Lappeenranta University of Technology
Faculty of Technology
Degree Programme in Energy Technology

New computer software for defining the three-dimensional geometry for a centrifugal compressor impeller

The topic of this Master's thesis was approved by the Department Council of Energy and Environmental Technology on **23.12.2010**

Supervisors: Pekka Röyttä

Examiners: Jari Backman
Pekka Röyttä

Kotka, 23.12.2010

Antti Tiihala
Puutarhakatu 15 B 19
48100 Kotka
FINLAND
+358505323070

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknillinen tiedekunta
Energiatekniikan koulutusohjelma

Antti Tiihala

New computer software for defining the three-dimensional geometry for a centrifugal compressor impeller

Diplomityö

2010

30 sivua, 11 kuvaa ja 3 liitettä

Tarkastajat: Jari Backman
Pekka Röyttä

Hakusanat: kompressori, juoksupyörä, suunnittelu, ohjelmointi

Keywords: compressor, impeller, design, programming

Diplomityössä suunnitellaan tietokoneohjelmisto keskipakokompressorin juoksupyörän geometrian määrittelyyn. Työn tilaajana toimii virtaustekniikan laboratorio Lappeenrannan teknillisestä yliopistosta. Työ tulee olemaan samankaltainen Tomi Putuksen vuonna 2009 tekemän diplomityön kanssa, jonka aiheena oli radiaalikompressorin juoksupyörän virtauskanavan suunnittelu. Putus kirjoitti diplomityössään tietokoneohjelman, jota voidaan käyttää määrittämään juoksupyörän kolmiulotteinen geometria annetuilla alkuarvoilla. Tässä työssä suunniteltava ja toteutettava ohjelma on lähes samanlainen, mutta suurena erona on käytetty ohjelmointikieli (C++) ja kompressorin meridionaalisen projektion muodon määrittely.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology
Degree Programme in Energy Technology

Antti Tiihala

New computer software for defining the three-dimensional geometry for a centrifugal compressor impeller

Master's Thesis

2010

30 pages, 11 figures, and 3 appendices

Examiners: Jari Backman
Pekka Röyttä

Keywords: compressor, impeller, design, programming

In this thesis, a computer software for defining the geometries for centrifugal compressor impellers is designed and implemented. The project is done under the supervision of Laboratory of Fluid Dynamics in Lappeenranta University of Technology. This thesis is similar to the thesis written by Tomi Putus (2009) in which centrifugal compressor's impeller flow channel is researched and commonly used design practices are reviewed. Putus wrote a computer software which can be used to define impeller's three-dimensional geometry based on the basic geometrical dimensions given by preliminary design. The software designed in this thesis is almost similar but uses different programming language (C++) and different way to define the shape of the impeller's meridional projection.

Contents

NOMENCLATURE

Abbreviations

ANSI American National Standards Institute
CAD Computer Aided Design
CFD Computational Fluid Dynamics
GLUT Graphics Library Utility Toolkit
IGES Initial Graphics Exchange Specification
SDL Simple Direct media Layer

1 INTRODUCTION

The main goal of this thesis is to design and impelement new computer software which will be used to define a three-dimensional geometry for the centrifugal compressor impeller. This project is under the supervision of Laboratory of Fluid Dynamics in Lappeenranta University of Technology. This software is called Curvepoint and it's programmed in C++ using developer tools offered by the GNU Project. The source code of Curvepoint is generic so it can be ported to any system supporting C++. However, Curvepoint is developed by using GNU/Linux and therefore that's the most tested platform.

Curvepoint defines the three-dimensional geometry for an impeller and saves it to the Initial Graphics Exchange Specification (IGES) format which meets the ANSI standard and is also widely supported by CAD softwares. There are several ways to use these files. For example, they can be used in CFD calculations.

The first part of this thesis describes centrifugal compressors in general. After that, the computer graphics history is revealed and B-splines are explained. The B-splines are very important in this project because the shape of both the meridional projection and the gamma distribution are defined by using them. The rest of this thesis deals with the Curvepoint software itself. Programming techniques, file formats, and the general structure of the software are explained in detail.

