	*Parameter*	*Description*
LP1 – 4	Length of pod cones that make up the pod body	OffsetS	Offset of Strut Leading Edge from X axis
DP1 – 3	Diameter of Pod cone edges	LS	Strut Length
RFP1 – 3	Radii of fillets on cone edges	HSUp	Height of upper strut
		HSLo	Height of lower strut (from pod body edge)
		θSFr – Rr	Angle of lower strut front and rear edges
		Rs	Radius of fillet upper and lower strut join edge

Table VIII-1 Pod design parameters

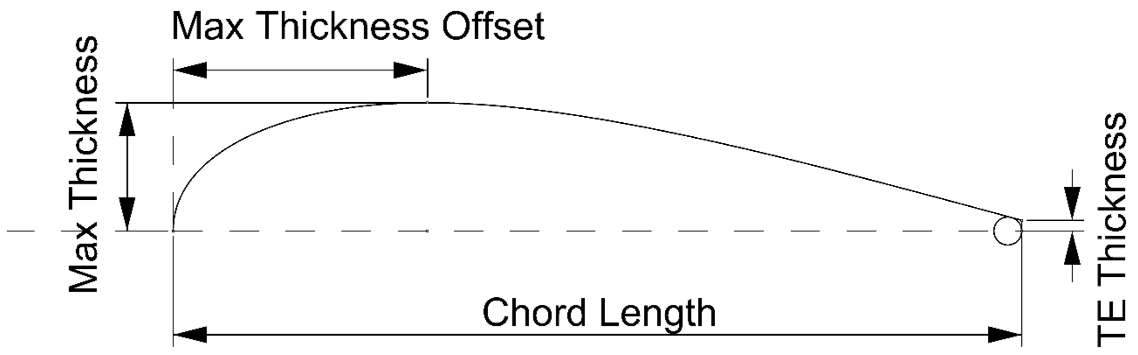


Fig. VIII-6 The strut hydrofoil section

VIII-4 The Assembly

VIII-4.1 Assembly Definition

An assembly is a document that stores a collection of components. The components used in the assembly can be pre-existing, or created within the assembly. The components are constrained in relation to each other and in space. They can also interact between them by using shared parameters.

VIII-4.2 The assembly materialized in CATIA

CATIA's assembly capabilities can be accessed in the Assembly Design workbench, under the Mechanical Design product.

The components are initially inserted with their origin points and axes coinciding with the assembly's origin point and axes. This frequently means that the components overlap and need to be placed in relation to each other and to their required positions (Fig. VIII-7). This is achieved by using a set of constraints, such as coincidence, contact, offset, angle and fixing.

In addition, certain measurements might need to be communicated from one component to another. This is achieved by copying the parameter from the component of origin and pasting it to the target component, using the paste special command. In order to maintain the parameter's updatability but also its link with the source component, it should be pasted "As result with Link".

The Assembly enables adding, removing and replacing components. This means that, for example, the Wageningen model used for the current pod can be replaced with another propeller model. However, in case of swapping out components, the mesh of the old component isn't transferred to the new one, which needs to be meshed from scratch.

VIII-4.3 The pod Assembly

In order to create the final pod model, the propeller body described in Chapter 3.2 and the Wageningen propeller model described in 3.1 where put together in an assembly.

Components

Propeller

The Wageningen propeller model describedin [Chapter] was used. For the Pulling propeller, the rear hub surfaces are not used. This is achieved by defining the rear cutting plane as the plane to be used with the Assembly constraints. In addition, a length parameter, set to reference, is created at the edge of the hub midto measure the edge's radius.

Pod

For the pod component, all of the final meshing surfaces are used

Constraints

Geometrical constraints

The propeller was arbitrarily chosen to be the anchoring component. So, a Fix constraint is applied to it. This way, the propeller can't be moved around.

In order to properly place the pod body in relation to the propeller, a coincidence constraint between the respective planes on the propeller and pod is assigned. More specifically, the cutting plane that split the hub mid and rear coincides with the pod's local YZ plane. By imposing this constraint, the pod body component is moved in such a way that the two planes coincide.

External parameters

Both components are fully defined using their respective Excel files. To simplify the assembly, a choice is made to keep this as is instead of creating a new file. However, there has to be at least some inner-communication between the components to avoid external calculations. For this case, a length parameter is created that savesthe measured length of the propeller's edge. This parameter is then pasted with the paste special command into the pod model, while preserving the link to the propeller model. Last, the pod's front circle's diameter (circle_p1) takes on its diameter value from the external parameter.

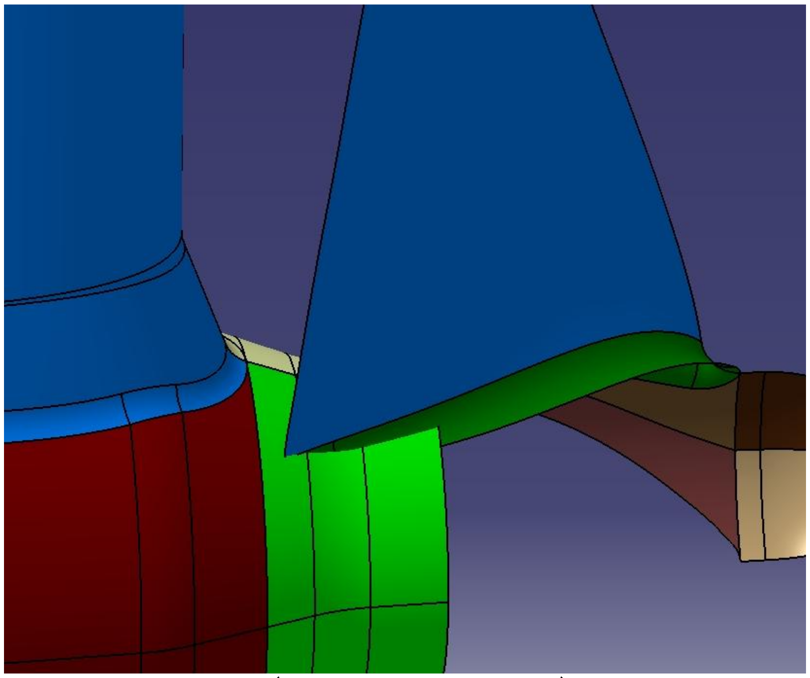


Fig. VIII-7 The assembly components (pod body and propeller) upon insertion. Before constraining they overlap

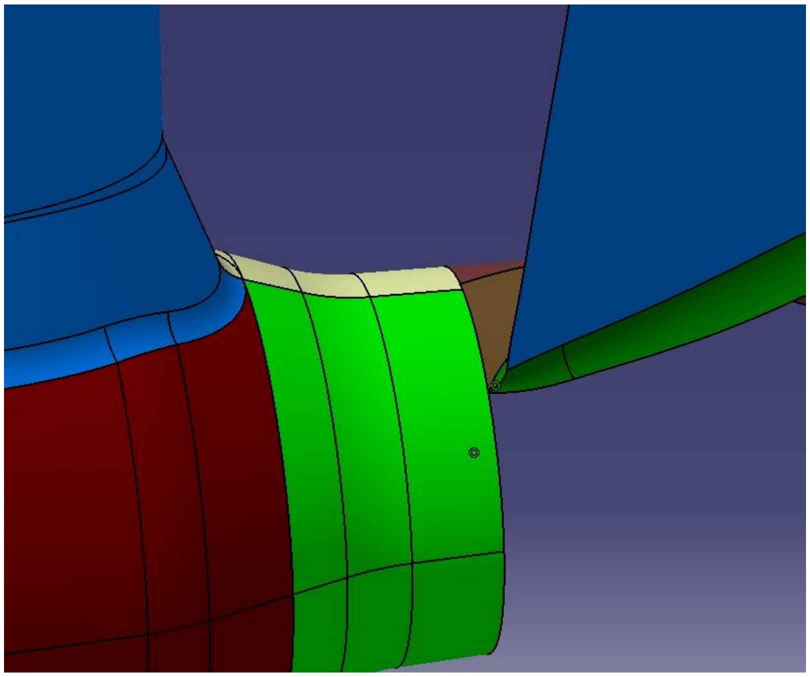


Fig. VIII-8 The assembly after being constrained. The components take their proper place relative to each other

VIII-5 Meshing

VIII-5.1 Meshing the Assembly

The mesh creation of an assembly is no different to the mesh creation of any other part. Each surface created receives a structured mesh, following the general method of meshing. A description for the excel file can be found in Appendix D.

For the propeller section, the same method is followed as with the propeller model.

The pod's surfaces are regarded as follows:

The strut and strut cap are meshed in a way similar to the propeller's blade surface (Face-back type)

The pod is meshed in a way similar to the propeller's hub. (Modified Face-back)

VIII-6 Podded propulsor meshes and runs

Lacking test results, it was considered suitable to check the produced pod geometry against its implementation in the BEM program. For this, models with configuration similar to Azipod XO, with pulling propellers (the pod body follows the propeller), were initially checked. The interaction of the propeller wake with the pod strut presented problems that could not be overcome presently. To verify that a pod model could complete a full run, a configuration with a pushing propeller (the propeller follows the pod body) was created and run successfully.

A series of different pod meshes are presented (Fig. VIII-9 - Fig. VIII-11) as well as instances from one of the successful runs with the pushing propeller configuration (Fig. VIII-12).

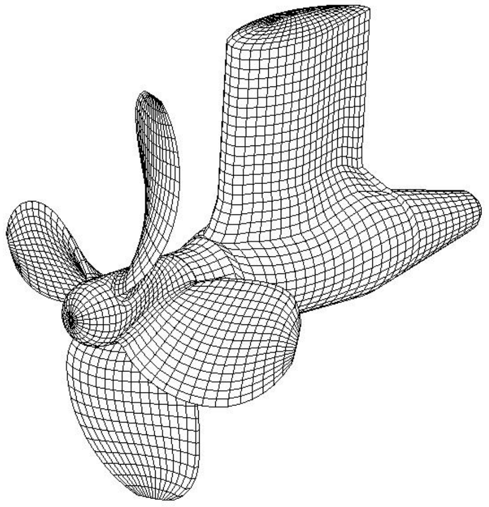


Fig. VIII-9 A pod with a B4-70D5P1 Wageningen propeller

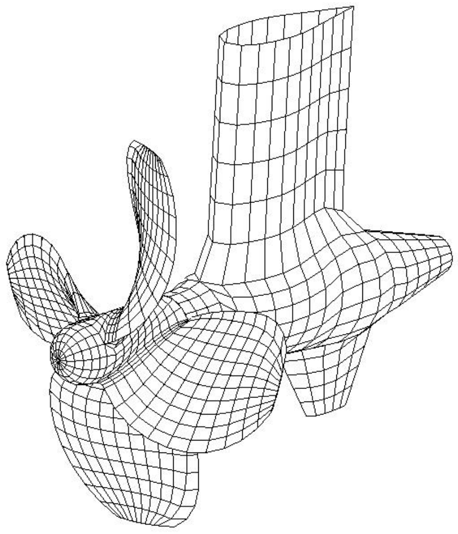


Fig. VIII-10 A pod with a B4-70D5P1 Wageningen propeller and 1 fin

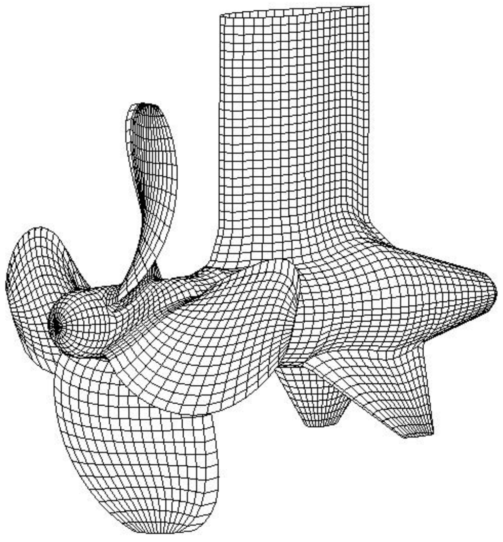


Fig. VIII-11 A pod with a B4-70D5P1 Wageningen propeller and 2 fins

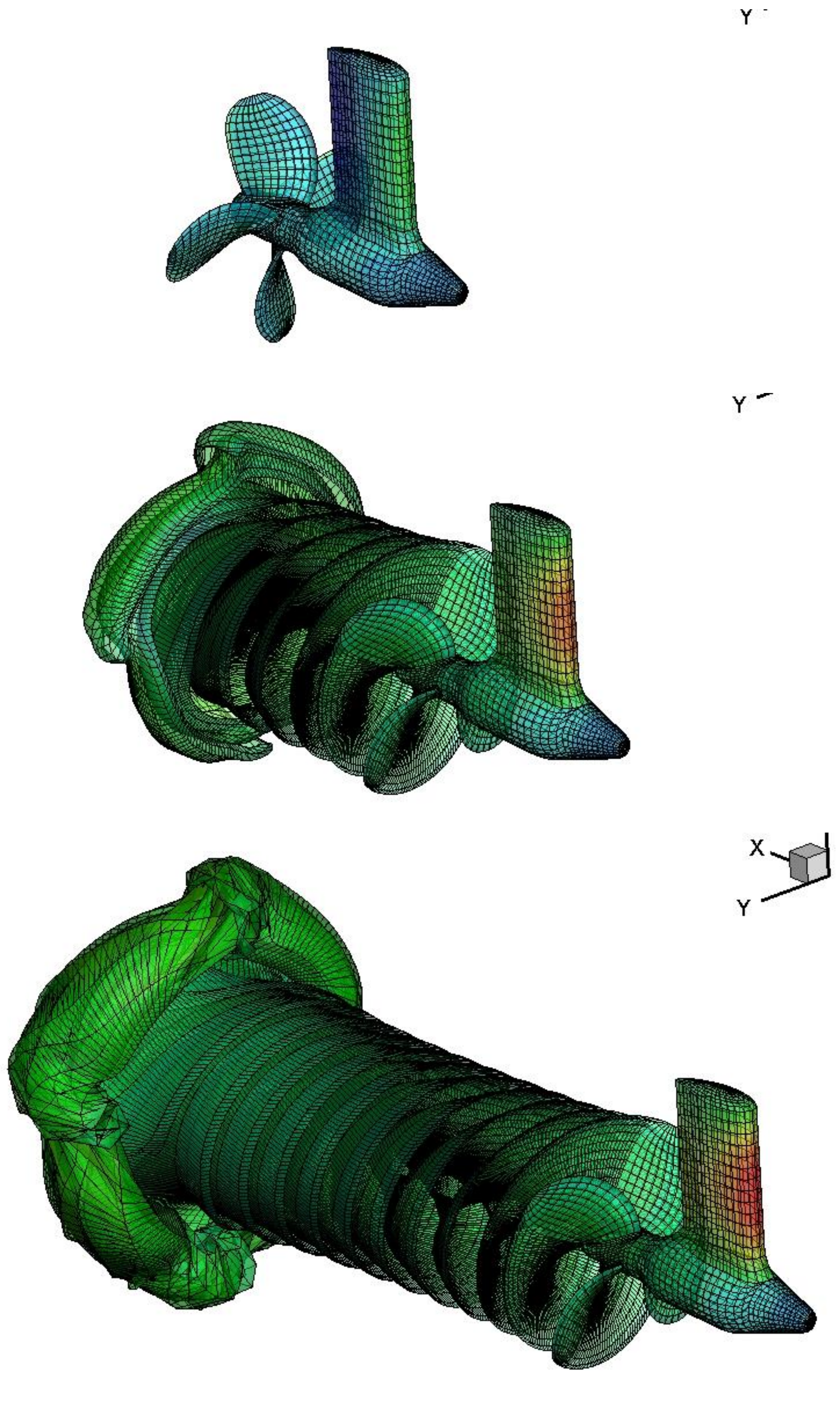


Fig. VIII-12 A pod configuration with a pushing propeller (B4-60-D5P1) at t=0, 0.67 and 1.37 s

Part IX. Application: SMP Workshop Propeller

A very important capability of the proposed meshing method is the ability to create a mesh from an imported geometry in a format supported by CATIA. Analytic description of such surface could be very time consuming. However, the importing, clearing up and preparation of the surface for meshing, as well as the meshing itself, is relatively quicker and easier with the proposed method.

IX-1.1 Data provided

To present this, the model of a controllable pitch propeller, VP1304, by SVA is used. The propeller was designed for academic purposes, with the intention to generate a stable tip vortex. Its main particulars are as presented in Table IX-1:

Propeller diameter	DP	[mm]	250.0000
Pitch at r/R=0.7	P0.7	[mm]	408.7500
Pitch at r/R=0.75	P0.75	[mm]	407.3804
Mean pitch	Pmean	[mm]	391.8812
Chord length at r/R=0.70	C0.70	[mm]	104.1670
Chord length at r/R=0.75	C0.75	[mm]	106.3476
Thickness at r/R=0.75	t0.75	[mm]	3.7916
Pitch ratio	P0.7/D	[-]	1.6350
Mean pitch ratio	Pmean/D	[-]	1.5675
Area ratio	AE/A0	[-]	0.7790
Skew	θ_{eff}	[°]	18.8000
Hub diameter ratio	dh/DP	[-]	0.1500
Number of blades	z	[-]	5
Direction of rotation			right-handed

Table IX-1 Propeller main particulars

The propeller form was presented both in the form of an offset table and as a series of 3D geometry files that additionally contained the rest of the tests setup models.

The file that is chosen to be imported to CATIA is the IGES (.iges) file. IGES (Initial Graphics Exchange Specification) is a vendor-neutral file format that allows the digital exchange of information among computer-aided design (CAD) systems. It is the file format most widely supported by CAD software, as well as programs that receive input from CAD software in the form of 3D geometry.

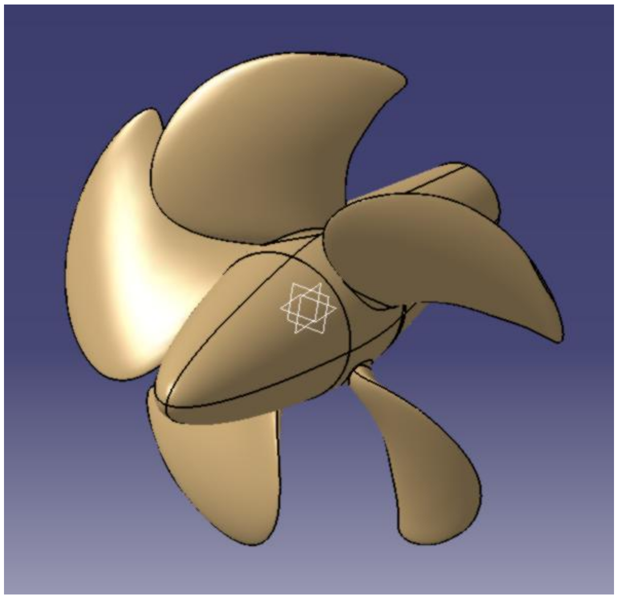


Fig. IX-1 The provided IGES model

The imported geometry does not contain a specification tree or any form of model history. The only imported objects are the surfaces that constitute the final object (Fig. IX-1). To create the mesh, the surfaces need to be manipulated in order to create surfaces suitable for use with the current method. This manipulation consists of joining and splitting surfaces, as well as creating some geometry such as curves and points to be used as splitting objects.

The mesh is created, as in Wageningen, for the 1/Zth of the model, which is then revolved inside the UBEM program. For this, the input model is segmented into 3 parts: the blade, the fillet and the hub, which are in turn split in order to follow the face-back input format the UBEM program requires. A description of the modifications done on the model are presented below:

IX-1.2 Blade

1. Each blade face to be used is joined together with its corresponding fillet surface.
2. A truncated cone surface, offset from the hub, with a diameter of 0.2R is used to split the joins into the Blade faces and the fillet.
3. The Blade faces are then split at the tip using a spline, based on the initial projected outline of the blade (Fig. IX-2). The spline is created and then extruded along the X direction to create the cutting surface.

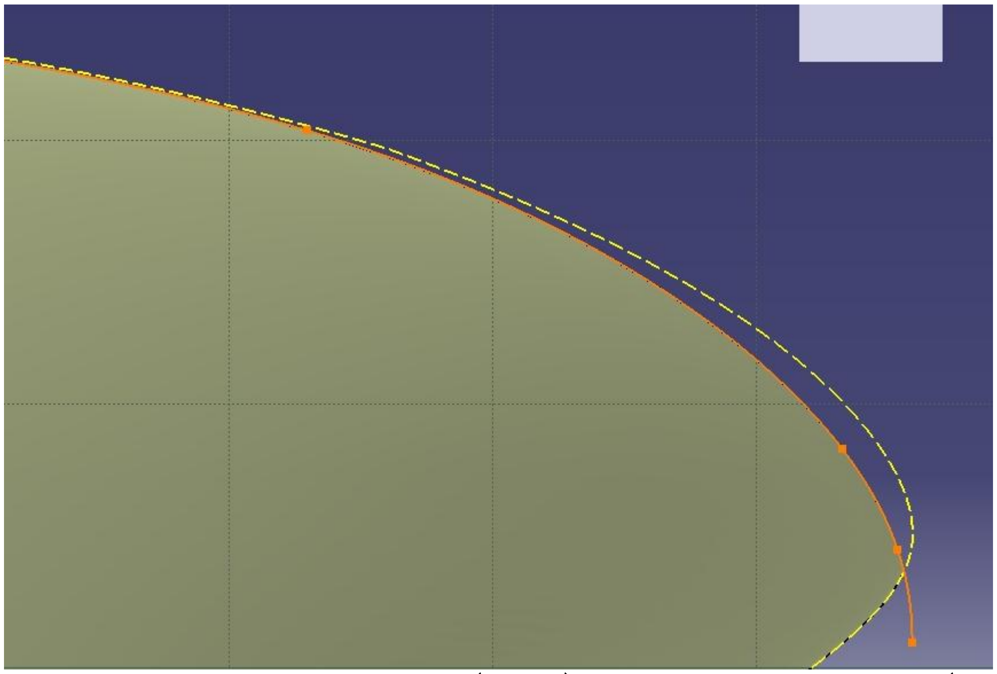


Fig. IX-2 The blade tip is split using a spline (orange) similar to its original outline (in dashed)

IX-1.3 Fillet

4. The fillet leftovers from the split in step 2 are joined with the hub.
5. The fillet's edge is extracted and the two supremums alongside the X direction are defined.
6. Since the propeller is designed as a Controllable Pitch one, the fillet's outline is unsuitable for the current method. So, a new spline, supported by the hub, is created around the fillet edge, interpolating the two supremums. The spline is created in such a way that the two supremums of the outline are also the supremums of the spline. This is achieved by constraining the tangency on the two points to be parallel to the Y axis.
7. Using the spline, the Join from step 4 is split into the fillet and the hub.
8. New Leading and Trailing edges are defined on the fillet, that end to the Leading and Trailing supremum respectively. The fillet from step7 is split into the Face and Back segment

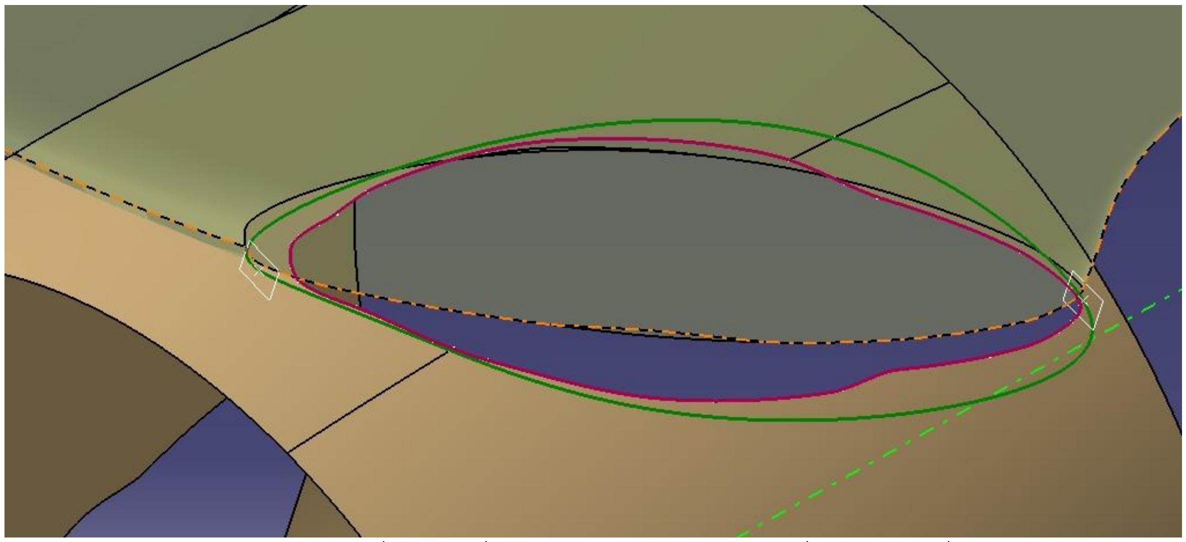


Fig. IX-3 The new fillet outline (in green), v.s. the old fille outline (in magenta) on the original hub surfaces provided by SMP. At either edge the two extremum points and cutting planes can be seen. The transparent surface is the propeller blade. The rest of the root fillet surface is hidden.

IX-1.4 Hub

9. The hub from step 7 is split with two planes parallel to plane YZ, that pass from the two supremum points. The hub front and middle (surface between them) are kept.
10. The spline from step 6 is copied and rotated around the X axis by 360/Z degrees (72 deg, since Z=5). The rotated copy splits the mid of the hub from step 9.
11. The hub middle is split in a manner similar to the Wageningen model, creating planes along its length, finding their intersections with the surface and then interpolating a spline through the midpoints of the intersection, supported by the surface. The spline splits the middle in two.
12. The rear hub is recreated as a truncated cone, in a manner similar to the Pod body model. The circles used for the blending have the same diameter as the original hub in the same place. A spherical cap created in the same way as the caps in the previous models is also added.
13. A circular hole is created to the Hub Front and Rear, with a small diameter, centred on the X axis
14. Both the front and the rear hub portions are split using 3 planes each, defined as following:
    a) First plane is created by rotating plane XZ around X axis, passing from the supremum point that is nearer the portion to be split
    b) Second plane is created by rotating first plane around X axis, by 360/Z degrees
    c) Third plane is the bisecting plane of first and second planes (it should pass from the spline endpoint)
15. After their creation, all of the final surfaces are rotated around the Z axis by 180deg. This is done because the UBEM program requires the forward direction towards the

negative X-axis, whereas the SMP model's forward direction is towards the positive X-axis.

IX-1.5 Created Surface

The final surfaces created are presented in Fig. IX-4:

- Face and Back of propeller Blade
- Face and Back of propeller Fillet
- Upper and Lower of Hub Front, Middle and Rear.

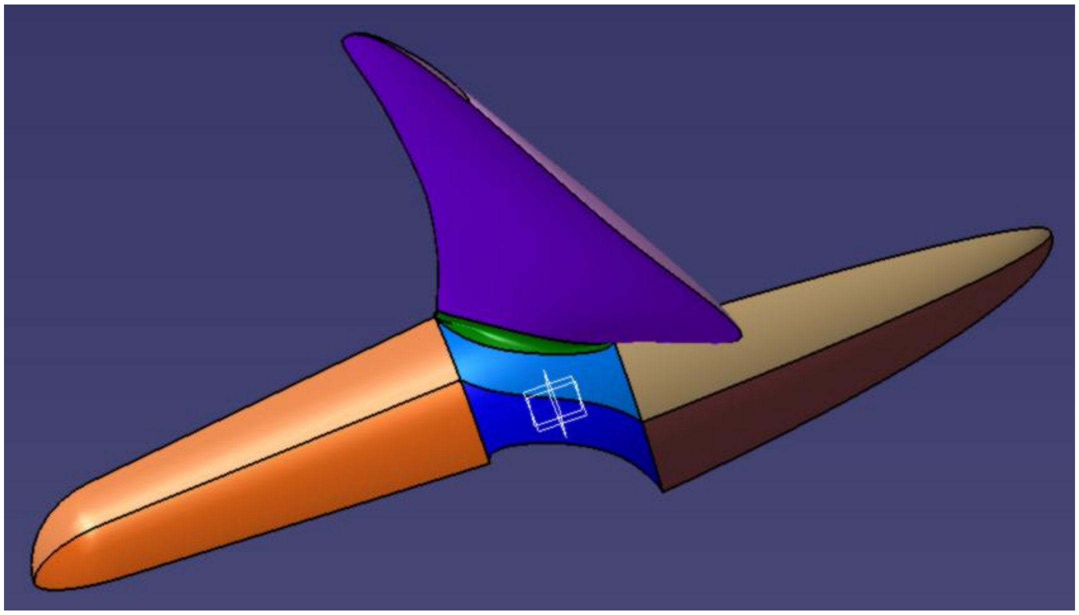


Fig. IX-4 The final surfaces to be meshed: Purple and pink: the blade pressure and suction side respectively; Green: the root fillet; Shades of brown: the hub front; Shades of blue: the hub mid; Shades of orange: the hub rear.

IX-2 SMP propeller meshes and runs

A set of different meshes were created from the provided IGES geometry and ran in the UBEM code with different degrees of success.

Some indicative meshes that were created with the method are presented in Fig. IX-5 - Fig. IX-8, whereas some of the run results are presented in Table IX-2

Table IX-2 SMP runs results

instance	15X20 Dense edges	23X30 Dense edges	25X40 Blades only
Fx	0.97	0.91	-0.96
K_T run	0.33	0.31	0.33
K_T provided	0.36	0.36	0.36
diff%	-8.30	-13.85	-9.25
Mx	0.0637	0.0614	0.0654
$10K_Q$ run	0.867	0.835	0.889
$10K_Q$ provided	0.960	0.960	0.960
diff%	9.72	13.06	7.44
η from prov.	0.609	0.609	0.609
η run	0.618	0.603	0.597
diff%	-1.56	-0.91	-1.95

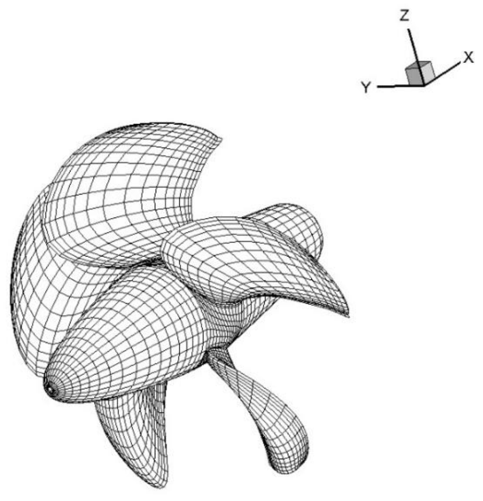


Fig. IX-5 SMP propeller, 15X18 mesh distribution, denser towards edgesz

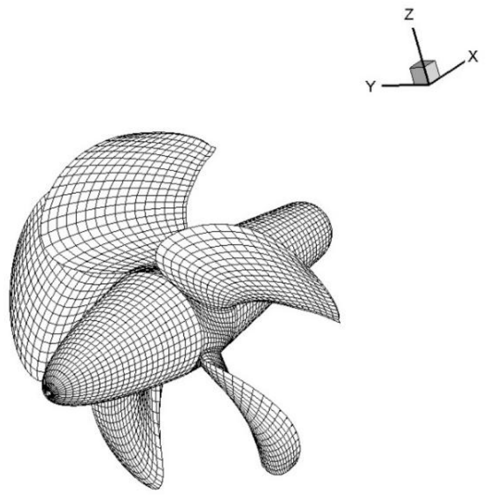


Fig. IX-6 SMP propeller, 15X20 uniform mesh

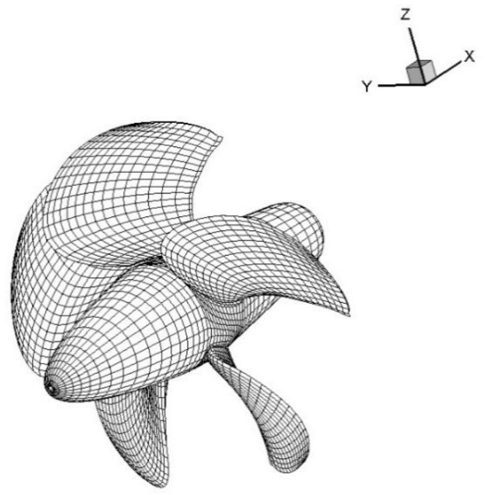


Fig. IX-7 SMP propeller, 15X20 mesh, denser towards LE

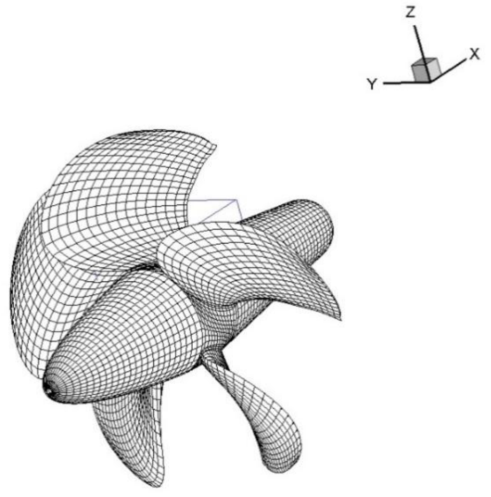


Fig. IX-8 SMP propeller, 20X30 uniform mesh

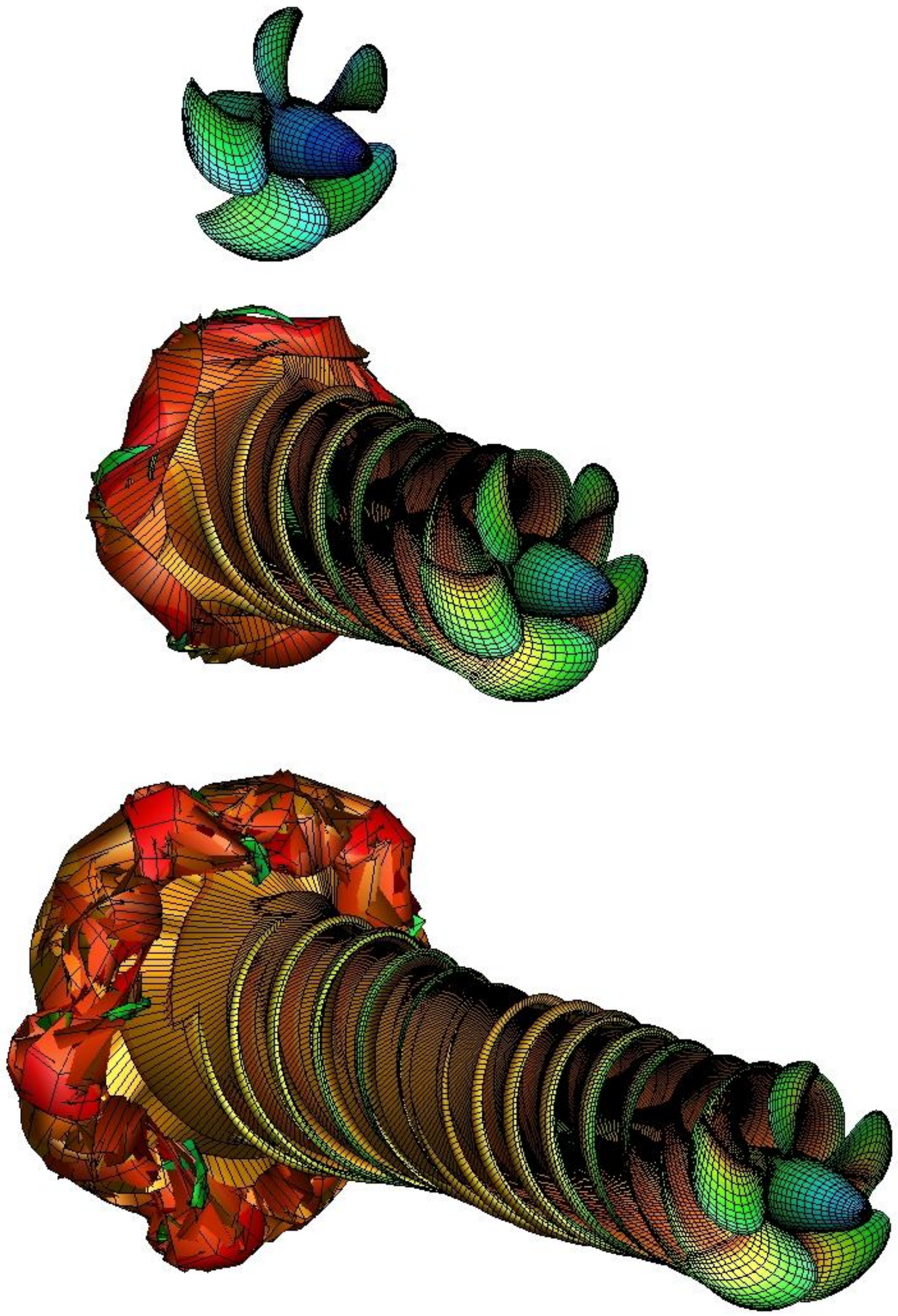


Fig. IX-9 SMP propeller instances, 15X10 denser mesh towards edges, at t=0, 0.06 and 0.12 s

Part X. Conclusions & Future Work

The aim of this thesis was to develop a methodology to generate computational meshes for BEM simulations for marine propulsors from a CAD geometry. A set of parametric models were created using the software CATIA, which then became the basis from which the computational meshes were generated. Both the geometry and mesh models are parametric, leading to the possibility of generating a series of meshes with different geometric attributes and mesh distributions. In addition to that, a pre-fabricated geometry provided by a third party is imported and successfully meshed using the same method.

The produced meshes are tested in the UBEM code to verify that the meshes are useable and produce satisfactory results. The meshes were successfully imported and simulated, with results conforming to those expected from test data

It becomes clear that mesh creation for complex geometries is feasible using the proposed method in conjunction with CATIA. This opens up the possibility of increasing the complexity of geometries in future research with BEM. In addition, the ease of handling both

A further improvement of the method would be to program most manual tasks with scripts, to simplify and make the procedure quicker. In addition, the interface program could be expanded upon, adding features such as extracting a sparser mesh from a denser one.

With the method developed, any geometry that can be created or imported in CATIA can be manipulated and exported as a BEM mesh. Future designers can expand on the models presented, adding new geometries and configurations, or even create new geometries from scratch and export them as grids using the method as a tool. In addition, CATIA offers the capability of deforming existing geometries in a parametric way, which can be applied to any existing geometry.

Some proposals for future research are:

- Creation of propeller models of different series or original designs

- Expansion on the existing models by reusing them in assemblies. For example, models of ducted and rudder propeller can be easily be created with the propeller model, while the pod can be changed by exchanging the propeller for another model.

- Creation of other propulsor types or energy saving devices

- Deformation of existing surfaces in a parametric way to investigate optimal form

Part XI. Bibliography

Carlton, J. (2012). Marine Propellers and Propulsion, Elsevier Ltd.

Christoph M. Hoffmann, R. J.-A. (2002). "Parametric Modeling." Handbook of Computer Aided Geometric Design: 22.

Kogent, I. (2009). Catia V5 Essentials, Jones & Bartlett Learning, LLC.

Oosterveld, M. W. C. and P. van Oossanen (1975). "FURTHER COMPUTER-ANALYZED DAT OF THE WAGENINGEN B-SCREW SERIES." International Shipbuilding Progress **22**(251): 251-262.

Politis, G. K. (2004). "Simulation of unsteady motion of a propeller in a fluid including free wake modeling." Engineering Analysis with Boundary Elements **28**(6): 633-653.

Politis, G. K. (2011). "Application of a BEM time stepping algorithm in understanding complex unsteady propulsion hydrodynamic phenomena." Ocean Engineering **38**(4): 699-711.

Stroud, I. (2006). Boundary Representation Modelling Techniques, Springer London.

Stroud, I. and H. Nagy (2011). Solid Modelling and CAD Systems: How to Survive a CAD System, Springer London.

Tsarsitalidis, V. (2015). Hydrodynamic Simulation of Biological Propulsion Systems and Application on the Design of Optimal Marine Propulsors, National Technical University of Athens.

NX Nastran User's Guide

Azipod XO Brochure

CATIA User's Manual

CATIA: www.catia.com

Rhino3D: www.rhino3d.com

Solidworks: www.solidworks.com

Siemens NX: www.plm.automation.siemens.com/en_us/products/nx/

Inventor: www.autodesk.com/products/inventor/overview

Aknowledgements

I would like to express my deepest gratitude to my assisting supervisor, V. Tsarsitalidis, for his assistance, advice, patience and insight. Without him, the current thesis would be much different.

I would also like to thank my supervising professor, G. Politis, for giving me the opportunity for this research, despite the initial indications that it would not be viable.

Last but not least, my utmost gratitude to my mother and brother, Maria and Fanis, who stand by me all these years, and all the people (who are too numerous to list) who have supported me over the course of my studies.

Appendix 0: Introductory Information

In the appendix, the basic elements of the meshing method will be presented. The general methodology can be summed up in Fig. 0-1

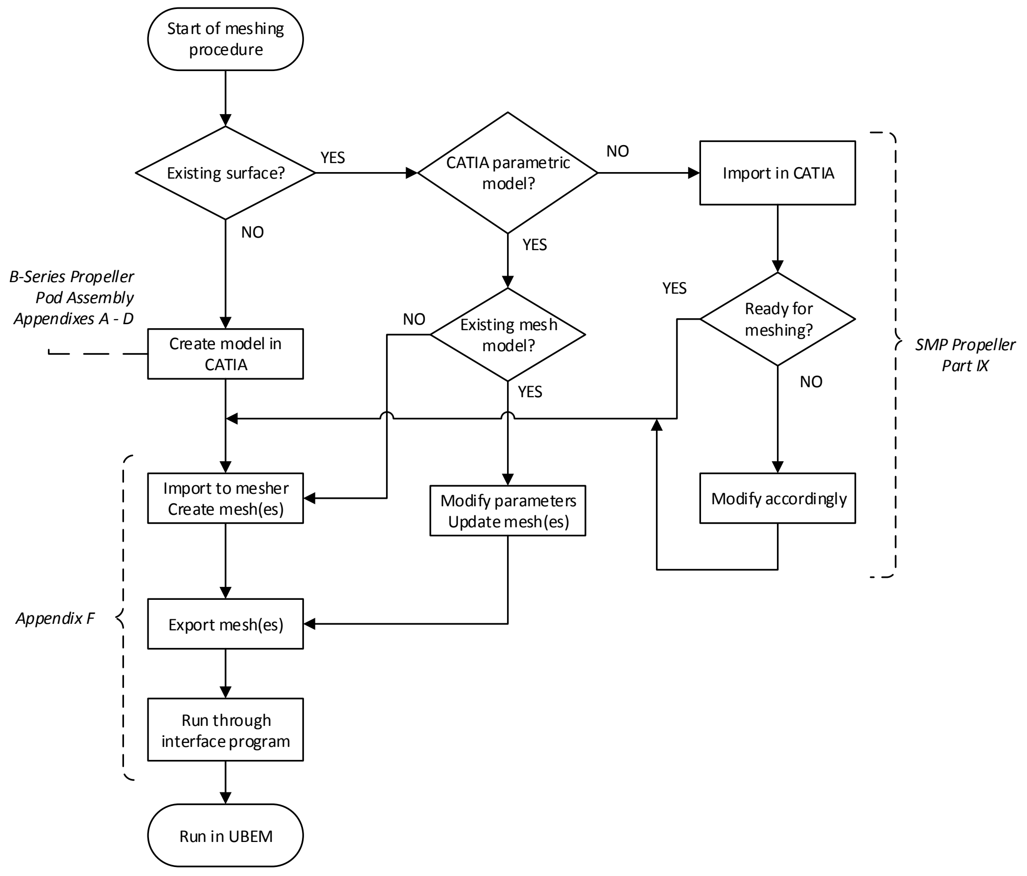


Fig. 0-1 Meshing methodology chart

For the creation of the meshes, as outlined by the 3 cases studied, the following exist:

In case of an existing surface: if there is both a parametric geometry and a mesh model, the mesh is modified only by changing the parameters. It can then be exported and translated for input in UBEM.