2 DESIGNING THE CENTRIFUGAL COMPRESSOR IMPELLER

In this thesis, we are only interested in the compressor impellers so there's no deeper research for the compressors in general. However, we must first know some basics of the centrifugal compressors. These are very common nowadays in many industries. There has been researching for the high-speed compressors in the Laboratory of Fluid Dynamics in Lappeenranta University of Technology and that field can be considered as their bravura.

2.1 What is a compressor?

The basic aim for a compressor is to compress a working fluid and deliver it at a pressure which is higher than its original pressure. If this fluid is liquid, it's practically incompressible and *the compressor* is not then the right word to describe it, so we call it *a pump*. There are numerous applications where the compressors are required, for example, to provide air for a combustion or to provide air for the driving pneumatic tools. (Boyce, 2003, p. 1)

There are many different types of compressors around. As mentioned before, in this thesis we are only interested in the centrifugal compressor but it's just one type among the others. The compressors can be divided in two main categories: a positive displacement and a continuous flow. The positive displacement is like, for example, an piston engine in which a successive volume of fluid is confined in a closed space to increase its pressure. The highest pressures are achieved by using the positive displacement compressors but these are not good with the high flows. The continuous flow, as its name predicts, doesn't have the same kind of syclic flow but its flow is *continuous*. However, the positive displacement method can also be like the continuous flow if the cycles are fast enough. The continuous flow compressors can have high flows but quite low pressure ratios. (Boyce, 2003, p. 2)