If there is only a parametric surface model, without a mesh model, the surface is input in the mesher workbench, the mesh model, created and then the mesh can be exported and translated as above.

In case of an existing surface that is not a CATIA surface (as is the case with the SMP propeller), it needs to be imported in CATIA, modified accordingly (split in segments and by bodies) and then input in the mesher to create the mesh model. The SMP model example can be found in **Part IX**

In case there are only offsets provided, the model must be built from scratch, creating the necessary parameters according to the design intent. A step-by-step example of a model based on offsets (B-Series) can be found in **Appendix A** with the parameter excel file described in **Appendix C**. The general overview of the model is contained in **Part VII**

In case the user is working from a concept design, the model must also be built from scratch, creating the necessary parameters according to the design intent. A step-by-step example of a model based on offsets (Podded propulsor) can be found in **Appendix A** with the parameter excel file described in **Appendix C**. The general overview of the model is contained in **Part VIII**.

Appendix A. Step-by-step creation of the B-series model

A-1. Blade

A-1.1. Blade Wireframe

Reference Lines

First, the propeller reference lines are defined.

The generator line [GL] is a straight **line** on the XZ plane that is at an angle <$\theta_{ip>}$with the Z axis. A set of **planes** offset from the XY plane, one for each radius, are created [section planes] and their **intersections** with the GL are located [GL points].

The pitch distribution is also drawn: each point of the pitch **spline** lies on the corresponding plane and is **offset** from the Z axis by the <pitch length>.

2D sections

Each flat section is created in a **sketch** supported by the respective plane. All offsets from TE and chord.

Pitch angles are calculated as following:

$$\theta_P = \tan^{-1}\left(\frac{p}{2\pi r}\right)$$

Where r is the respective radius, p is the respective pitch.

1. Draw **line** [chord] – impose **length constraint**<chord length>, turn into **construction element**
2. Draw **line perpendicular** to chord, passing through LE [LE perpendicular], turn into **construction element**.
3. Create two **splines** with N control points [Face] and [Back], N number of points as given for each section face. Impose offset constraints from TE and chord as needed. Note: LE control point coincides with LE perpendicular, TE control point coincides with TE on chord.
4. Impose **offset constraint** for GL point <length A>
5. Impose **angle constraint** between sketch's V-axis (global Y axis) and chord. Value as calculated from pitch angle equation.
6. Create **output** [chord output]
7. Exit sketcher
8. **Associate design table** values with appropriate constraints and update part

Helixes

Each helix starts at the GL intersection point with the respective plane, but will extend by <length1=a> and <length2=(c-a)> to each side, both lengths measured on curve.

Using the **helix** tool:

- Starting point : [GL point]
- Axis : X axis
- Pitch length : Pitch length for given radius
- Height:$< h = sin\theta_p(c - a) >$
- Orientation: clockwise (dependent on the propeller orientation)
- Starting angle:$< \varphi = \frac{-360\cdot\alpha\cdot sin\theta_p}{p} >$

3D sections

The 3D sections are constructed with the **Shape morphing** tool, by deforming the flat sections to follow the helical lines. For each section:

- Element to deform: create extract -> selected section side to deform (each side is deformed separately)
- Reference: respective output chord
- Type: reference point/curve
- Target: respective helix
- Constraint: point

Outlines

To create the outlines, the **Spline** tool is used. 4 splines are created: [LEface], [TEface], [LEback], [TEback].

1. Create **points** on 3D section curves start and end.
2. Interpolate points with **Spline** as needed (Fig. 0-4)

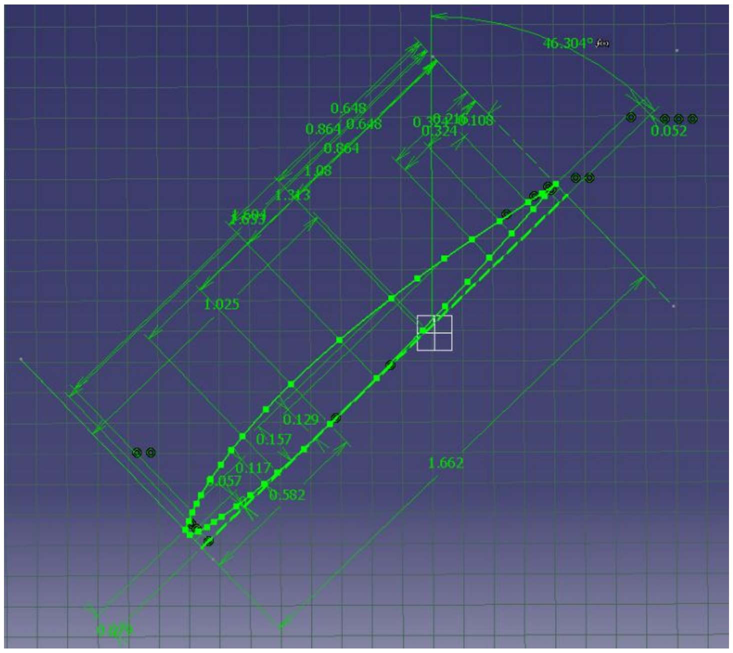
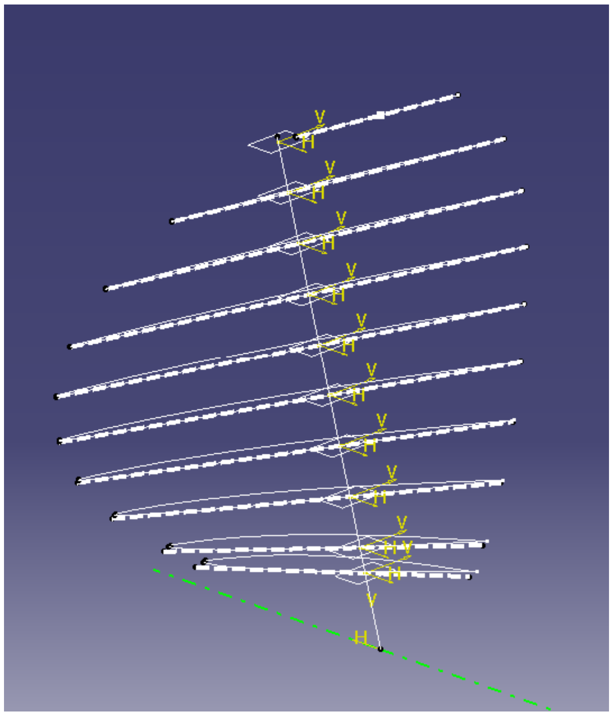


Fig. 0-1 Blade flat section definition with some of the constraints. The sketch is fully constrained, so the geometry is green


.Fig. 0-2 Blade Flat Sections with generator line and section planes– Dashed white lines are chord outputs, dashed green line is X axis

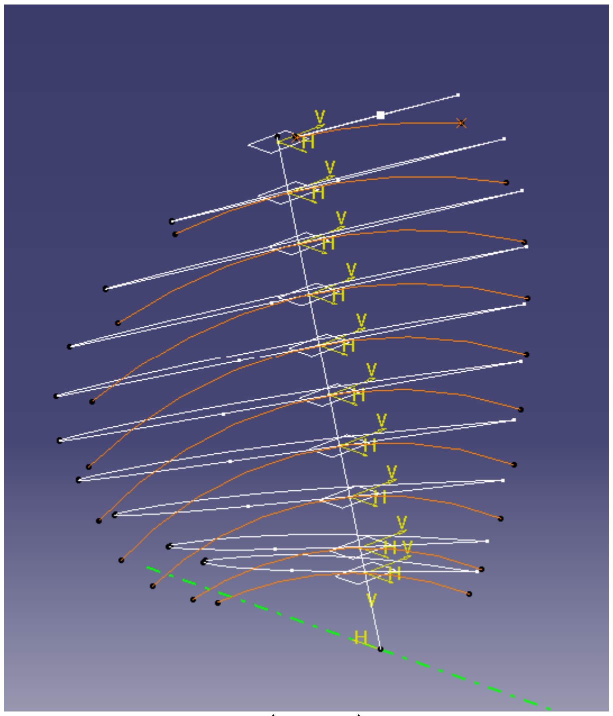


Fig. 0-3 Helixes (orange) and flat sections (white) with generator line and section planes. Dashed green line is X axis. The flat sections will be deformed according to the helixes.

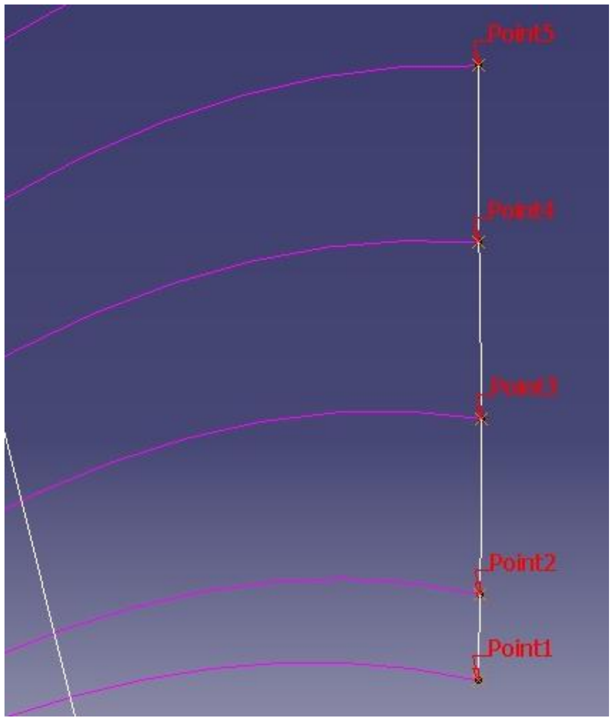


Fig. 0-4 Creation of outline spline by interpolating the endpoints of the cylindrical sections. Shown: sections 0.15R-0.5R

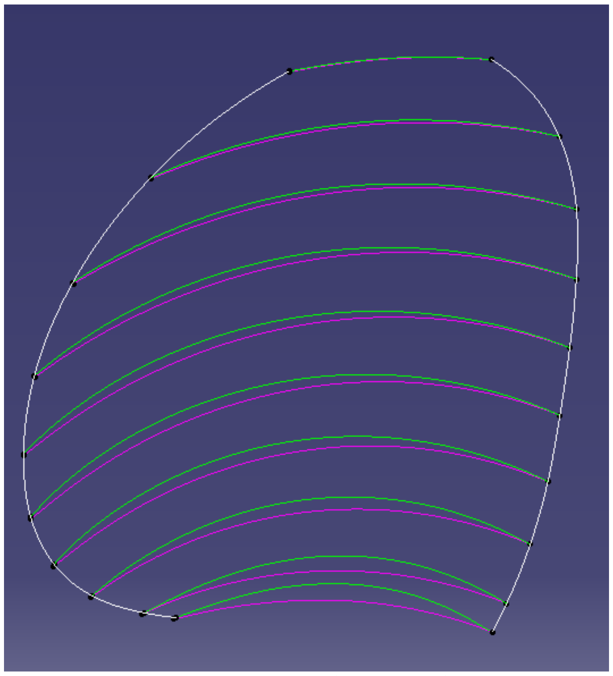


Fig. 0-5 Cylindrical Face (magenta) and Back (green) sections with outline (white)

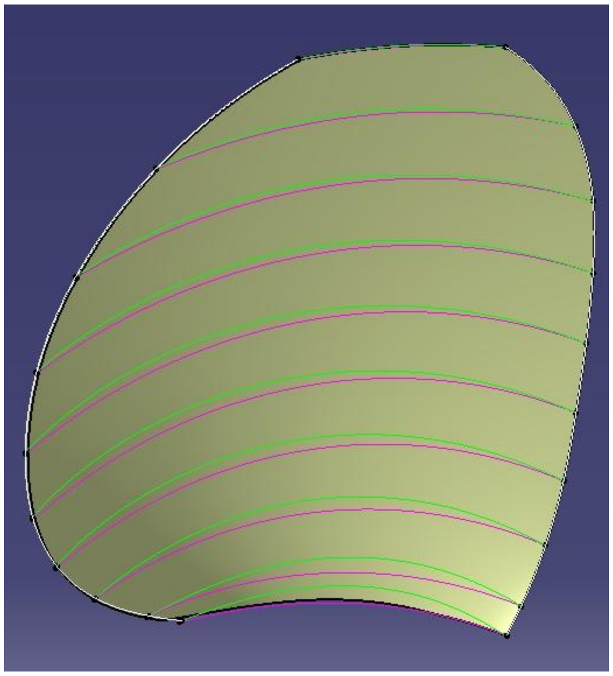


Fig. 0-6 Cylindrical sections and resulting blade surface

A-1.2. Blade surface

Face and Back surface

Working in the **Freestyle workbench**.

1. To create the blade face surfaces, the **Net Surface** tool is used for each side, [Face] and [Back]:

   - Guides: the deformed curves for the side being created. Set section 0.7R as the driving guide.
   - Profiles: select the two outline splines. Set TE outline as the driving profile
   - Surface [Face_srfc] and [Back_srfc] are created.

Return to **Generative Shape Design workbench**

2. Create **cylinder** with radius r=0.155R
3. Trim Face and Back surfaces with cylinder (necessary for more reliable model updates)

Leading Edge Fillet

First, a closing strip will be created in the Leading Edge:

1. Create **Extracts** for the LE edge of both Face and Back surfaces
2. Create **points** on start, mid, 0.75 of length and end of each extract. Create **line** between each face point with respective back point, resulting in 4 lines.
3. Create **Multisections Surface**[LE strip] using lines from step two as sections and extracts from step1 as guides.
4. Join LE strip, Face and Back surfaces using **Healing.** (Fig. 0-7)

The circle in the LE is finally created with a **Tritangent Fillet,** on the Healing from step4, with the following settings [blade_srfc] (Fig. 0-8):

- Extremities: smooth
- Faces to fillet: [Face_srfc; Back_srfc faces]
- Face to remove: LE strip
- Trim support is selected
- No limiting element

As it can be seen in Fig. 0-8, the tritangent fillet leaves a raised edge at the tip. This will be fixed further on, when preparing the surfaces for meshing.

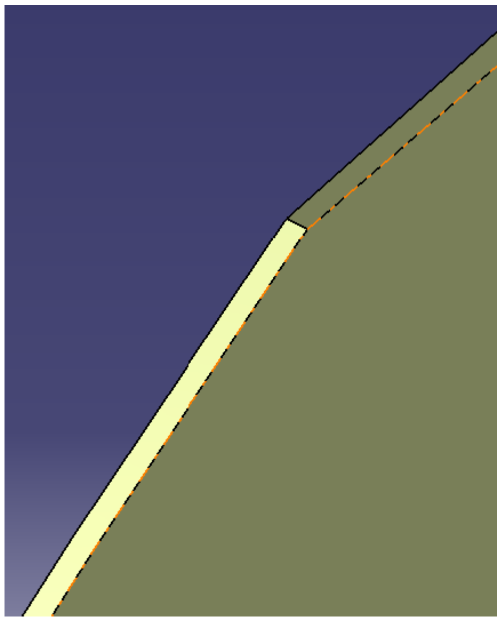


Fig. 0-7 LE strip detail at propeller tip before fillet

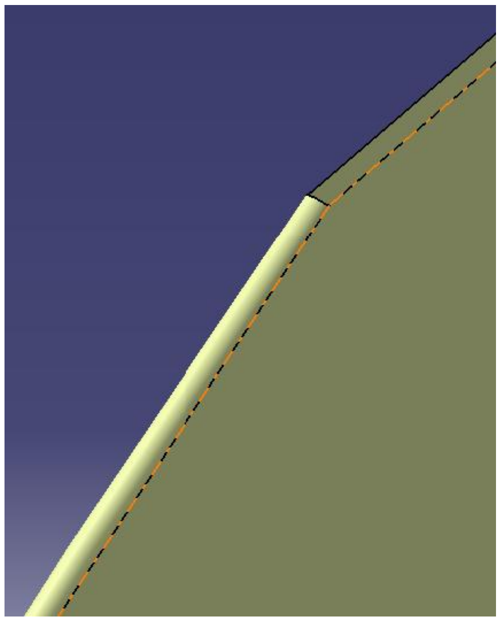


Fig. 0-8 Detail at LE propeller tip after tritangent fillet. The tip will be rounded off in further steps

A-2. The hub

The following method is proposed:

1. Create a**2D sketch** on [XZ plane] for the [hub profile]. It is essentially a straight line at an angle < θ_{hub} > with the X-axis. Impose **constraints**: horizontal length (should be enough to support the blade), LE distance from X axis (essentially the hub's radius), angle with X axis, edge offset from GL. Exit sketcher
2. Create a **Helix** starting from the hub profile's edge and tapering outwards [hub helix]:
   - Starting point : hub profile's LE
   - Axis : X axis
   - Pitch length : Pitch length for 0.2r
   - Height: hub's length
   - Orientation: dependent on the propeller orientation – clockwise for right-handed propellers
   - Starting angle: 0
   - Radius variation: select hub profile
3. Revolve hub helix around X axis to create [hub_srfc]. Total angle should be less than 360^{o}, but the surface created should be able to support at least two blades (Fig. 0-9).

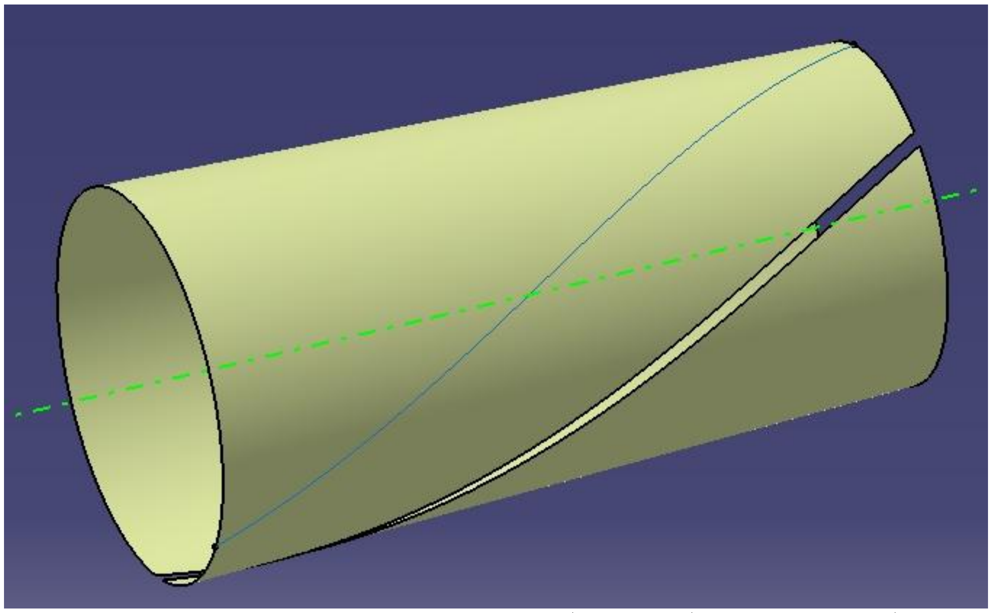


Fig. 0-9 Hub surface: created by revolving the helix (light blue) around X axis (dashed green)

A-3. Root Fillet

The root fillet is created as a dress-up feature. Before creating it, the blade and the hub surfaces need to be joined into one object.

Trim [blade_srfc; hub_srfc]. Pick the appropriate surface parts to be kept to create the [Blade_Hub_Trim].

<u>Spines</u>

1. Create a cylinder with radius 0.2r around X axis, either by **extruding** a circular profile, or by using the **cylinder** command.
2. Create **intersection** between the cylinder and blade [Fillet Edge]
3. Create point exactly on intersection TE [TE Fillet Point]
4. Create 3 **extracts** of the intersection: [Face Fillet Edge], [LE Fillet Edge], [Back Fillet Edge].
5. On the Back and Face fillet edges, create 5 **repetition points and planes**on each. Also create a point and normal plane at 0.1 from each extract's TE [limiting plane] and another pair exactly on the LE edge. There shouldn't be a point on the curves' Trailing edges.
6. On the LE extract, create 3 **points and normal planes pairs**: one on each edge and one at the midpoint of the curve.
7. Using the planes created in steps 4 and 5, create 3 **Spines**: [Face spine], [LE spine], [Back spine]. The Face and Back spines should go up to the limiting planes (Fig. 0-10).