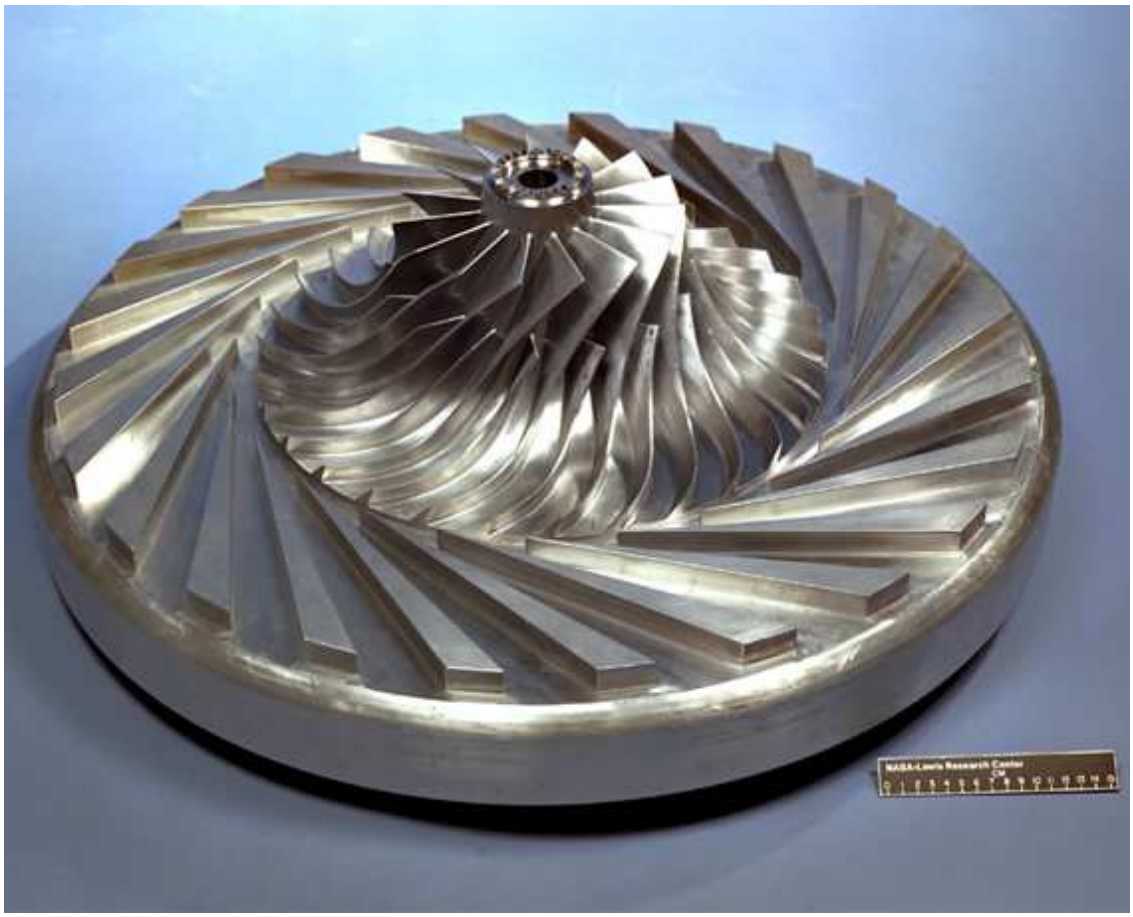


Figure 1: CC-3 impeller and a wedge diffuser. NASA Glenn Research Center (2007)

2.2 The centrifugal compressor

The centrifugal compressor impeller and a wedge diffuser can be seen in the figure 1. This type of compressor is the continuous flow compressor and that's well-known for its smooth operation, large tolerance of process fluctuations and especially for its high reliability. The range in size varies from pressure ratios of 1.3:1 per stage in the process industries, to 3-7 per stage in small gas turbines. On experimental models, the pressure ratio can even be as high as 13:1. With the pressure ratios higher than 5:1, a special kind of diffuser is needed because of the supersonic flow (Mach Number greater than 1). The operating range which is considered as a stable varies from 45 % to 90 % of the rated capacity. The operating speeds of these compressors can be extremely high. For aircraft applications, for example, the rpm range can be 50 000 - 100 000 but most units run below 20 000. (Boyce, 2003, p. 8-9)

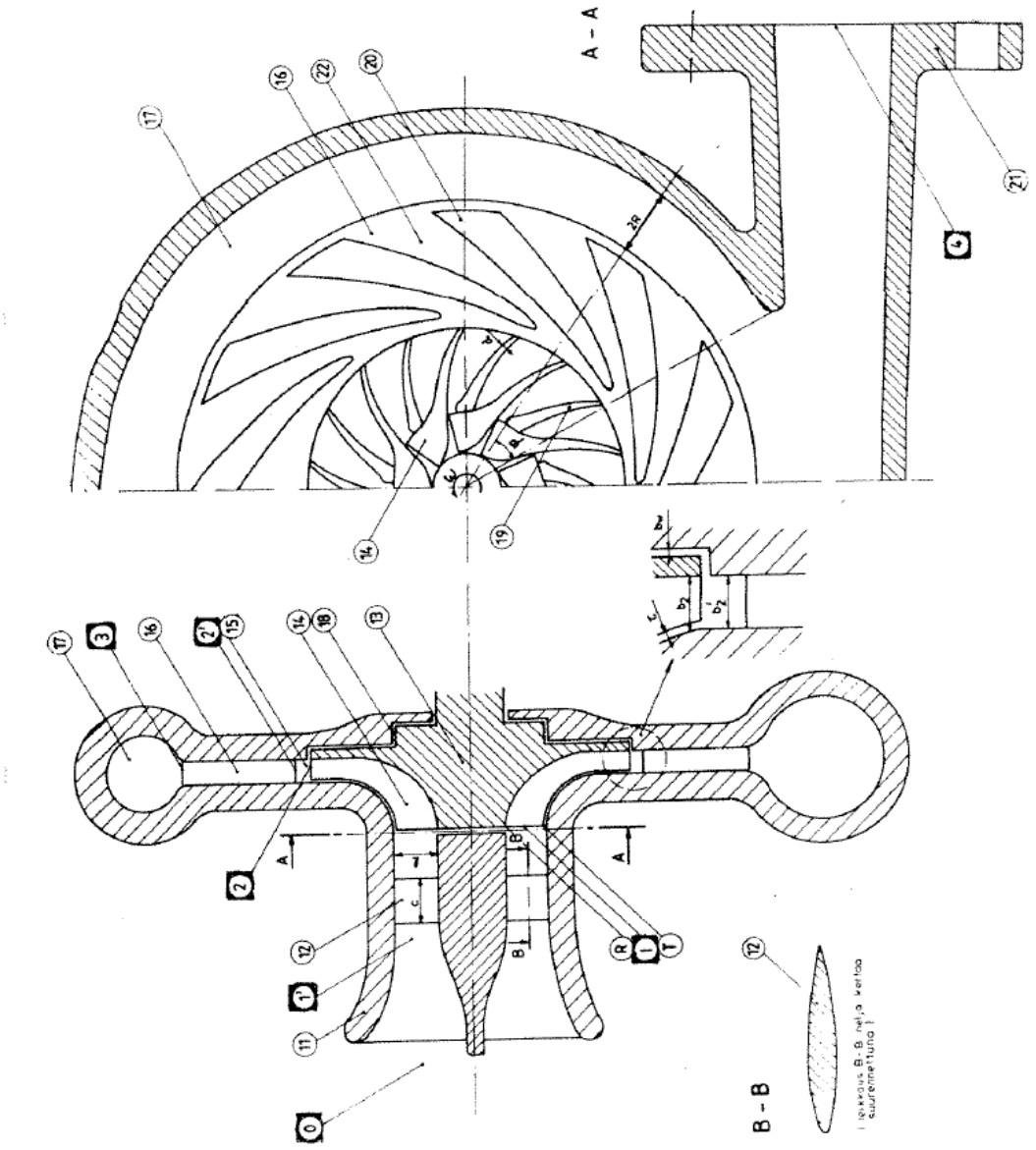


Figure 2: The main parts of a centrifugal compressor. Larjola (1988)

Inside the compressor impeller and diffuser, the velocity of the fluid is converted to the pressure. The diffuser consists essentially of vanes which are tangential to the impeller and normally the compressor is designed so that the half of the pressure rise takes place in the impeller and the other half in the diffuser. (Boyce, 2003, p. 9)

The main parts of a centrifugal compressor is presented in the figure 2. The main parts are an inlet duck (11), an impeller (13) and a diffuser (16). (Larjola, 1988, p. 7)

The most of the kinetic energy is converted into the pressure energy inside the diffuser. This can be seen in the figure 3. As just mentioned above, the diffuser consists essentially of vanes and these vane passages diverge to convert the velocity head into the pressure energy. As shown in the figure 4, the inner edge of the vanes is in line with the direction of the resultant fluid flow from the impeller. (Boyce, 2003, p. 9)

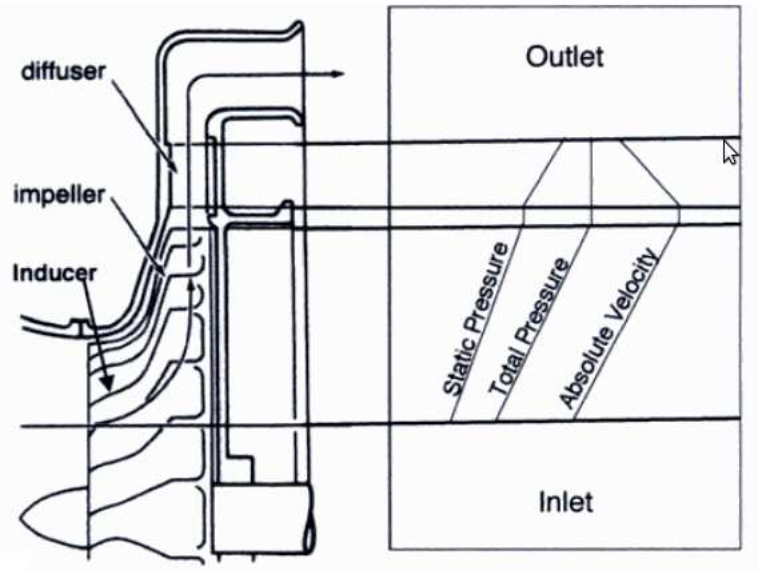


Figure 3: Aerodynamic and thermodynamic properties of the fluid in a centrifugal compressor stage. Boyce (2003)