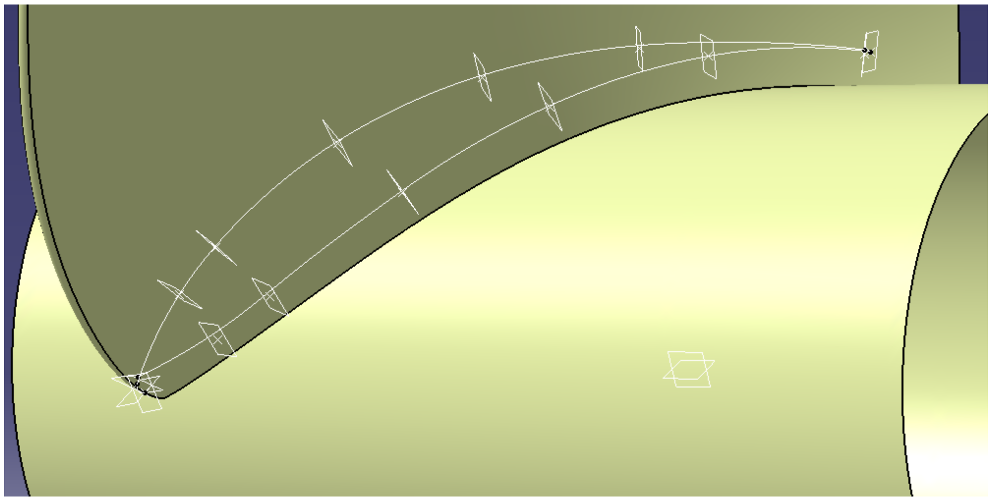


Fig. 0-10 Points, planes and spines used in the root fillet creation. The points are created on the fillet's upper edge at 0.2R, not currently shown. The planes pass from the points and are vertical to the fillet's edge. The spines are defined by the planes, using the "spine" tool.

<u>**Fillet**</u>

The main fillet surface is created with the **Face-Face Fillet** tool in 3 parts for Face, Leading Edge and Back. For each face on the blade, create a face-face fillet with the hub's face:

- Support: [Blade_Hub_Trim]
- Extremities: smooth
- Faces to fillet: 2 elements (select as needed)
- Trim support: no
- Under the more>> menu:
- Hold curve: respective Fillet Edge curve from step 3
- Spine: respective Spine from step 6.

Three different fillet surfaces are created [Face_fillet], [LE_fillet], [Back_fillet] (Fig. 0-11)

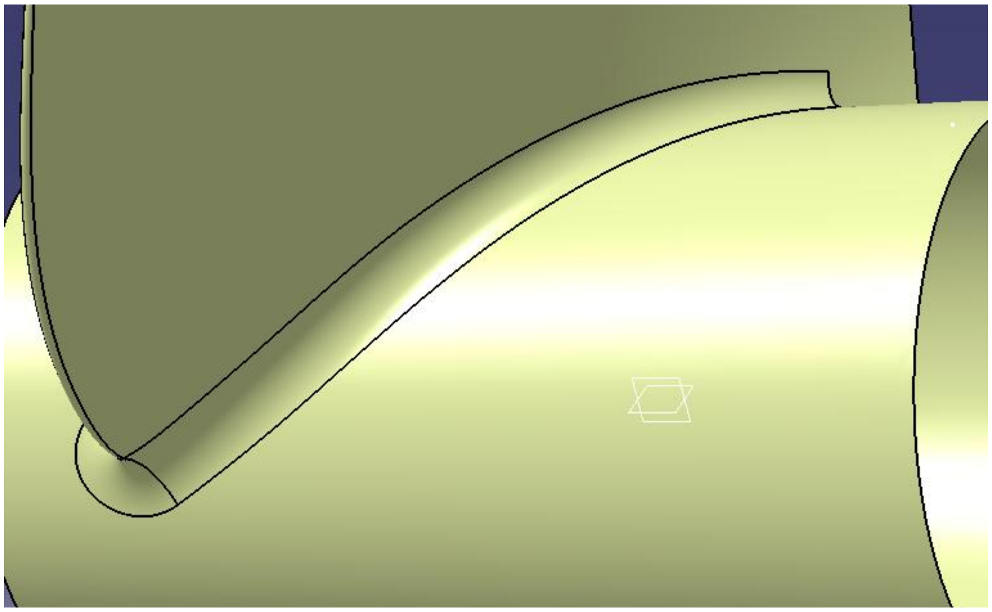


Fig. 0-11 Blade face and LE fillets, as created with the Face-to-face fillet tool. The TE portion will be added in next steps

To create the TE portion of the fillet:

1. Create a plane tangent to the blade's TE near the fillet area [TE plane]:
    a. Create **line** from TE Fillet Point tangent to blade TE edge
    b. Create two **lines** tangent to Fillet Edge on each side, passing from TE Fillet Point
    c. Create bisecting **line** to lines in b
    d. Define a **plane** using lines from a and c
2. On TE plane create a sketch [TE arc]:
    a. **Project 3D elements**: Blade TE. Make construction element.
    b. Intersect 3D elements: Hub Surface. Make construction element.
    c. Create circular **arc, tangent** to a and b.
    d. Impose **constraint** on upper arc edge: **coincidence** with TE Fillet Point
    e. Impose **constraint** on lower arc edge: **coincidence** with b. Create **output** of point [TE Fillet Point 2]
    f. Exit sketcher
3. Create [Face FilletCurve] by **intersecting** with Face limiting plane.
4. Create **spline** on hub (geometry on support: [Blade_Hub_Trim]), Face side, that coincides with lower fillet edge and continues to TE Fillet Point [Fillet Spline].
5. Switch to Freestyle workbench.
6. Create Net Surface:

    - Guides: [Face Fillet Curve; Fillet Spline]
    - Profiles: [TE arc; Face Fillet Curve]
7. Repeat steps 3-6 for Back side

Join the Face, LE and Back Fillets with the TE portions using **Healing.** The Fillet surface is complete [Root_Fillet].

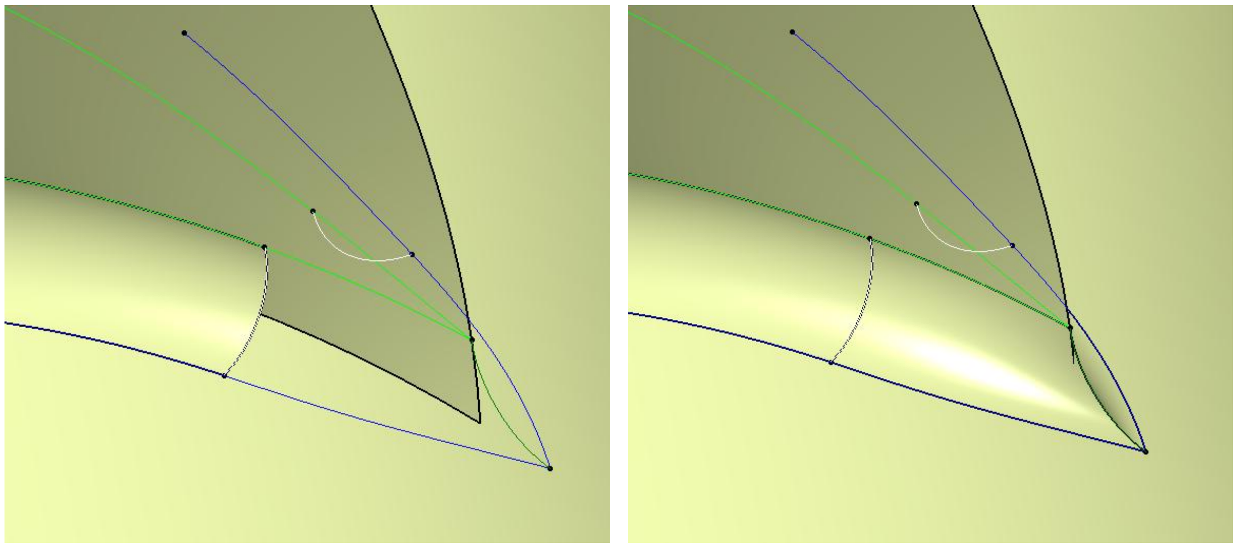


Fig. 0-12 TE root fillet's wireframe (left) and resulting surface (right)

A-4. The hub caps

The Boundary Element Method used requires that the body to be input doesn't have big openings. In the propeller model this is achieved using a pair of caps at each end of the hub. The caps are spherical and have a small circular hole, centred at their point of intersection with the X-axis. They are constructed as follows:

1. Create **sketch** (support: ZX plane) [sketch_front_cap]
   Intersect3D elements: [hub_srfc] – turninto construction element [mark1] (will use only upper intersection)
   Create **Arc:** centre on H-axis **(coincidence),** point1 coincidence with mark_1, point2 coincidence with H axis.
   Impose constraint: tangency [Arc; mark_1] (Fig. 0-13)
2. **Revolute:** [sketch_front_cap] around X axis [front_cap]
3. Repeat steps 1-2 for [rear_cap]
4. **Circle** on YZ plane [hole_circle]. Radius= <0.005*D>
5. **Extrude:** Profile: hole_circle
   Direction: X axis
   Limit1: up to element: front_cap, Limit2: up to element: rear_cap [hole_cylinder]

6. **Join:** [hub_srfc; front_cap; rear_cap] [hub_with_caps]

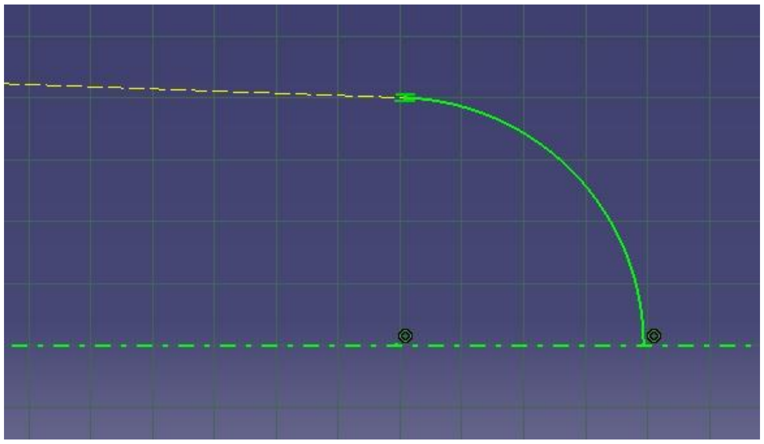


Fig. 0-13 Tangent arc (green) at cap with the hub intersection (dashed yellow). The cap surface will be created by revolving the sketch around X-axis

A-5. Preparing surfaces for meshing

The model mesh is segmented into 3 surface groups: blade, fillet and hub.

The UBEM program requires each surface to be defined in a Face-Back type, so the surfaces need to be split accordingly. This is done in the surface design workbench, before switching over to the mesher. The blade and root fillet surfaces are split along the Leading Edge, into a Face and Back portion. The hub is split transversely, with planes parallel to YZ that pass from the front and rear edges of the fillet. This creates 3 parts: the front, mid and rear. The mid of the hub is split longitudinally, using a spline, that is created roughly in the middle between two fillet edges. The front and rear parts are split in two symmetric portions using planes.

A-5.1. Blade Surface

The blade surface is segmented into a Face and a Back surface as follows:

1. **Split** the Blade_Hub_Trimusing the 0.2R cylinder, keeping the upper part [Blade Surface]
2. Create **extract** of blade's LE round fillet face [LE face extract]
3. Locate midpoint of LE Fillet's Edge [LE point]
4. Create **isoparametric curve** on the LE face extract so that it splits the extract in middle [LE isoparametric](Fig. 0-14)
    - Support: LE face extract
    - Point: LE point
    - Direction: pick suitable direction
5. **Extrude**the1.0r helix by the X direction and use it to **split** LE face extract, keeping the lower part.

6. **Split** trimmed from (5) using the LE isoparametric curve, keeping both sides [LE Face], [LE Back] (Fig. 0-15)
7. **Extract** faces on Face and Back
8. Join with **healing** the Face extract with LE Face and the Back extract with LE Back [Face2], [Back2]
9. Create 2 edge **extracts** on the Face2 and Back2 tip edges.
10. Create point at the tip extracts **intersection**

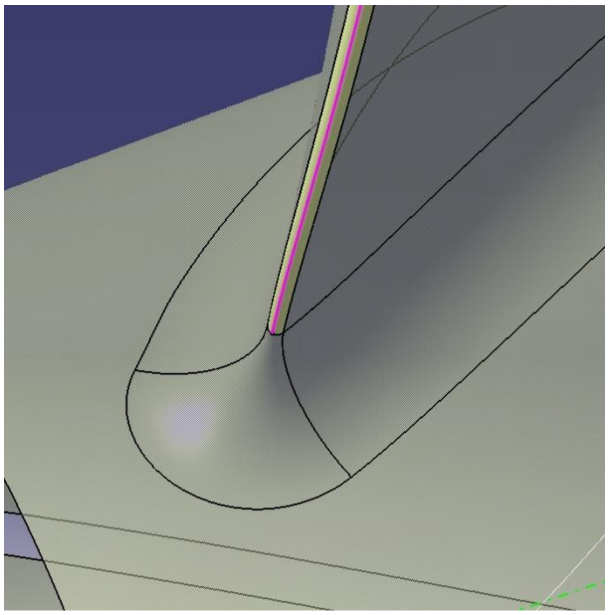


Fig. 0-14 LE extract and isoparametric curve used to split the Leading Edge fillet (magenta). Root fillet, hub and rest of the blade are slightly transparent

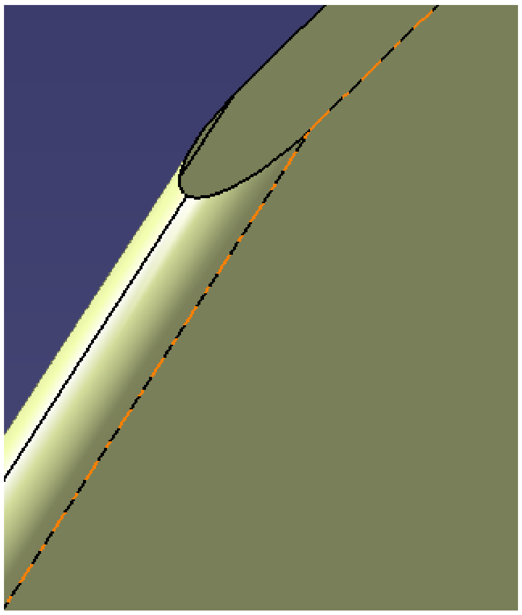


Fig. 0-15 Leading edge near tip, after split into its face and back portion and rounded off.

A-5.2. Fillet and Hub

The fillet surface is split in a way similar to the blade, resulting in 2 surfaces, Fillet Face and Fillet Back.

The hub is split transversely at the extreme edges of the fillet in the front and rear, by two planes parallel to YZ, resulting in 3 pieces: front, mid and rear. The mid portion is split longitudinally in two parts, upper and lower, with the upper being the one closest to the fillet Face edge. The front and rear are similarly split in two longitudinally, using a rotation of the XZ plane that passes from the two fillet edge extremums. The final hub pieces to be meshed are the 6: Hub Upper Front, Mid and Rear and Hub Lower Front, Mid and Rear.

<u>Filet split</u>

1. Create **Boundary** of the lower fillet edge
2. **Project** boundary on Hub

3. Find the two**extremums**of the projected curve along the X axis [LE extremum], [TE extremum]
4. Create **line** tangentto LE isoparametric curve at LE point
5. Create **line** between LE point and LE extremum
6. Create **plane** from lines in (4) and (5)
7. **Intersect** Fillet with plane from (6), keep Front edge with **Near**
8. Extract TE arc
9. **Connect** (7) and (8) – leave Trim Elements unchecked
10. **Join** (7), (8) and (9) together [Splitting Join]
11. **Split** Fillet using Splitting Join. Keep both sides [Face Fillet], [Back Fillet])

<u>**Hub mid**</u>

1. **Split** projected from (Filet spit -2) using (Fillet split-10) as cutting element. Keep both sides [Face Fillet Projected], [Back Fillet Projected]
2. Create **Circular pattern**[Back Pattern]
   Parameters: Instance & angular spacing
   Instances: 2
   Angular spacing: <360deg/Z>.
   Reference Element: X axis
   Object to Pattern: Back Fillet Projected
3. **Project** Back Pattern on Hub
4. Create two planes parallel to YZ plane, passing from the extremum points [Front Cutting Plane], [Rear Cutting Plane]
5. **Split** Hub with Front and Rear Cutting Planes. Keep the section between them.
6. **Split** (5) using Face Fillet Projected and projected Back Pattern as cutting elements. Keep the section between them [Split Hub]
7. Create a set of 5 planes between [Front; Rear Cutting plane] using **Planes between**
8. **Intersect** set of planes from (7) with Split Hub [Intersection set]
9. Find **midpoint** for each curve in Intersection set.
10. Create **extract** of Front and Rear edges of Split Hub[Front Hub Extract], [Rear Hub Extract] and find midpoints
11. Create spline interpolating points from (9) and (10) [Long. Hub Splitting spline]
12. Split the Split Hub using Long. Hub Splitting spline. Keep both sides. [Hub_mid_upper], [Hub_mid_Lower]

<u>**Front and Rear Hub**</u>

1. Split [hub_with_caps] with [Front Cutting Plane]. Keep front portion
2. Create **Line**:
   Line type: Point-point
   Point1: LE extremum
   Point2: intersect: [X-axis, Front Cutting Plane]
3. Create Plane:
   Plane type: through two lines: [Line from (2); X-axis] [F_long_cutting_p1]
4. **Revolve** F_long_cutting_p1around X axis by <360deg/Z> [F_long_cutting_p2]

5. **Revolve** F_long_cutting_p1 around X axis by <360deg/Z/2> [F_long_cutting_p3]
6. Split (1) with (3), (4) and [hole_cylinder]. Keep surface between
7. Split (6) with (5), keep both sides [Front_hub_upper], [Front_hub_lower]
8. Repeat (1)-(7) for rear [Rear_hub_upper], [Rear_hub_lower]

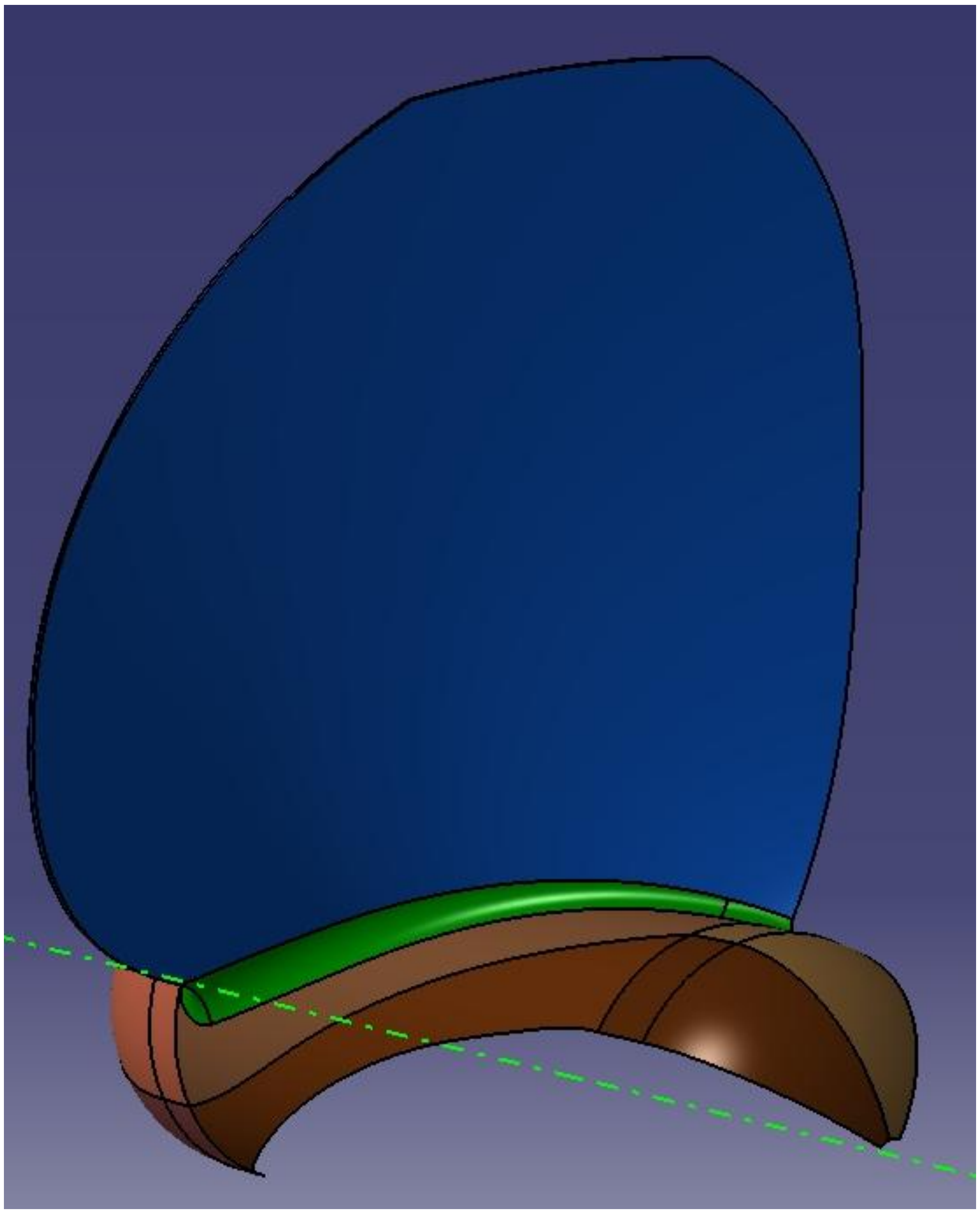


Fig. 0-16 The surfaces to be meshed (with shades of blue are the blade surfaces, shades of green the root fillet, shades of brown the hub surfaces)