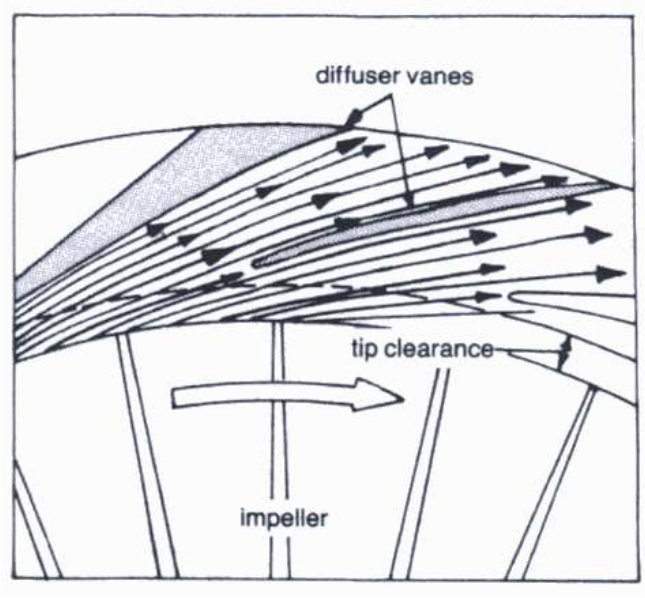


Figure 4: The flow in a vaned diffuser. Boyce (2003)

2.2.1 Types of impellers

There are three types of impeller blades used: radial blades, backward curved blades and forward curved blades. These are defined (Boyce, 2003, p. 11) according to the exit blade angles

$$\beta_2 = 90°\mathrm{C} \Rightarrow Radial\,blade\,, \tag{1}$$

$$\beta_2 < 90°\mathrm{C} \Rightarrow Backward\,blade\,, \tag{2}$$

$$\beta_2 > 90°\mathrm{C} \Rightarrow Forward\,blade\,. \tag{3}$$

The backward curved impeller is the most common since they have a low outlet kinetic energy, a low diffuser inlet Mach number, and the surge margin is the widest when compared to the radial and forward curved impellers. On the other hand, the backward curved impeller has some disadvantages such as a low energy transfer, a complex bending stress, and difficulties in manufacturing. By contrast, the radial curved impeller is easy to manufacture and it is also a reasonable compromise between the low energy transfer and the high absolute outlet velocity. As a disadvantage, the radial curved impeller surge margin is narrow. (Boyce, 2003, p. 12)

The forward curved impeller is relatively rare but its main advantage is a high energy transfer capability. (Boyce, 2003, p. 12)

2.2.2 Velocity triangles

In the figure 5, the velocity triangles can be seen. In this thesis, we are only interested in the real geometrical angle of the blade. We called this angle *a gamma*. The gamma distribution (see 4.2.5) tells us what is the current angle as a function of a meridional lenght.

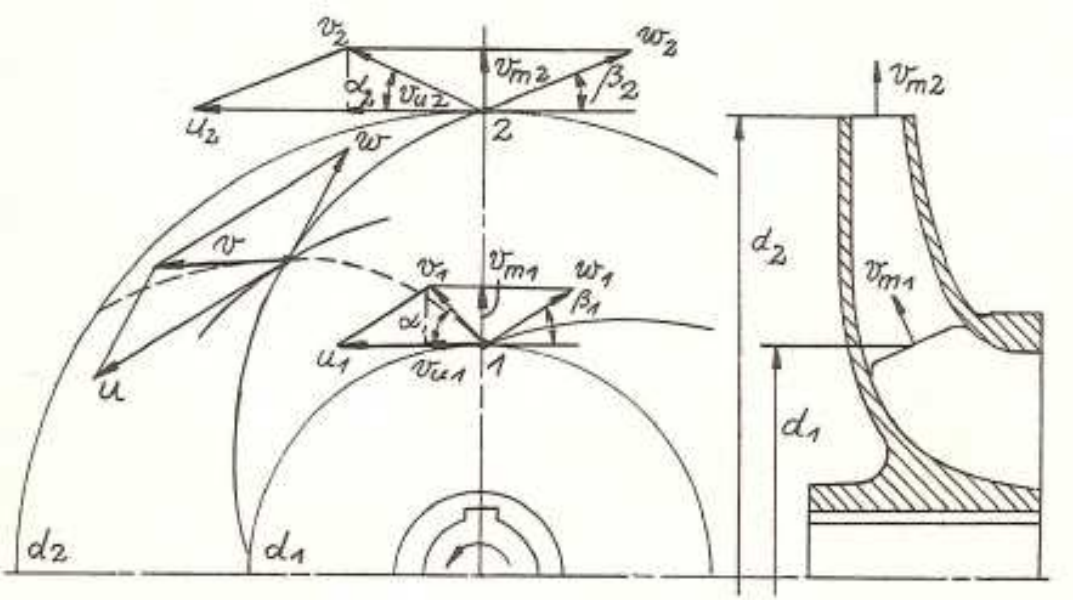


Figure 5: Velocity triangles. Wirzenius (1978)

3 COMPUTER GRAPHICS

There is a long history of a technical drawing and also designing methods in general. There was no mathematics involved in the earliest drawing methods. In Italy, there were reneissance naval architects known for using drafting techniques that involved conic sections. These desing techniques led to the use of *splines*. The splines are basically wooden beams bent into the optimal shapes. With using these, it was possible to draw very smooth lines that are well suitable in naval architectures. (Farin, 2002, Preface)

In the twentieth century, aiplanes and cars made their appearance. This was a huge improvement also in designing methods because the need for smooth lines became more important. This was strictly linked to aerodynamics but also, especially in cars, to an artistic design. The computer graphics today is so evolved that the whole airplane or a car, just to mention a few, can be modelled virtually. In this thesis, we will learn what are the building blocks of these very large structures.

In this thesis, author's opinion concerning about the learning methods of the computer graphics is that drawing techniques performed with the *hardware* rather than the software is much more informative. The hardware in this context means, for example, these wooden splines as described above and has nothing to do with the computer hardware that performs calculations and such things. Of course, this whole hardware thing can be considered at a conceptual level only.

3.1 Bézier curve

In this thesis, the Bézier curve deserves only a quick overview. Bézier curves can be considered as a special case of the B-spline which is in-depth introduced in section 3.2.

Farin (2002, p. 437) defines the Bézier curve as:

> *"A polynomial curve that is expressed in terms of Bernstein polynomials."*

These Bézier curves are widely used in the computer graphics and can represent smooth curves which are scalable and exclusively defined. The simplest curve is a straight line between the points P_0 and P_1 and is called a linear Bézier curve. The higher the count

of points, the higher the order. Usually it's not convenient to use very high order Bézier curves but use B-splines instead.