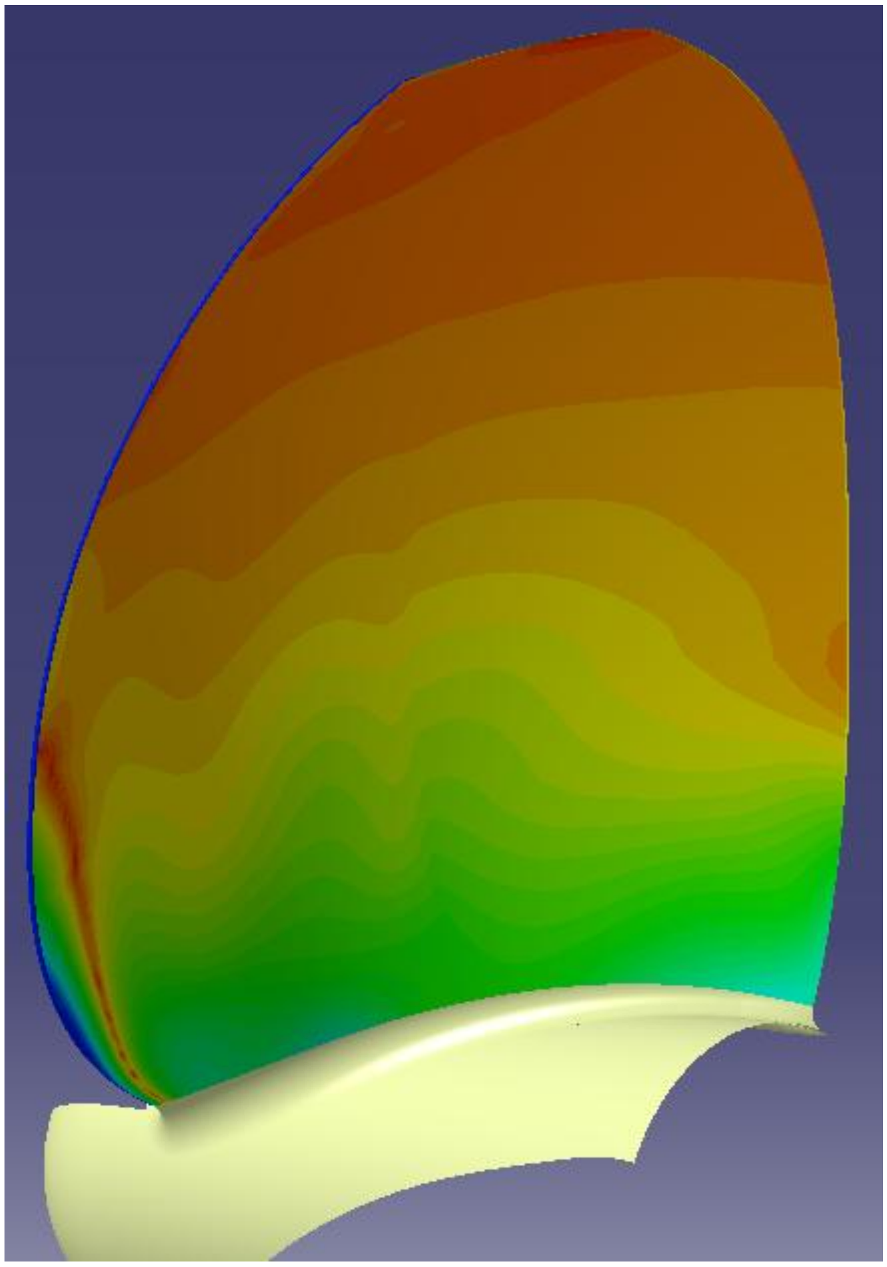


Fig. 0-17 Gaussian-style qualitative surfacic curvature analysis on the propeller blade. Red is minimum, blue is maximum

Appendix B. Step-by-step creation of the podded propulsor model

B-1. **Strut**

B-1.1. Strut Wireframe

1. Create **sketch**: (support: XZ plane) [strut_outline]
   Upper strut:

   - **Rectangle**: height= <h1>, length = <ls1>
   - **Constraint: offset** [rectangle edge; sketch V-axis] = <offset1>
   - **Intersect 3D elements**: pod – make construction element [mark_s1]
   - **Constraints: offset** [rectangle lower edge; upper mark_s1 line] = <h2>
   - **Splines:** interpolate points on vertical rectangle edges. Start and end points of splines coincide with rectangle corners. [spline_front], [spline_rear]
   - Lower strut:
   - **Line:** horizontal [line_s1]
   - **Constraint: offset** [line_s1; lower rectangle edge] = <h2*1.5>
   - Create two **lines,** starting from each of the lower rectangle corners [line_front], [line_rear]
   - **Constraint: angle** [line_front; line_s1], [line_rear; line_s1] =<θstrut_front>, <θstrut_rear> the front and rear angle respectively.
   - Create **output** of the following: spline_Front, spline_rear, line_front, line_rear, two lowest corners of rectangle (LE_output, TE_output), lower edges of lines s2 and s3 (LE_lower_output), (TE_lower_output)

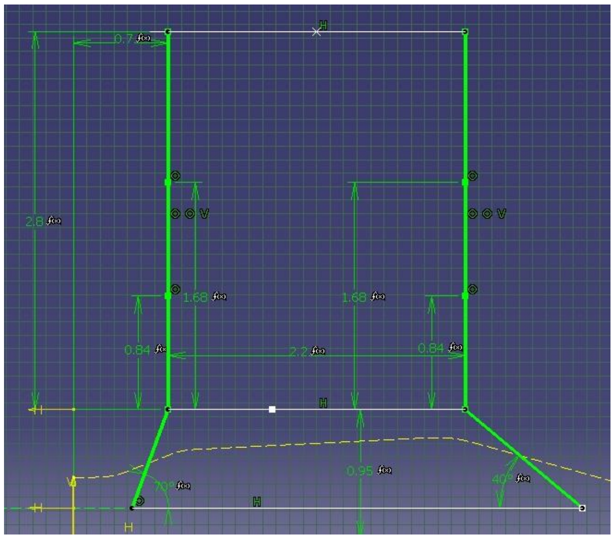
Note that the rectangle is created to aid with keeping the splines straight. In case the outline is not straight, the coincidence constraints of the spline points with the lines can be deactivated and the points moved freely in the desirable configuration. If the outline is to remain straight, the splines can be omitted. In this case, outputs of the vertical rectangle edges will be created instead (the vertical lines will be toggled from construction elements to standard ones). In addition, the verticality constraint on the lines can be deactivated and the lines rotated freely to create a slanted outline.

As an example, the variants in Fig. 0-19, Fig. 0-20 can be created.

2. Create planes, parallel through point:
   Reference: XY plane
   Point: LE_output [plane_u_foil] – LE_lower_output [plane_l_foil]
3. Create sketch: (support: plane_u_foil) [foil_upper] (Fig. 0-21)

   - **Line:** endpoints:[LE_output, TE_output] – make construction element [chord_1]
   - **Line** throughTE_output. **Constraint**: perpendicular to chord_1 [TE_perp_1]
   - **Point**. **Offset constraints**: from LE = <max_t_sp*ls1>
   - from chord_1 = <max_t*ls1> [max_t1]

- **Point** on TE_perp. **Offset constraint**: from chord =<te_t*ls1> [te_t1]
- **Line**: endpoints:[max_t1;te_t1] [line_uf1]
- **Spline**: interpolate [LE_output; max_t1; te_t1]
- **Tangency** on LE_output: **constraint**: perpendicular to chord_1
- **Tangency** on max_t1 point: **constraint**: parallel to chord_1
- **Tangency** onte_t1: **constraint**: parallel to line_uf1
- Create **output**: max_t1 [outp_maxt1]



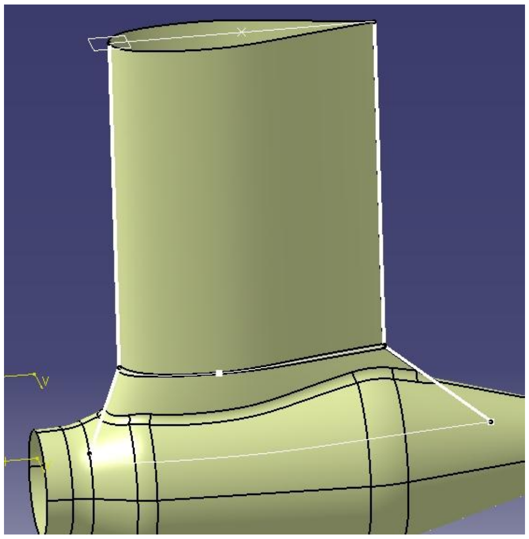


Fig. 0-18 Left: Outline in sketcher, with its constraints. In yellow dashed curve, the pod intersection with the sketch plane. Right: the resulting strut

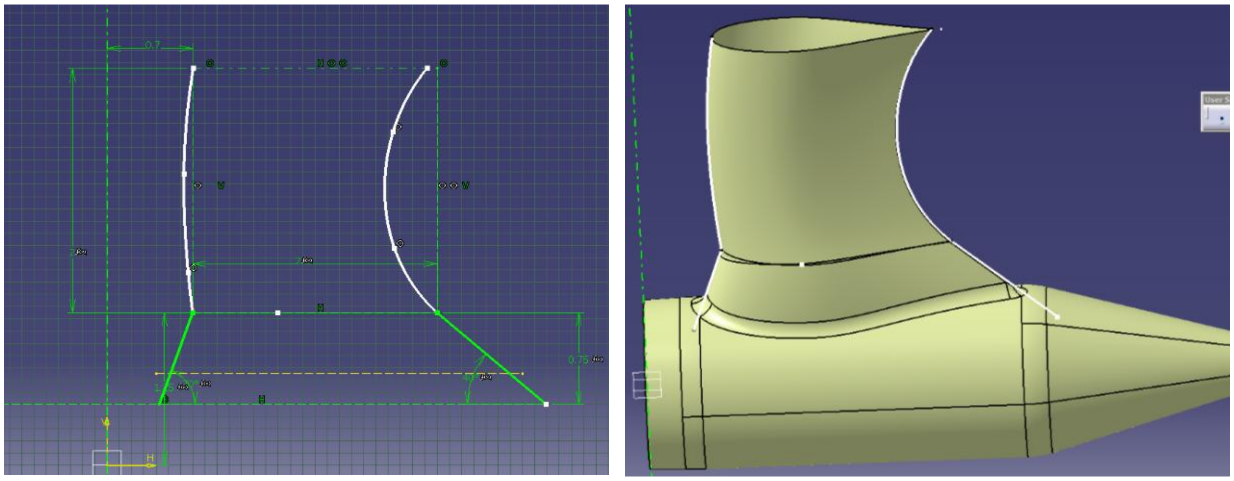


Fig. 0-19 Outline Variant - Removed Coincidence Constraint of Spline Control Points to create curved outline

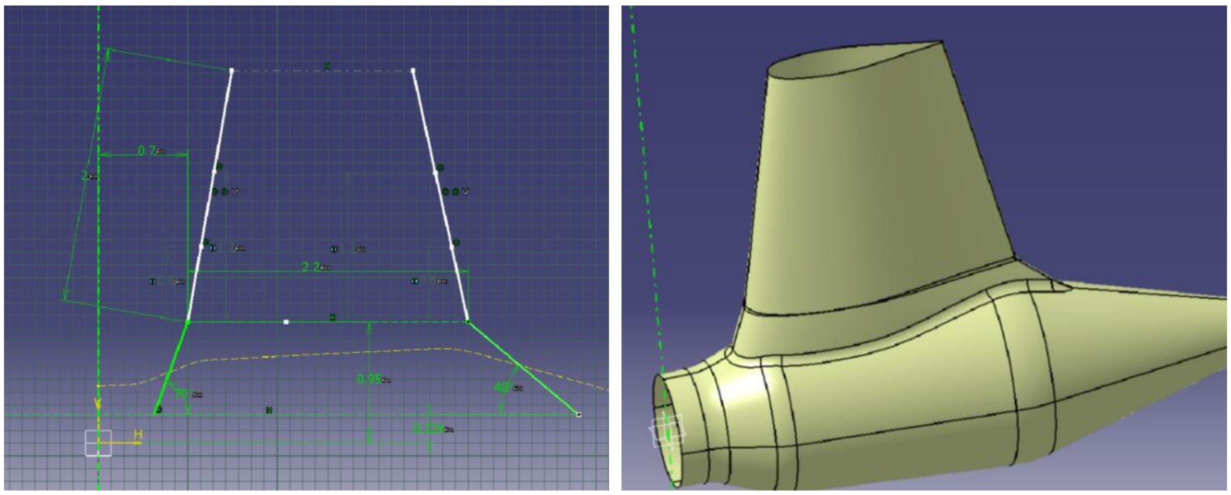


Fig. 0-20 Outline Variant - Removed Verticality Constraint of Lines to created slanted outline.

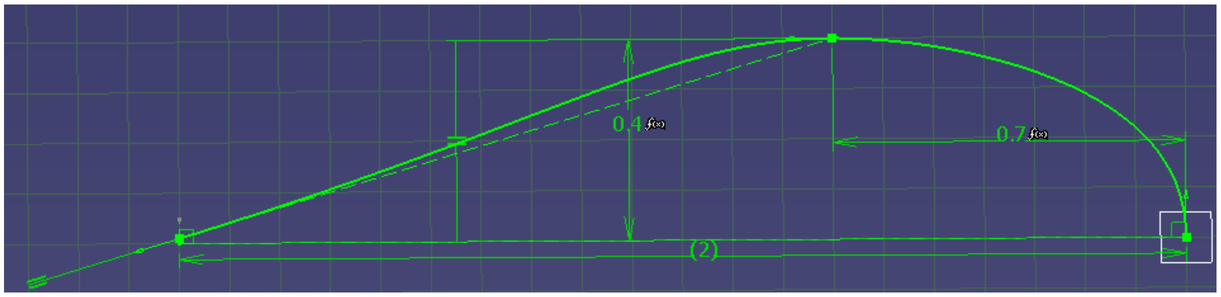


Fig. 0-21 Foil_upper section sketch with constraints

4. Project outp_maxt1 to plane_u_foil [max_t2]
5. Create sketch: (support: plane_l_foil) [foil_lower]

   - **Line**: endpoints:[LE_lower_output; TE_lower_output] – make construction element [chord_2]
   - **Line** throughTE_lower_output. **Constraint**: perpendicular to chord_2 [TE_perp_2]
   - **Point** on TE_perp. **Offset constraint**: from chord =<te_t*ls1> [te_t2]
   - **Line**: endpoints:[max_t2;te_t2] [line_uf2]
   - **Spline**: interpolate [LE_lower_output; max_t2; te_t2]
   - **Tangency** on LE_lower_output: **constraint**: perpendicular to chord_2
   - **Tangency** on max_t2 point: **constraint**: parallel to chord_2
   - **Tangency** onte_t2: **constraint**: parallel to line_uf2
6. **Plane**: parallel through point [plane_s1]
   Reference: XZ plane; point:LE_output
7. **Project**: [spline_rear; support: plane_s1] [proj_upper]
   [line_rear;support: plane_s1] [proj_lower]
8. **Line**: endpoints:[max_t2;outp_maxt1] [line_maxt]

B-1.2. Strut Surface

1. **Multi-sections surface**: [section: foil_upper; Guides: spline_front, Proj_upper] [upper_strut]
2. **Multi-sections surface**: [sections: foil_upper, foil_lower; Guides: line_front;proj_lower] [lower_strut]

Switch back to Generative shape design

3. **Join** [upper_strut; lower_strut] [join_s1]
4. **Symmetry**: element: join_s1, reference: XZ plane [sym_s1]
5. **Extract**: [rear edge of upper_strut] [extr_u]
   [rear edge of lower_strut] [extr_l]
6. **Extrude**: profile: extr_u [upper_TE]
   Direction: Y-axis
   Limit1: up-to element [join_s1], Limit2=0
7. **Extrude**: profile: extr_l [lower_TE]
   Direction: Y-axis
   Limit1: up-to element [join_s1], Limit2=0
8. **Join**: [join_s1;sym_s1;upper_TE;lower_TE] [join_s2]
9. **Tri-tangent fillet**: Support: join_s2 [tri_fi1]
   Faces to fillet: upper_strut faces
   Faces to remove: upper_TE face
10. **Tri-tangent fillet**: Support: tri_fi1 [tri_fi2]
    Faces to fillet: lower_strut faces
    Faces to remove: lower_TE face
11. **Edge Fillet**: Object to fillet: tri_fi2 [strut]
    Radius= <rfstrut>
    Edge: join between upper and lower strut

The resulting suface can be seen in Fig. 0-18

B-2. **Pod**

B-2.1. Pod Wireframe

1. Create **planes** offset from YZ by <lp1> [plane_p1], offset from plane_p1 <lp2> [plane_p2], offset from plane_p2 by <lp3> [plane_p3], offset from plane_p3 by <lp4> [plane_p4], offset from plane_p4 by <lp5> [plane_p5]
2. Create **circle** on plane [YZ] [circle_p1]
    - Centre: Intersection [YZ plane; X axis]
    - Diameter = <external parameter: hub diameter on touching edge>
3. Create **line** [line_p1]:
    - Angle/normal to curve
    - Support: ZX plane
    - Angle: external parameter: <θhub>
    - Point: create extremum: on circle_p1, max on Z axis
    - Up-to: YZ plane
4. Create circle on plane_p1. [circle_p2]
    - Circle type: centre and point
    - Centre: intersect [plane_p1; X axis]
    - Point: end of line_p1
5. Create circles on planes [plane_p3; plane_p4; plane_p5], centre is the intersection of each plane with X axis. Diameters are <d1> [circle_p3], <d2> [circle_p4] and <d3> [circle_p5]

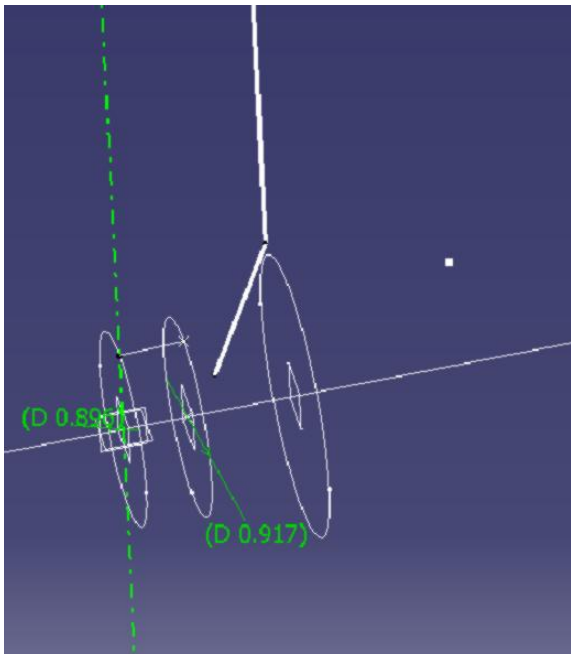
B-2.2. Pod Surface

To create each part of the pod surface, create a series of **blends,** blending between two consecutive circles. For closing points, create a max-extremum on Z axis

Example:

1. **Blend** [blend1]
   First curve: circle_p1
   First support: No selection
   Second Curve: circle_p2
   Second Support: No selection
   Under the closing points tab:
   First closing point: extremum (Max) of first curve on the Z axis
   Second closing point: extremum (Max) of second curve on the Z axis

   Repeat for rest of circles [blend2; blend3; blend4]



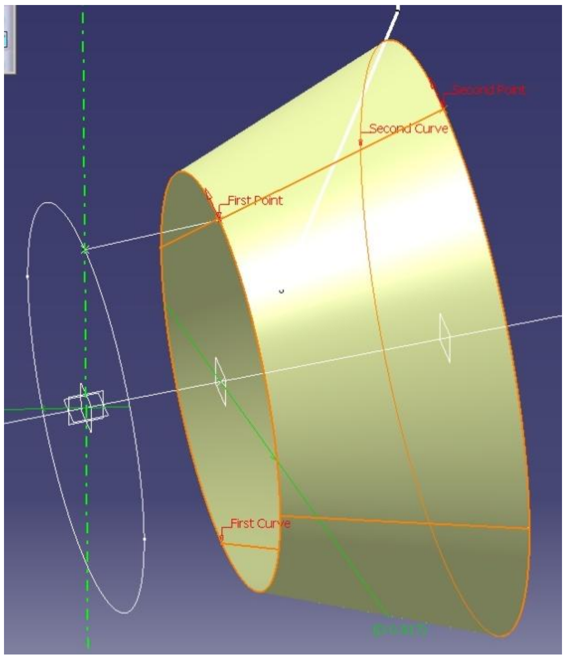


Fig. 0-22 Left: Wireframe front circles and X axis with strut outline front

Right: Blend2 (blend between second and third circle)

2. **Join** [blend1+blend2+blend3+blend4][Join_p1]
3. **Edge Fillet** on Join_p1 [fillet_p1]
   Radius: <rfp1>
4. **Edge Fillet** on fillet_p1 [fillet_p2]
   Radius: <rfp2>
5. **Edge Fillet** on fillet_p2 [fillet_p3]
   Radius: <rfp3>
6. Create **sketch** (support: ZX plane) [sketch_p1]
   - **Intersect3D elements:** [fillet_p3] – into construction element [mark_p1] (will use only upper intersection)
   - Create **Arc:** centre on H-axis **(coincidence constraint),** point1 coincidence with mark_p1, point2 **coincidence** with H axis.
   - Impose constraint: tangency [Arc – mark_p1]
7. **Revolute:** sketch_p1 around X axis [rear_cap]
8. **Circle** on plane_p5 [hole_circle]. Diameter= <d2*0.1>
9. **Extrude:** Profile: hole_circle
   Direction: X axis
   Limit1: up to element: rear_cap, Limit2: dimension = 0 [rear_hole]
10. **Split:** Element to cut: rear_cap
    Cutting element: rear_hole
    Keep cap part [cap_with_hole]
11. **Join:** [fillet_p3 ; cap_with_hole] [pod]

B-2.3. Pod and Strut Joined

1. Trim [strut; pod] [trim1]
2. Edge fillet: object to fillet: trim1 [propulsor_body]
   Radius = <rfpod>

B-2.4. Pod with Fin

The creation of the fin is similar to the creation of the upper portion of the strut, so the step-by-step procedure will not be repeated. The fin is joined to the pod model the same way as the strut is joined to the pod (see B-2.3)

B-3. **Meshing preparation**

For use in mesh creation without fins:

1. **Split**: Element to cut: propulsor body, cutting elements: XZ plane – Keep right side [bod_right]
2. **Extract**: lower edge of fillet between strut and pod (tangent continuity) [extr_f1]
3. **Planes:** parallel through point: Reference: YZ plane, points on either end of extr_f1 [plane_m1], [plane_m2]
4. **Split**: element to cut: bod_right, cutting elements: extr_f1
   Keep both sides [strut_r], [pod_r1]
5. **Consecutive Splits**: pod_r1 with planes plane_m1, plane_m2 and plane_p2 (at the cap) as necessary to create 4 surfaces: front, mid and rear pod and cap as per fig.13.
6. Repeat steps 1-5 for left side

For use in mesh creation with fins, the pod body is split around the fin(s) accordingly. The fins are split with their planes of symmetry, keeping both sides.