The Bézier curves were invented in the late 50s and early 60s. Paul de Casteljau and Pierre E. Bézier discovered them simultaneuously, latter being more credited. These are parametric curves and contains some limitations like an unability to represent some simple curves such as circles. To define a Bézier curve of degree n, we need to choose n+1 control points so that they indicate the shape of the desired curve. As one or more control points are moved, the shape of the curve changes. (Shene, 2008, Unit 5)

The construction of the Bézier curves according to Shene (2008, Unit 5): given n+1 points $P_0, P_1, P_2, P_3, ..., P_n$ in space which are the *control points* and the Bézier curve defined by these control points is

$$C(u) = \sum_{i=0}^{n} B_{n,i}(u) P_i \ , \tag{4}$$

where the coefficients are defined as

$$B_{n,i}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-1} \ . \tag{5}$$

The point that corresponds to u on the Bézier curve is the *weighted* average of all the control points, where the weights are the coefficient $B_{n,i}(u)$. This is referred as the ***Bézier basis functions*** or ***Bernstein polynomials***. The line segments $P_0P_1, P_1P_2, P_2P_3, ..., P_{n-1}P_n$ are called *legs* which form a control polyline. All the basis functions are non-negative and $C(u)$ pass through the first and the last control point (P_0 and P_n). The domain u is [0,1] but other values can also be used by converting them first. (Shene, 2008, Unit 5)

If the domain is [a,b], then the converting to [0,1] is simply

$$u = \frac{u-a}{b-a} \ . \tag{6}$$

All the properties of the Bézier curves are not perfectly covered in here. Although, some of those are worth a mention, like a convex hull property and a partition of unity. The first

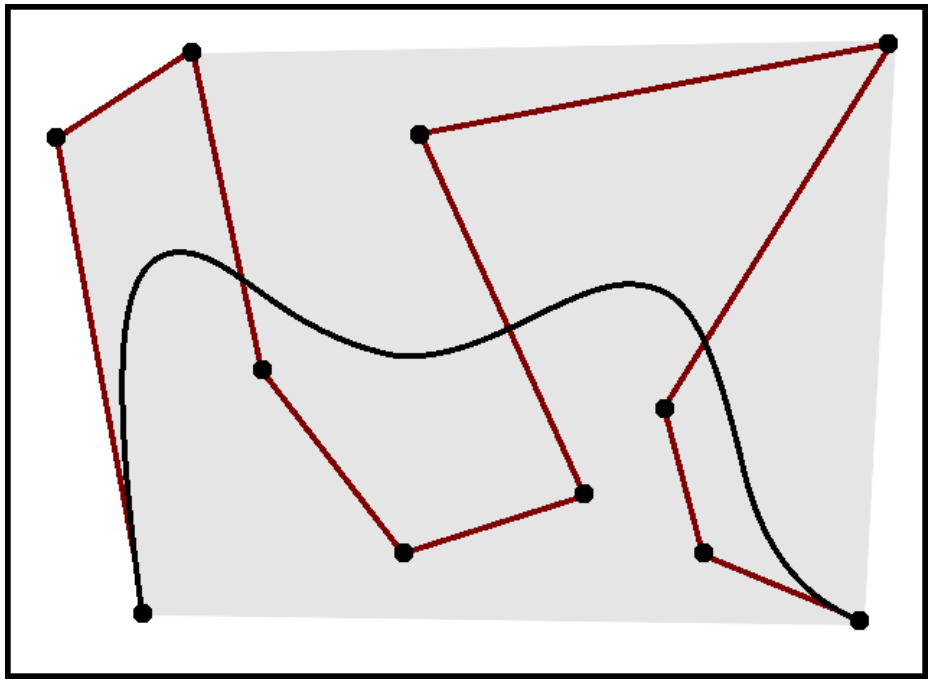


Figure 6: Convex Hull Property.

one means that the Bézier curve defined by the given n+1 control points lies completely in the convex hull of the given control points. The convex hull of a set of points is the smallest convex set that contains all thepoints and this is demonstated in the figure 6 where the convex hull of the 11 control points is shown in color gray. The curve will not go outside of this region and that's an important thing to know when determining limits. The second thing to mention, the partition of unity, means that the sum of basis functions at a fixed u is 1. (Shene, 2008, Unit 5)

3.2 B-spline

The B-spline is the essential part of the almost any modern graphics development performed by the computer. Therefore, the B-spline is also the essential part of the Curve-point's computing.