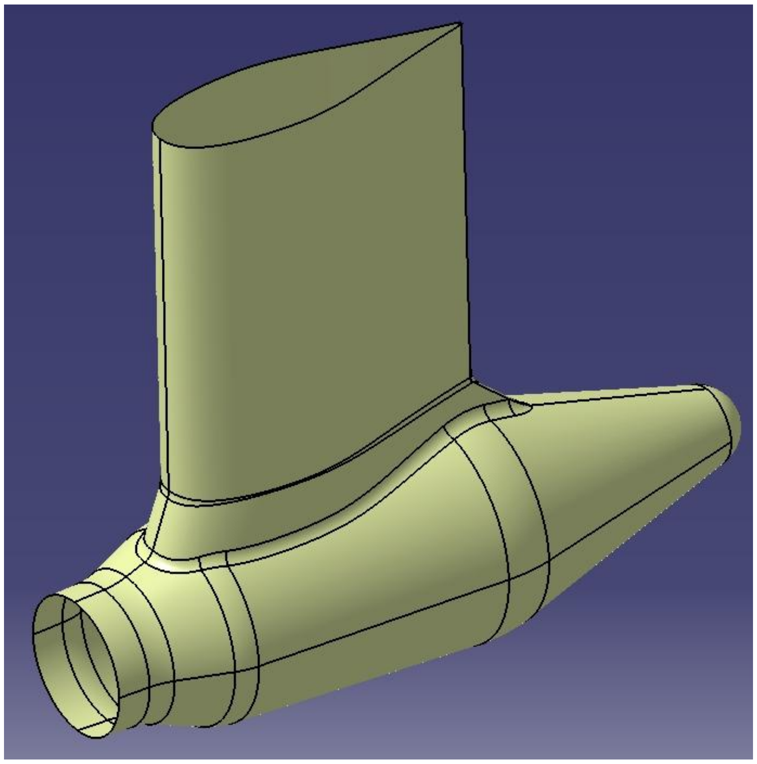


Fig. 0-23 The propulsor's body before being split for meshing

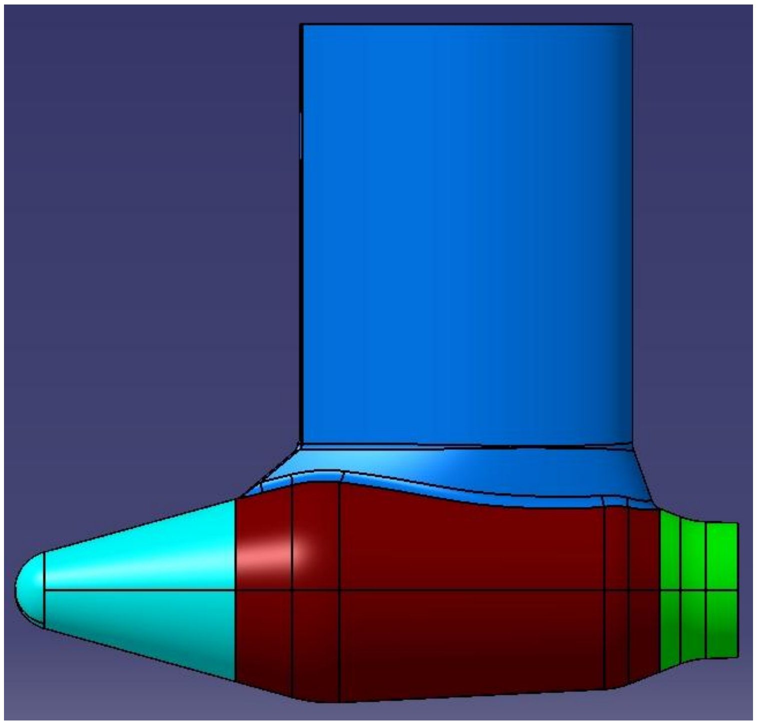


Fig. 0-24 Right half of surfaces to be meshed: Blue: strut - Green: Hub front - Red: hub mid - Cyan: Hub rear

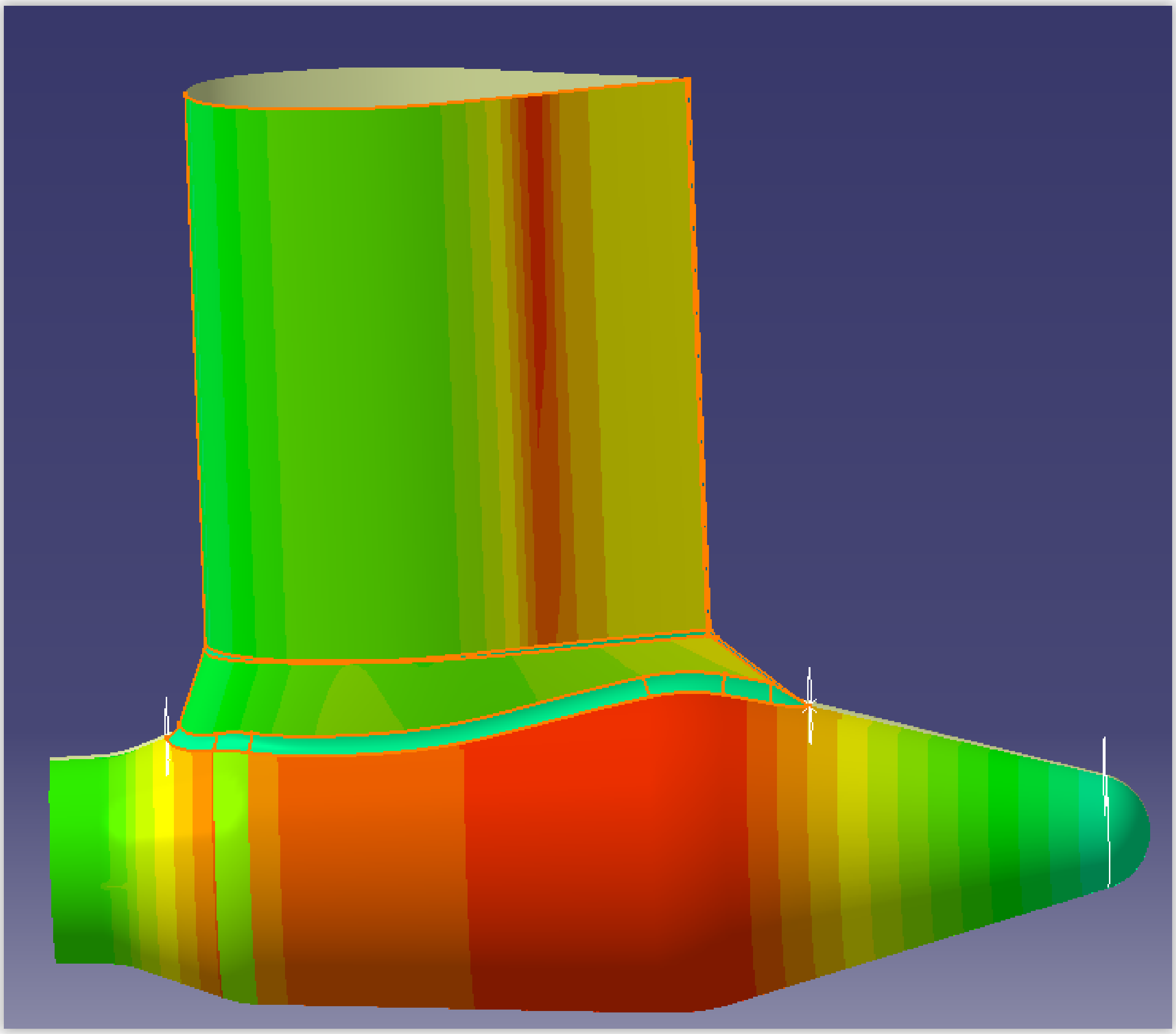


Fig. 0-25 Qualitative surface curvature analysis of the pod, using the “maximum” style of analysis. Red is minimum value, blue is maximum

Appendix C. The B-Series model excel files

The B-series parametric model is modified externally using the Excel file 'Wageningnen.xls', while its mesh distribution model is modified with the 'Wageningen_elements.xls' file. These

The model parameters file

The offsets for the B-series sections are calculated using a set of equations and relations that can be found in the bibliography. For these calculations, the required data are: the diameter 'D', number of blades 'Z' and the expanded blade area ratio 'A_E/A_O'. In addition to those, the pitch ratio 'P/D' and rake angle 'θ_{ip}' are required to construct the propeller.

The excel file contains the aforementioned main data, in addition to all the calculated offsets and other required parameters that will input in CATIA, and is organized as follows:

Number of sheets: 14

Sheet 1: Contains the main parameters modified by the user. Apart from main parameters, there are three more parameters included: the tip thickness, tipchordpercentage and hub_angle. The main data portion of the sheet that controls the model update is presented in Table 0-1

Below these, the initial calculations for each section are presented, which, apart from the pitch distribution column, are not modified by the user and are contained in the sheet for ease of reference.

Table 0-1 Sheet1: Propeller Main Data

A_E / A_0	Expanded blade ratio	
Z	Number of blades	
P / D	Pitch ratio	
D (m)	Diameter	(m)
Propeller rake angle (θ_{ip})	Rake angle	(deg)
tip thickness(m)	Thickness at the tip section	(m)
tipchordpercentage	Percentage of tip helix used in the outline creation	
Hub_angle	Angle of Hub profile	(deg)

Sheets 2 – 11: contain the calculated section offsets for sections at r/R= 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and tip. Each section takes up one sheet. All offsets are measured from the chord's TE and are arranged in such a way that facilitate associations in CATIA, as described in Appendix E. The sheet for section at r/R=0.2 is presented in Table 0-2. Some offsets are omitted from the presentation, to avoid unnecessary repetition.

Table 0-2 Sheet 3: section offsets for r/R=0.2

0.2_ChordLength	Chord length
0.2_A(GenToLE)	Distance of Generator line from TE
0.2_offset19	X offset for point 19 on back side
0.2_yback19	Y offset for point 19 on back side
0.2_offset18	X offset for point 18 on back side
0.2_yback18	Y offset for point 18 on back side
[...]	[...]
0.2_offset2	X offset for point 2 on back side
0.2_yback2	Y offset for point 2 on back side
0.2_offsetface19	X offset for point 19 on face side
0.2_yface19	Y offset for point 19 on face side
[...]	[...]
0.2_offsetface2	X offset for point 2 on face side
0.2_yface2	Y offset for point 2 on face side
0.2_yfaceTE	Y offset for TE point on face side
0.2_yBackTE	Y offset for TE point on back side
0.2_yBackLE	Y offset for LE point on back side
0.2_yFaceLE	Y offset for LE point on face side
0.2_B	Distance of max. thickness point from TE

Sheet 12: contains the pitches for each section, as calculated from the distribution defined in sheet1.

Sheet 13: contains the full offset calculations as determined by the B-series formulas. This sheet provides the data for the section offset sheets and is not imported in CATIA design table

Sheet 14: contains all the calculated offsets and is used to rename and re-arrange them in a suitable way. Section offset sheets 2 to 11 source their data from this sheet.

C-2. **The mesh distributions file**

The mesh distributions file contains parameters that control the number of elements and the distribution along the edges.

Number of sheets: 4

Sheet1: is the main data sheet that the user modifies.

The number of Elements segment contains the element number parameters and must always contain integer numbers as values. The descriptions of each parameter appear next to the parameter name. For the elements distribution segment, the distribution names can be seen in Fig. 0-26. The ratios values can be any real number. The symmetry values can be either 0 (false: the distribution is not symmetric) or 1 (true: the distribution is symmetric)

Number of Elements		
Chordwise		
Blade Face Chordwise	*Number of CW elements on blade face*	
Blade Chorwise Total	*Total number of CW elements on blade body*	
Front Hub Chordwise	*Number of CW elements on hub front*	
Rear Hub Chordwise	*Number of CW elements on hub rear*	
Hub Chordwise Total	*Total number of CW elements on hub*	
Spanwise		
Spanwise Blade	*Number of SW elements on blade face*	
Spanwise Fillet	*Number of SW elements on fillet face*	
Spanwise Hub (per face)	*Number of SW elements on each hub face*	
Spanwise Hub Total	*Total number of SW elements on hub*	
Elements distribution		
Distribution	**Ratios**	**Symmetry**
Chordwise		
CWBladeTop (CW1)		

CWBladeBottom (CW2)		
CWFilletBottomFace (CW3)		
CWFilletBottomBack (CW5)		
CWHubmid (CW4)		
CWHubFront (CWHF)		
CWHubRear (CWHR)		
Spanwise		
SWBladeLE (SWBLE)		
SWBladeTE (SWBTE)		
SWFilletLE (SWFLE)		
SWFilletTE (SWFTE)		
SWHubUpperFront (SWHUF)		
SWHubLowerFront (SWHLF)		
SWHubUpperRear (SWHUR)		
SWHubLowerRear (SWHLR)		

Sheets 2 -4 contain the data from sheet1, but organised for input in CATIA

Sheet2: element numbers

Sheet 3: distribution ratios

Sheet 4: symmetry notations. The values assigned in Sheet 1 are turned into “false” or “true” in order to be input in CATIA as Boolean parameters.

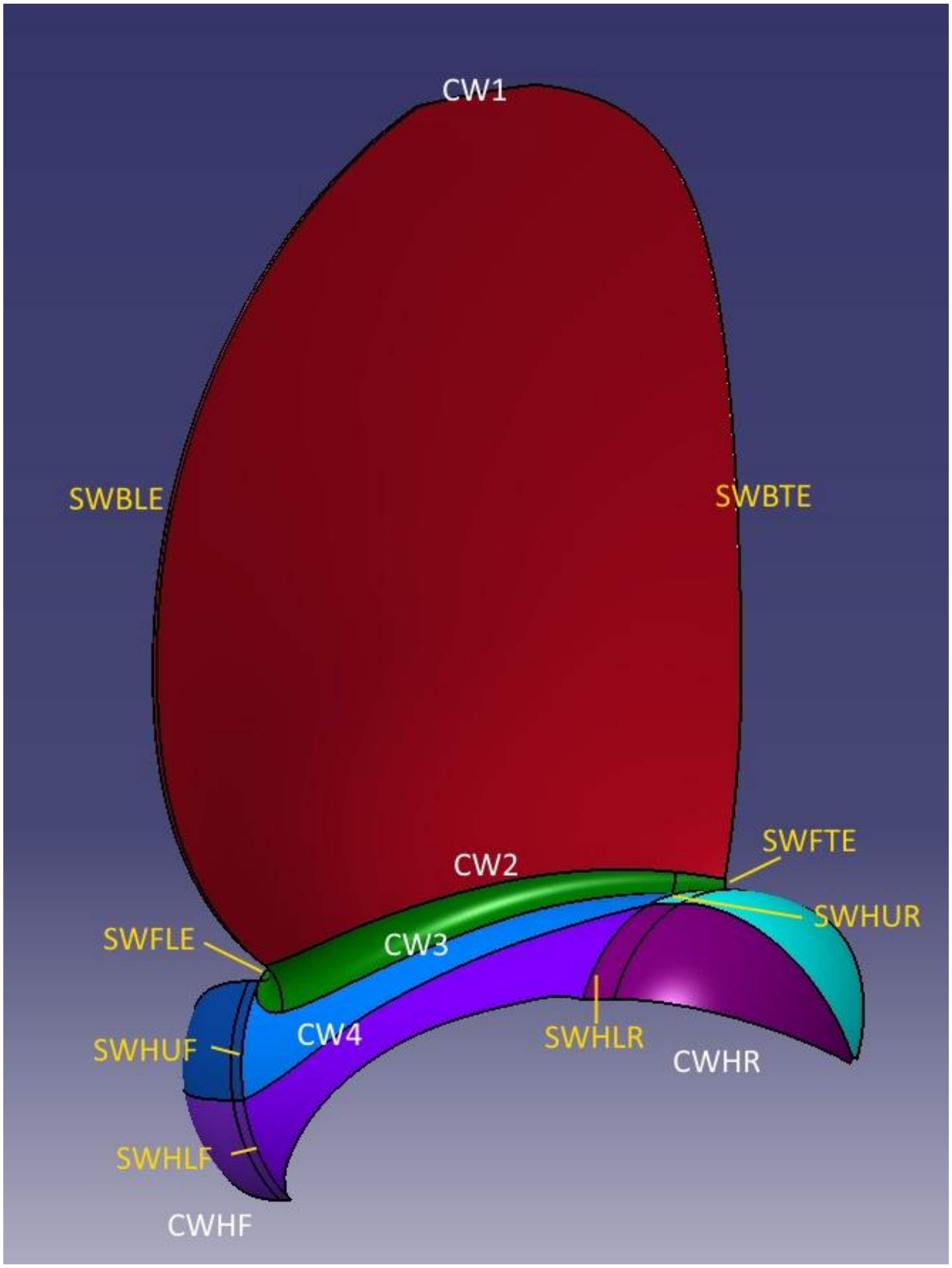


Fig. 0-26 Propeller mesh distribution naming convention

Appendix D. The podded propulsor excel files

The podded propulsor is an Assembly document, consisting of the propeller part and the pod body part. The propeller part used for the current setup is the Wageningen model, for which the excel file is already presented in Appendix C. The pod body model has its own Excel file, which will be presented further down.

For the mesh creation, an excel file is used for all of the assembly, which is an expansion of the propeller mesh excel file. For this reason, only the parameters concerning the pod body meshes will be presented here.

The pod body model is modified externally using the Excel file 'pod_dimensions.xls', while the assembly mesh is modified with the 'assembly_elements.xls' file.

D-1. The pod part model parameters file

The excel file contains the parameters described in VIII-2.2 - Design intent.

Number of sheets: 3

Sheet 1: contains the pod's and strut's geometric parameters as presented in Table 0-3 and Fig. 0-29

Table 0-3 pod_dimensions.xls Sheet 1: Pod main parameters

Pod body	
LP1 – 4	Length of pod cones that make up the pod body
DP1 – 3	Diameter of Pod cone edges
RFP1 – 3	Radii of fillets on cone edges
Strut	
OffsetS	Offset of Strut Leading Edge from X axis
LS	Strut Length
HSUp	Height of upper strut
HSLo	Height of lower strut (from pod body edge)
θSFr	Angle of lower strut front edge
θSRr	Angle of lower strut rear edge
Rs	Radius of fillet upper and lower strut join edge

Sheet 2: contains the foil section's parameters (Table 0-4)

Table 0-4 Pod_dimensions.xls Sheet 2: foil section parameters

l_chord	Chord length
max_thickness_prc	maximum thickness as percentage of the chord length
max_th_prc_fromLE	offset of max thickness from the LE as percentage of the chord length
te_offset_prc	thickness on TE as percentage of chord length

Sheet 3 : contains the fin's parameters (Table 0-5)

Table 0-5 Pod_dimensions.xls Sheet 3: Fin parameters

F_LC	Length of fin's foil cross-section chord length
fin_max_thickness_prc	Sections' maximum thickness as percentage of the chord length
fin_max_th_prc_fromLE	offset of max thickness from the LE as percentage of the chord length
fin_te_offset_prc	thickness on TE as percentage of chord length
FH	Height of fin, measured from edge of pod
FO	Offset from pod body's front edge
F_θFr	Fin outline's front angle
F_θRr	Fin outline's rear angle
fin_root_fillet	Radius of fillet at the root of the fin
fin_angle	In case of two fins, their angle relative to the XZ plane

D-2. The pod Assembly mesh excel file

The podded propulsor assembly mesh distribution file, 'assembly_elements.xls', is the same as the propeller one, but with added parameters for the pod. The propeller parameters are the same, so the user is referred to the description of the propeller mesh parameters file in Appendix C.

Number of sheets: 7

Sheet 1: is the main data sheet that the user modifies.

The number of Elements segment contains the element number parameters and must always contain integer numbers as values. The descriptions of each parameter appear next to the parameter name. For the elements ratio segment, the distribution names can be seen in Fig. 0-28 . The ratios values can be any real number. The symmetry values can be either 0 (false: the distribution is not symmetric) or 1 (true: the distribution is symmetric)

Fig. 0-27 Pod mesh excel file parameter names

Number of Elements	
Chordwise	
Strut Chordwise	*Number of CW elements on strut per face*
Pod Front Chordwise	*Total number of CW elements on pod front*
Pod Rear Chordwise	*Number of CW elements on pod rear*
Fin chordwise	*Number of CW elements on fin per face*
Spanwise	
Strut Spanwise	*Number of SW elements on Strut*
Pod Spanwise (per side)	*Number of SW elements on pod per side*
Fin spanwise	*Number of SW elements on fin*

Sheets 2 -7 contain the data from sheet1, but organised for input in CATIA

Sheets 2-4: propeller data

Sheet 5: pod element numbers

Sheet 6: pod distribution ratios

Sheet 7: pod symmetry notations. The values assigned in Sheet 1 are turned into "false" or "true" in order to be input in CATIA as Boolean parameters.

Table 0-6 Pod edges distributions naming convention (see also: Fig. 0-28)

Spanwise Distributions		Chordwise Distributions	
Name	*Explanation*	*Name*	*Explanation*
SWSF	Spanwise Strut Front	CW1-3	Chordwise at the Mid, 1-3 from strut top to pod bottom
SWSR	Spanwise Strut Rear	CW2-3F	Chordwise top (2) and bottom (3) edges on Front Hub
SWH1-4	Spanwise Hub Edges (1 to 4 from front to rear	CW2-3R	Chordwise top (2) and bottom (3) edges on Rear Hub

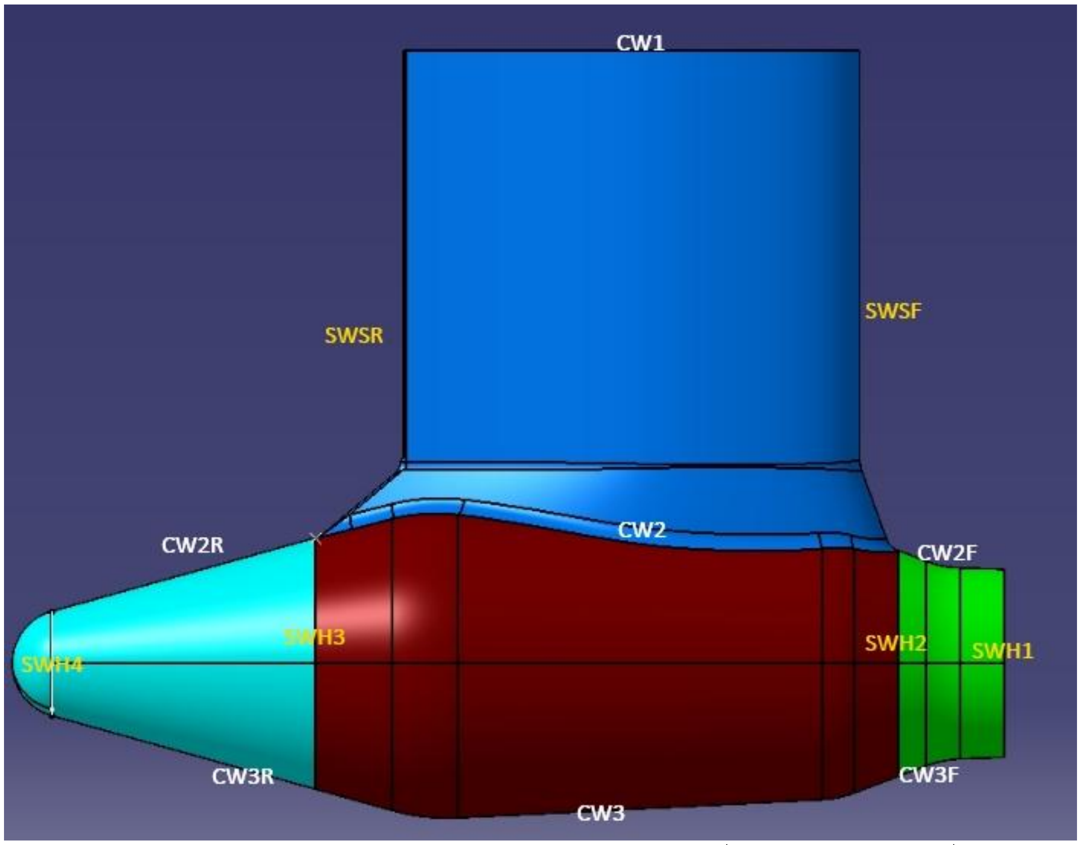
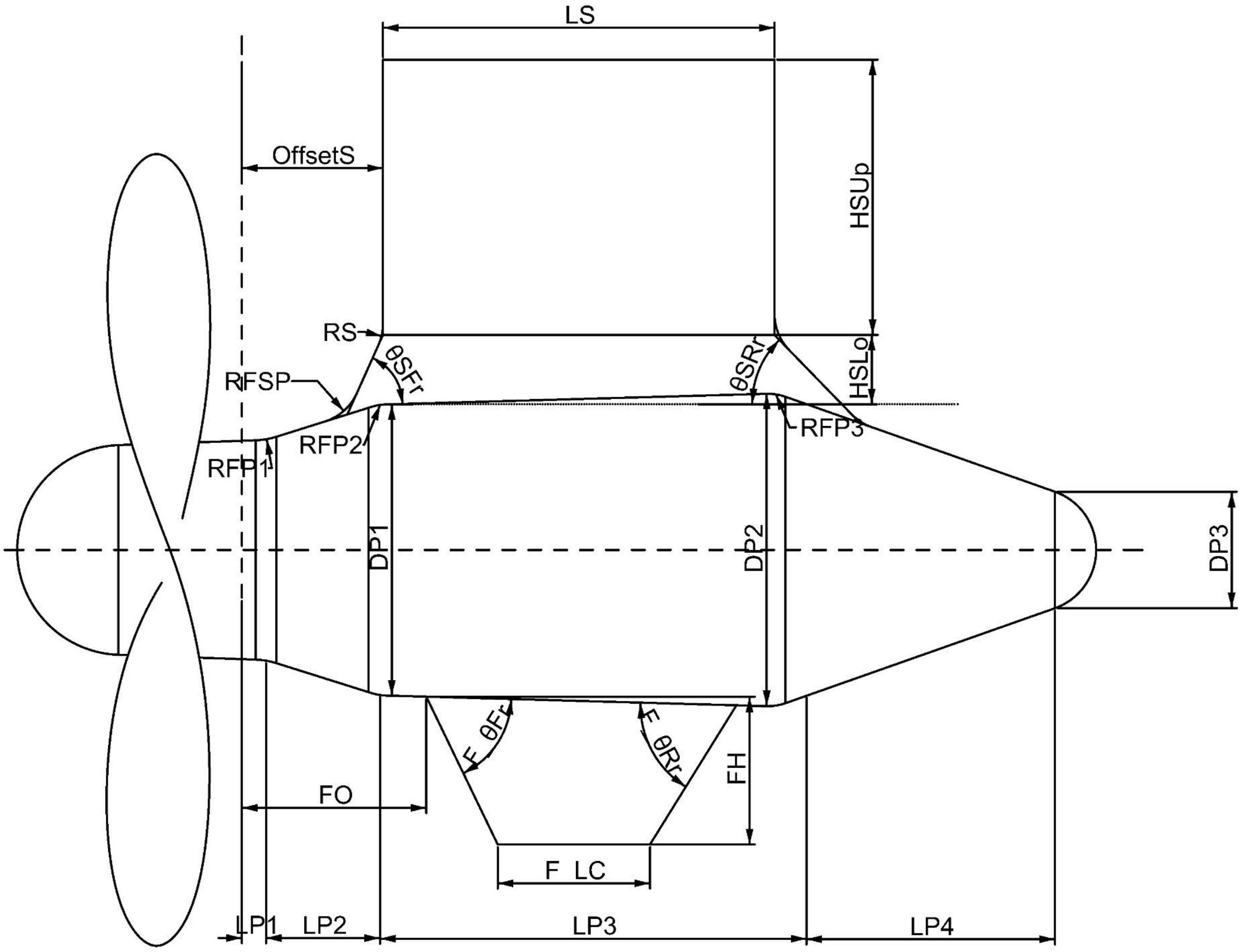


Fig. 0-28 Pod mesh distribution naming convention (see also: Table 0-6)


LS
OffsetS
HSUp
RS
θSFr
θSRr
HSLo
RFSP
RFP2
RFP3
RFP1
DP1
DP2
DP3
F θFr
F θRr
FH
FO
F LC
LP1
LP2
LP3
LP4

Appendix E. Example of design table creation: a propeller section

The design table feature is a very important part of the proposed method, for the creation and modification both of the geometry and of the meshes. For this reason, the step-by-step creation of a propeller section, from the excel file, to the design table creation, to the association and update is presented. The reader can expand on this example and apply a similar procedure to every model where external modification of parameters would be preferred (as opposed to in-model parameter modification). Additional examples can be found in CATIA User's Manual.

E-1. Design table file creation

The creation of a design table in a CATIA requires the creation of an external file that communicates the table data to CATIA. This external file is either a Microsoft Excel® .xls file, or a .txt file. In the current thesis, all the design tables are saved in the Excel file format and in some cases a single excel file provides data to multiple design tables. This will be explained below.

The design table Excel file must contain the following data:

Parameter name (optional: including units)

Parameter value

The table orientation can be either vertical or horizontal and the names should be unique for each parameter.

In case of a single Excel file containing multiple design tables, a good practice would be to assign different (exclusive) sheets to each table. However, the sheet order should never be changed. It is also recommended that the Excel file be stored in the same folder as the CATIA document(s).

For this example, the file used is "Wageningen.xls" ; its description can be found in Appendix C. The creation of section on the 0.2R plane is described. For this, the data used are the spline point offsets, the chord length and the distance of the generator line from the TE. The offsets parallel to the chord (X offsets) are measured from the section's trailing edge. The Y offsets are measured from the chord line. The design table data are contained in Sheet 3.

The creation of the section is part of the propeller blade wireframe creation and some of the previous steps in wireframe creation are assumed (as described in Appendix A).

E-2. Step-by-step creation of section with associations

1. Create Excel file containing required parameters and save
2. In CATIA, create the section sketch as follows:

   i. Enter sketcher
   ii. Draw **line** [chord] – impose length constraint and turn into **construction element**
   iii. Draw **lines** perpendicular to chord, passing through chord's LE and TE points [LE perpendicular, TE perpendicular], turn into **construction elements**.
   iv. Create two **splines,** each with 20 control points
   v. Spline LE points: impose coincidence constraint with LE perpendicular, offset constraint from chord
   vi. Spline TE points: impose coincidence constraint with TE perpendicular, offset constraint from chord
   vii. Impose offset constraints on rest of spline points
   viii. Impose offset constraint: GL intersection point from TE perpendicular
   ix. Impose angle constraint between chord and H-direction
   x. Exit sketcher

   Notes: (vi) For quicker constraint definition, the auto-constraint tool is recommended. Since the selection order of the points is important when using it, a tip to make creating the associations easier is selecting the points in such a way that reflects the order in which the parameters appear in the Excel file. In our case, the points were selected in order, from TE to LE (the LE and TE points where left out, since they are constrained in previous steps). The chord and TE perpendicular where set as reference elements and the constraint mode was selected to be stacked. The offsets created appear in the specification tree in the same order as selected, with the X offset of each point followed by the y offset, followed by the X offset of the next point and so on. This order is reflected in the excel file

3. Create Design table and associations for the section offsets:

   i. Select Design Table creation from the Knowledge toolbar. In the window that pops up, input table name, select "Create a design table from pre-existing file", select orientation and input sheet index and press ok
   ii. In automatic association, select no. The table is created

iii. Go to the associations tab. On the specifications tree select the sketch. The constraints contained in the sketch are shown.

iv. Associate each offset in the parameters box with the corresponding design table parameter in the columns box by selecting them and clicking the Associate button

v. Select OK and exit

vi. Update model and check for errors

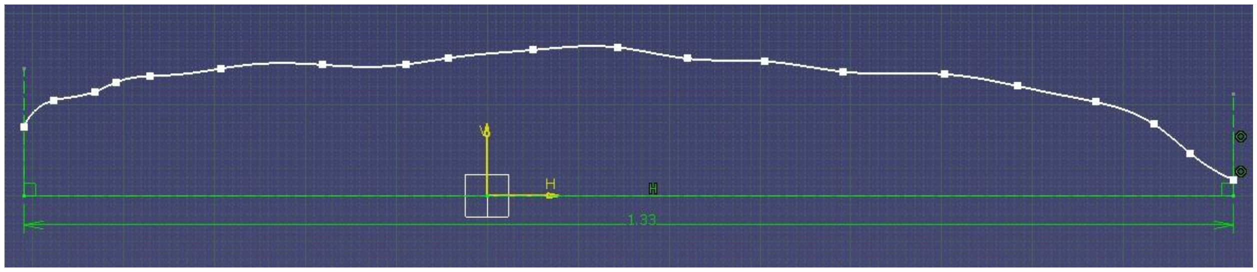
Notes: (i)For the 0.2RSection offsets, sheet index=3, horizontal orientation
(ii) selecting yes or no does not influence the outcome in the current case, since there are no parameters with a similar name
(iii) Using the filter is optional, but facilitates the procedure
(iv) The selected offset from the parameters box turns orange so that the user can see which constraint is currently selected.

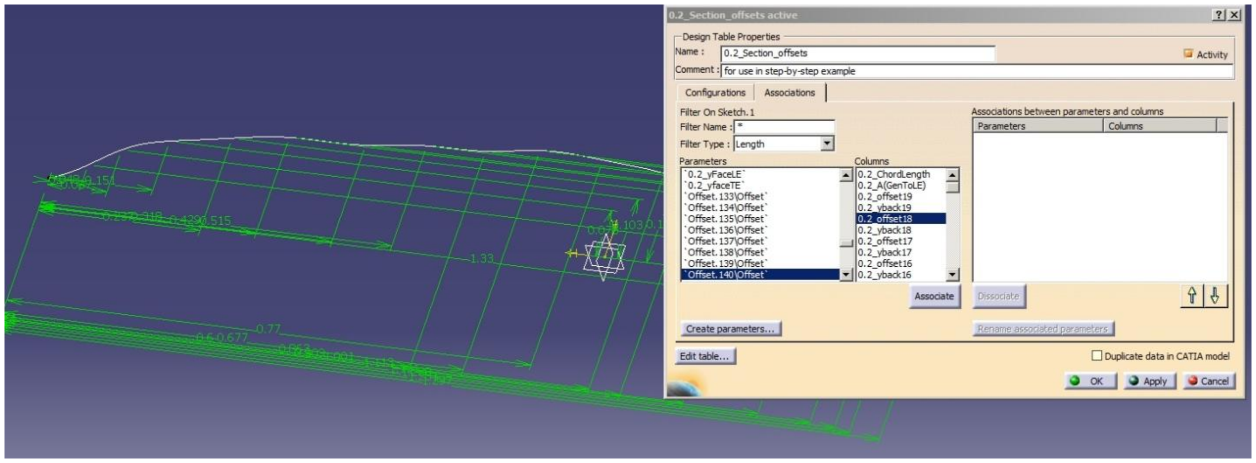
4. Create Design table and parameters for the pitch length:
   i. Create design table as in steps 3.i – 3.ii
   ii. Click the Create Parameters button
   iii. Select the desired parameters and click ok
   iv. Click OK to exit design table definition. The created parameters appear in the specification tree

Notes: (i) sheet index = 12, orientation horizontal for the Wageningen file
(ii) In the created propeller model, a sketch with the pitch distribution as a spline is created, where the spline points offset from the V-axis at each radius is the corresponding pitch length. However, in this example the pitch lengths are saved in parameters in order to showcase this capability
(iii) The pitch length parameters type should be either real or length.

5. Set pitch angle:
   i. Enter the section sketch
   ii. Double-click the angle constraint -> right-click the value box -> edit formula
   iii. Write the pitch angle formula, clicking the corresponding parameters where required.

Notes: (ii) this is one common way to bring up the formula editor for a constraint. Formulas can always be edited
(iii) It is assumed that the propeller main parameters, including the propeller diameter D have been imported in a design table as well.
*The formula for the current example is: atan(`pitch_r0.2` /(2*PI *0.2*0.5*`D`)), where `pitch_r0.2`: the parameter containing the pitch for the 0.2 section*
`D` : The parameter containing the propeller diameter.




0.2_Section_offsets active

Design Table Properties

Name : 0.2_Section_offsets

Activity

Comment : for use in step-by-step example

Configurations | Associations

Filter On Sketch.1

Filter Name : *

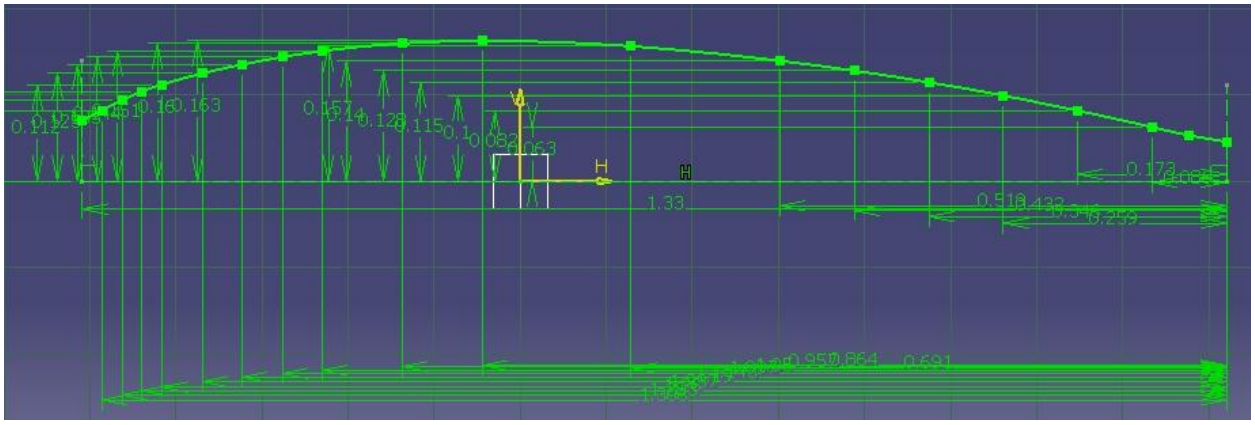
Filter Type : Length

Parameters:
```
`0.2_yFaceLE`
`0.2_yfaceTE`
`Offset.133\Offset`
`Offset.134\Offset`
`Offset.135\Offset`
`Offset.136\Offset`
`Offset.137\Offset`
`Offset.138\Offset`
`Offset.139\Offset`
`Offset.140\Offset`
```

Columns:
```
0.2_ChordLength
0.2_A(GenToLE)
0.2_offset19
0.2_yback19
0.2_offset18
0.2_yback18
0.2_offset17
0.2_yback17
0.2_offset16
0.2_yback16
```

Associations between parameters and columns

Parameters	Columns



Appendix F. Step- by-step creation and export of meshes

The mesher can be found in the Advanced Meshing Tools of the Analysis and Simulation product. The mesh data is saved in a CATIA analysis file (.CATAnalysis)

Meshing with the current method consists of the following general phases:

A. Creation of surfaces to be meshed from model
B. Creation of design table containing mesh distributions
C. Creation of the meshes based on split surfaces
D. Export of the meshes/patches that compose a single body into a bulk data file
E. Run mesh file through the interface program to produce body input file
F. Check patches orientation with Tecplot
G. Run through UBEM
H. Final check of surface vectors with Tecplot

Step A is the model creation phase as described in the respective chapters.

Steps B and C are one-off and saved in the analysis file. For use of existing documents, they are by-passed

Steps D-E are required every time a mesh needs to be exported

The rest of the steps can be bypassed if there already is an exported mesh from the existing model and the new meshes are created by a simple update of the geometry or the mesh distribution, since the mesh orientation remains the same. In case a surface needs to be remeshed (for example due to an update error) or a new mesh be created, a check should be made in Tecplot as in step F.

F-1. Step-by-step mesh export for use in UBEM

The following step-by-step description deals with the general steps B-H

1. Create a new Analysis file from the product
2. Create the design table and import it in the Analysis file
3. Create parameters from the imported design table
4. Create mesh on supporting surface:
   i. Select **surface mesher** from the meshing methods toolbar
   ii. Select supporting surface
   iii. In the global meshing parameters, mesh tab set the following:

Shape:	Frontal quadrangle method
Type:	Linear
Mesh size:	As required (here, 0.02m)
Quads only	Selected

Automatic mesh capture	unselected

In the geometry tab, set constraint sag the same as mesh size

iv. Click OK to enter the surface mesh workbench.

v. Select **Element distribution** from the 1D mesh specification toolbar

vi. Define nodes distribution on edge as follows:

Support	Select edge
Type	Geometric or Arithmetic
Number of edges	Edit formula-> set corresponding parameter
Size 2/Size1	Edit formula -> set corresponding parameter

Click OK to accept nodes distribution (appears as green dots on the edge)

vii. Define nodes on the other surface edges

viii. Select **mapped method** from the 2D Mesh Specifications toolbar and then click on the surface

ix. Specify required parameters.

  i. Split quadrangles must remain unselected

  ii. Mesh domain corners can be seen on the model, denoted as C1, C2, C3 and C4. In case they need to be moved, it can be done by clicking first on the domain corner and then on the surface vertex it will be moved to.

  iii. Select OK to mesh the surface

x. Exit the sketcher

5. Create meshes on the rest of the surfaces
6. Export mesh as follows

i. In the specification tree, deactivate every mesh <u>that will not be exported</u>

ii. Click **Export Mesh** from the Import/Export toolbar

iii. Save as bulk data file (.dat) with a filename meshxx.dat, where xx=00 to 99

7. Run mesh through the interface program

i. Create interface program's input, specifying the following:
Line2: Bulk data mesh file name
Line 4: number of meshes that the body consists of
Line 6: total number of chordwise and spanwise elements the body consists of
Line 8:type of surface (1: face-back surface, 2: modified face-back)
Line 10: print direction (1: chordwise, 0: spanwise)
Line 12: output filename
Line 15, 17,…(depending to number of surfaces): orientation markers and
To set the orientation markers, either check element order in CATIA mesher, or assign arbitrary values, run and open with tecplot, modifying as in step 8.
In case the same geometry has been run through the interface programme before, the same markers can be used
Execute program

8. Open output file with Tecplot.
Plot type: 3D Cartesian
Switch on the mesh and contour layer

The mesh should appear as created in CATIA.

Check the i and j contours: In a face-back surface, i contours should go from the Face LE to the TE and then from the Back LE to the TE; j contours should go from bottom to top. (Fig. 0-31, Fig. 0-32)

In a modified face-back surface, i contours should go from the center seam to one side and then from the center seam to the other side; j contours should go from front to rear. (Fig. 0-33)

If the surface appears twisted, or the contours are wrong, modify the orientation markers in the interface program's input file and re-run. Repeat until the mesh surface is correct

9. Repeat steps 6-8 for every body that will be input in UBEM
10. Finally, when the vector file is created in UBEM, open with Tecplot to check vector orientation. The calculated normal vectors should point outwards. In case of a body's vectors pointing inwards, re-arrange the corresponding meshes in the CATIA specification tree, export and run body meshes through the interface program.

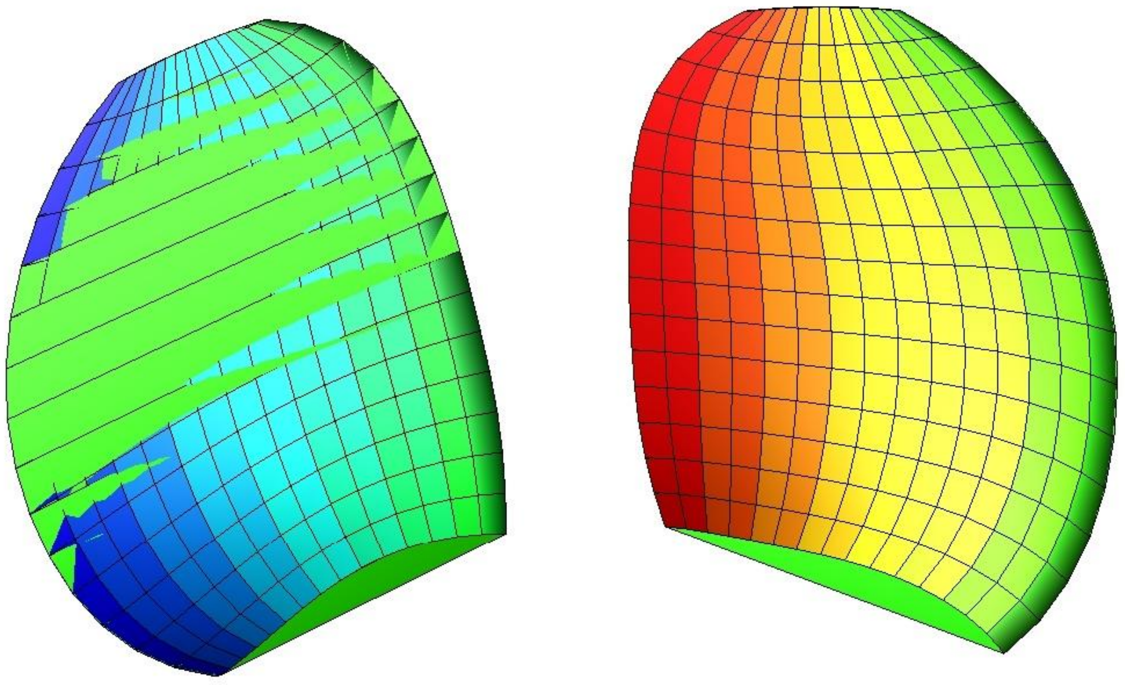
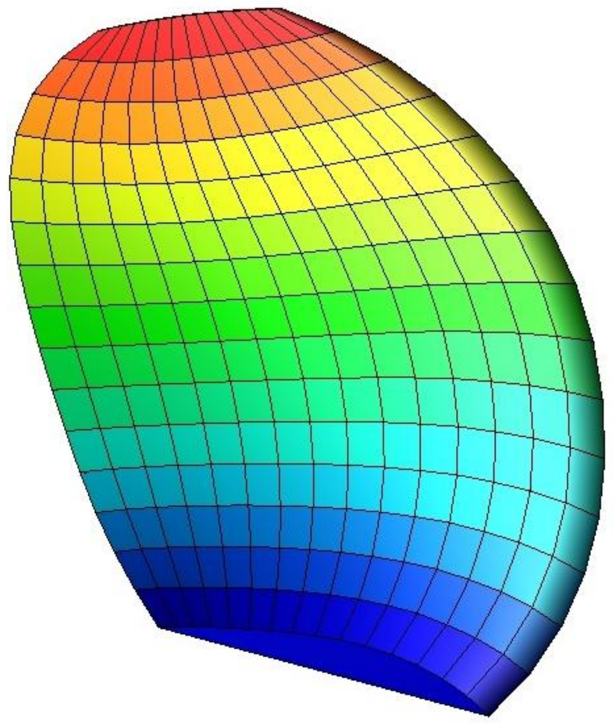


Fig. 0-31 i contours on face-back surface type mesh (propeller blade – left: Face side, right: Back side)



`

Fig. 0-32 j contours on face-back surface type mesh (propeller blade –Back side)

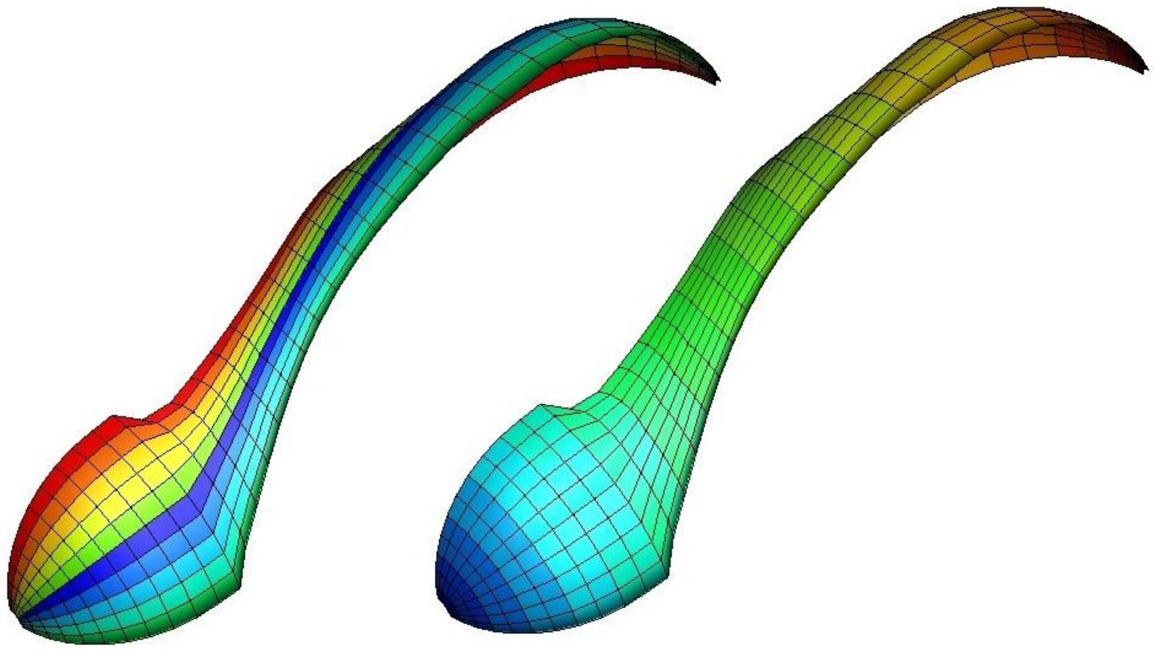


Fig. 0-33 Contours on a modified face-back type surface: left: i contours, right: j contours

Appendix G. The source code of the interface program

```
    program Mesh_translator

    implicit none

    ! Variables
    character:: cchar*2, filename*10, outfile*17
    integer:: ccount, cindex, gcount, gindex, aa, bb, ii, jj, kk, ll, nn,
srfcn, ndif, ncw, nsw, temp, prntdirection, cc,ss, tp
    integer:: idif, tabind, nodn, nodind, sindex, iscircular, ispatches,
stepcw, stepsw, ni, nj, startcw, endcw,startsw, endsw
    integer, allocatable:: element(:), srfc(:), srfci(:,:), na(:), nb(:),
nc(:), nd(:), node(:), srfcinfo(:,:), indexi(:,:), indexj(:,:), stepi(:),
stepj(:)
    logical:: encc, encg, facechange
    real, allocatable:: xn(:), yn(:), zn(:), x(:,:), y(:,:), z(:,:), xa(:,:),
ya(:,:), za(:,:), ncoord(:,:),te(:,:)
    real:: difference

    ! Body of Mesh_translator

    open (unit=40, file='input.dat')


    1   FORMAT(8x,i16,16x,f16.9,f16.9,8x,/,8x,f16.9)       !Formatting for
reading GRIDS
    2   FORMAT(8x,i8,i8,i8,i8,i8,i8)                        !Formatting for
reading CQUADS
    3   FORMAT('node=', i6,f16.9,f16.9,f16.9)
    5   FORMAT(a2)
    18  FORMAT (a17)
    6   FORMAT('element=',i8,i8,i8,i8,i8,i8)

    read(40,*)
    read(40,*) filename
    read(40,*)
    read(40,*) srfcn   !number of surfaces
    read(40,*)
    read(40,*) ncw, nsw
    read(40,*)
    read(40,*) tp
    read(40,*)
    read(40,*) prntdirection
    read(40,*)
    read(40,18) outfile

    OPEN (unit=20, file=outfile)

    read(40,*)
    allocate (srfcinfo(srfcn,5), srfci(srfcn,2)) !srfci will contain indexes
for the start and end of each srf in the node matrix

    do ll=1,srfcn
        read (40,*)
        read (40,*) (srfcinfo(ll,ii),ii=1,5)    !1-3 is srfc orientation, 4
is cw elements, 5 is sw elements
    end do

        !CATIA meshes must be created so that each patch doesn't share nodes
with the adjacent one (no condensation)
        !ncw=ncw+srfcn       !final number of nodes is ncw+1, though. This is
just the number of nodes in the file
                            !CAUTION the above is true only for patches that
share a spanwise connection

    open (unit=10, file=filename)


   !1st Readthrough -
   !---------DATA LOCATION IN INPUT FILE

   !The logical variables are used to locate the start of each data group
inside the input file
   !grid or cquad. When the first set of data of each group hasn't been
encountered yet, value is 'false'
   !On encounter, they toggle to 'true'

    encg=.false.
    encc=.false.

    gcount=0
    ccount=0
    ii=0
    sindex=0

     do
        ii=ii+1
        read(10,5) cchar
            if (cchar.EQ.'EN') then
                exit
           end if
            if (cchar.eq.'GR'.OR.cchar.eq.'* ') then
                 if (encg==.false.) then
                    encg=.true.
```

```
                  gindex=ii
               end if
               gcount=gcount+1    !each node requires 2 lines of data in
the input mesh file, so total number of nodes equals half number of counted
lines
            end if
            if (cchar.eq.'CQ') then     !elements denoted with CQUAD4 in mesh
file
               if (encc==.false.) then
                  encc=.true.
                  cindex=ii
               end if
               ccount=ccount+1    !counts elements
            end if
    end do

    nodn=(ncw+srfcn)*(nsw+1)

    if (tp.eq.2) then
        nodn=(nsw+2)*(ncw+srfcn/2)  !the surfaces are added in the chordwise
direction and they have a "LE" that runs in the chordwise direction
    end if

    if (nodn.NE.(gcount/2)) then
        write (*,*) 'error, check input file element distribution'
        write(*,*) nodn, 'inputfile <> meshfile', gcount/2
        pause
    end if

    nodn=gcount/2   !number of nodes counted in input mesh file

    allocate (node(nodn), xn(nodn), yn(nodn), zn(nodn))
    allocate (element(ccount), srfc(ccount), na(ccount), nb(ccount),
nc(ccount), nd(ccount))

    rewind (10)

    ii=1
    do  while (ii<gindex)
        read(10,*)
        ii=ii+1
    end do
    kk=0
    do while (ii>=gindex.and.ii<=(gindex-1+gcount/2))
        kk=kk+1
        read(10,1) node(kk), xn(kk), yn(kk), zn(kk)
        ii=ii+1
    end do
    cindex=cindex-nodn
        do while (ii<(cindex))
        read(10,*)
        ii=ii+1
    end do

    kk=0
    nn=1
    srfci(1,1)=1

    do while (ii>=cindex.and.ii<=cindex+ccount-1)
        read(10,5) cchar
        if (cchar.eq.'CQ') then !reading the first 2 characters for each line
(record)
        backspace (10)          !and then needs to get to the beginning of the
line to read it with formatting
            kk=kk+1
            read(10,2) element(kk), srfc(kk), na(kk), nb(kk), nc(kk), nd(kk)
            if (srfc(kk)>nn) then   !indexing the starting and ending element
of each surface
                srfci(nn,2)=kk-1
                nn=nn+1
                srfci(nn,1)=kk
            end if
            ii=ii+1
        end if
    end do
    srfci(srfcn,2)=kk

    !------------END OF DATA LOCATION

    !-------------------------------------------------------

    !------------MESH CREATION

    !each element is defined by 4 nodes: na, nb, nc, nd, presented in this
order in the input file
    !The mesh is created by ordered elements, so I'm using the elements to
create the ordered mesh
    !assuming the i direction as a 'row' and the j direction as a 'column',
I'm using the na node to fill in the mesh
    !at the end of each 'row', I'm also using the nb node.
    !to fill in the final row of nodes, I'm using the nc nodes of the last
element on each 'column'
    !the last node (on the corner) is node nd of the last element of the mesh

    ncw=ncw+1       !final number of nodes is number of elements+1 per
direction
    nsw=nsw+1
    facechange=.false.
    if (tp.eq.1) then
        ncw=ncw+1   !TE nodes are double for each part face-back
        allocate (te(nsw,3))

    else if(tp.eq.2) then

         nsw=nsw+1   !TE nodes are double for each part face-back
ncw
         !nsw=(nsw+1)/2  !each surface is half the total nsw, so we have
nsw/2+1 per surface. Total nsw=nsw+2, already have +1 in previous line

    end if

    nn=ncw*nsw

    allocate (x(nn,nn),y(nn,nn),z(nn,nn))
    x=0
    y=0
    z=0

    cc=1 !cc, ss are indexes for final array (cc chorwise, ss spanwise)
    ss=1
    nn=0

    do ll=1,srfcn   !this loop creates the coordinates array for each surface
        nn=(srfcinfo(ll,4)+2)*(srfcinfo(ll,5)+2)

        allocate(xa(nn,nn),ya(nn,nn),za(nn,nn))
        xa=0
        ya=0
        za=0

        ii=1
        jj=1

    !The surface consists of smaller sub-surfaces(faces) that each gets its
own mesh. In order to create the final mesh, I read the mesh part for each
sub-surface
    !and then I add it in the final array by removing the common nodes

        !start of sub-surface reading - array creation
        do kk=srfci(ll,1),srfci(ll,2)
            xa(ii,jj)=xn(na(kk)) !the array is filled primarily by the na
nodes of each element
            ya(ii,jj)=yn(na(kk))
            za(ii,jj)=zn(na(kk))
            ii=ii+1 !the ii index is with the surface direction (ii=1,ncw+1
if surface has chorwise direction, else ii=1,nsw+1). Let's say ii defines
'columns' and jj 'rows'
            if (kk.LT.srfci(srfcn,2)) then
```

```
                if (nb(kk).NE.na(kk+1)) then    !if nb(kk)=na(kk+1) then
elements kk, kk+1 are next to each other in the 'row', otherwise, element kk
is the last one in the 'row'
                    xa(ii,jj)=xn(nb(kk))
                    ya(ii,jj)=yn(nb(kk))
                    za(ii,jj)=zn(nb(kk))
                    !indexi(ll,2)=ii !should be either ncw-1 for a chorwise
distribution or nsw-1 for a spanwise distribution
                    jj=jj+1     !change of 'row'
                    ni=ii
                    ii=1
                end if
            else    !last element of the surface, there's no kk+1 so I need
this exception
                xa(ii,jj)=xn(nb(kk))
                ya(ii,jj)=yn(nb(kk))
                za(ii,jj)=zn(nb(kk))
                jj=jj+1     !change of 'row'
            end if
        end do
        ii=1
        !The following loop is to fill the last 'row' of nodes

        kk=srfci(ll,2)-(ni-2) !elements in 'row' are ni-1, nodes are ni
        do while (kk.LE.srfci(ll,2))

            xa(ii,jj)=xn(nd(kk))
            ya(ii,jj)=yn(nd(kk))
            za(ii,jj)=zn(nd(kk))
            kk=kk+1
            ii=ii+1
        end do

        !this is to add the last node in the corner (nc of the last element
of the surface)
        kk=srfci(ll,2)
        xa(ii,jj)=xn(nc(kk))
        ya(ii,jj)=yn(nc(kk))
        za(ii,jj)=zn(nc(kk))
        ni=ii   !number of nodes in the i direction (number of 'columns')
        nj=jj   !number of nodes in the j direction (number of 'rows')
        !end of sub-surface array


         if (tp.eq.2) then
            if (ll.eq.(srfcn/2+1).and.facechange.eq..false.) then
                facechange=.true.
                cc=1
            end if
        end if

        !Adding the sub-surface array to the final array
        startcw=1
        stepcw=1
        startsw=1
        stepsw=1

        if (srfcinfo(ll,3).EQ.0) then
            endcw=ni    !the chordwise direction is i   (I have ni columns)
            endsw=nj    !the spanwise direction is j
        else
            endcw=nj    !the chorwise direction is j
            endsw=ni    !spanwise direction is i

        end if

        if (srfcinfo(ll,1).EQ.1) then !sub-array is created with the
direction of added sub-arrays
            stepcw=-1
            temp=startcw
            startcw=endcw
            endcw=temp
        end if

        if (srfcinfo(ll,2).EQ.1) then !sub array is created from top-to-
bottom
            stepsw=-1
            temp=startsw
            startsw=endsw
            endsw=temp
        end if

        if (ll.LT.srfcn) then   !removes one set of common nodes between
adjacent patches --- CAUTION! this is true only if common edge is spanwise!

            endcw=endcw-stepcw

        end if

        if (tp.eq.1.or.tp.eq.2) then  !type 1 (face-back surfaces) don't have
a common edge on TE, so I need to preserve this)
            if (ll.eq.srfcn/2) then
                endcw=endcw+stepcw
  !              ncw=ncw+1
            end if
        end if

        if (srfcinfo(ll,3).EQ.0) then
                if (tp.eq.2.and.ll.gt.srfcn/2) then
                    ss=nsw/2+1
                end if

            do ii=startcw,endcw,stepcw
                do jj=startsw,endsw,stepsw
                    x(cc,ss)=xa(ii,jj)
                    y(cc,ss)=ya(ii,jj)
                    z(cc,ss)=za(ii,jj)

                    ss=ss+1
                end do
                ss=1
                cc=cc+1
                if (tp.eq.2.and.ll.gt.srfcn/2) then
                    ss=nsw/2+1
                end if

            end do
        else
            if (tp.eq.2.and.ll.gt.srfcn/2) then
                    ss=nsw/2+1
            end if
            do jj=startcw,endcw,stepcw
                do ii=startsw,endsw,stepsw
                    x(cc,ss)=xa(ii,jj)
                    y(cc,ss)=ya(ii,jj)
                    z(cc,ss)=za(ii,jj)

                    ss=ss+1
                end do
                temp=ss
                ss=1
                if (tp.eq.2.and.ll.gt.srfcn/2) then
                    ss=nsw/2+1
                end if

                cc=cc+1

            end do

        end if

    deallocate (xa,ya,za)
    end do


    !Tecplot output ------------------------------------
    !Header --------------------------------------------

    write(20,4) filename
```

```
    4 FORMAT('TITLE = ',a6 )
      write(20,7)
    7 FORMAT('VARIABLES = "x", "y", "z", "i","j" ')
    9 FORMAT(' ZONE T=" ISO theta ", I=',i5,', J=',i5,', F=POINT ' )
    11 FORMAT('nodes CW=', i5,', SW= ',i5)
    8       FORMAT(3(1x,f16.9),i5,i5)

        if (prntdirection.eq.0) then    !spanwise distribution (span nodes
printed first)
            write(20,9) nsw, ncw

            do jj=1,nsw
                do ii=1,ncw
                    write(20,8) x(ii,jj), y(ii,jj), z(ii,jj), ii, jj
                end do
            end do

        else if(prntdirection.eq.1) then        !chordwise distribution
(chord nodes printed first)
            write(20,9) nsw, ncw

            do ii=1,ncw
                do  jj=1,nsw
                write(20,8) x(ii,jj), y(ii,jj), z(ii,jj), ii, jj

                end do

            end do
        else

            write(20,9) ncw, nsw

            do jj=1,nsw
                do  ii=1,ncw
                write(20,8) x(ii,jj), y(ii,jj), z(ii,jj), jj, ii
                end do

            end do

        end if

    end program Mesh_translator
```