Farin (2002, p. 438) defines the B-spline as:

> *"A piecewise polynomial function. It is defined over a knot partition, has local support, and is nonnegative. If a spline curve is expressed in terms os B-splines, it is called a B-spline curve."*

The B-splines are the building blocks of the geometry object produced by Curvepoint. When compared to the Bézier curve, the B-spline can be difficult to understand. However, it's *possible* by using uncomplex figures and equations. In literature, there are many ways to explain the B-splines and this variety can be confusing. In this thesis, the simpliest methods are preferred.

Any B-spline can be converted in to one or more Bézier curves and vice versa. By default, the B-splines don't interpolate any of its control points whereas Bézier curves automatically clamps its endpoints. It is, however, possible *to force* the B-splines to interpolate any of its control points without repeating it. That's not possible with Bézier curves. (Andersson, 2003, p. 23)

3.2.1 A B-spline curve definition

The definition of the B-spline curve according to Shene (2008, Unit 6) is

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \ , \tag{7}$$

where $N_{i,p}$ are B-spline basis functions of degree p, $P_i = \{P_0, P_1, P_2, ..., P_n\}$ are control points and $U = \{u_0, u_1, u_2, ..., u_m\}$ is a knot vector. In addition, n, m and p must satisfy $m = n + p + 1$. The equation 7 is very similar to Bézier curve but involves more information. (Shene, 2008, Unit 6)

3.2.2 Cox-de Boor recursion formula

Shene (2008, Unit 6) defines B-spline basis function

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} , \tag{8}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \ . \tag{9}$$

In the figure 7, all knot spans are listed on the left column and all degree zero basis functions on the second. This triangular computation scheme helps to understand the way of computing $N_{i,p}(u)$ for p greater than 0. (Shene, 2008, Unit 6)

$[u_0, u_1]$	N0,0					
		N0,1				
$[u_1, u_2]$	N1,0		N0,2			
		N1,1		N0,3		
$[u_2, u_3]$	N2,0		N1,2		N0,4	
		N2,1		N1,3		N0,5
$[u_3, u_4]$	N3,0		N2,2		N1,4	
		N3,1		N2,3		
$[u_4, u_5]$	N4,0		N3,2			
		N4,1				
$[u_5, u_6]$	N5,0					

Figure 7: A triangular computation scheme for a Cox-de Boor recursion formula.

4 DESIGNING THE SOFTWARE

4.1 Programming language

Tomi Putus used MATLAB programming language when he wrote his software for his thesis. To run that software we need the commercial MATLAB program. In this thesis, we don't want the software being dependent on anything except the standard free libraries.

There's no the Programming Language which is the only one suitable in this project. Programming in this project is mostly putting mathematical equations to the form computer understand and can thereby calculate them. An efficiency isn't the main issue here because the calculations used in this project are performed relatively fast on a modern computer. The most important thing when selecting the programming language is that it's well known. Hence, others can read and understand the source code. C++ programming language is selected in this project because it meets all the criteria.

However, it's worth a mention that the author is already familiar with C++ programming language. That might be the real reason for the language selection for Curvepoint. A coding style is from Kerninghan and Ritchie (1988)

4.2 Curvepoint file formats

Curvepoint needs numerical data for defining the impeller geometry. This data are stored in files and are loaded by the program. In the open source world, it's convenient to use plain text files for clarity. It's also possible to use a custom format by using binary files. These would be very compact, resource friendly, and can be read fast by the computer. However, the problem is that the examining or modifying these settings by hand is very difficult for the human. Because of the open source nature of this project, it's clear that the plain text files are used for storing numerical data.

The loader which loads values from these plain text files isn't a *stand-alone* application but a subroutine. It's written in C++ using mostly a String library for parsing the data and it's designed to be as universal as possible.

4.2.1 .csv - Output of Centriflow

This format is not the author's design. Centriflow is 1D designing software for the centrifugal compressors and its result file is used by the Curvepoint. The file suffix is *.csv*.

This file contains pretty much everything that is needed for the centrifugal compressor geometry. However, some 3D geometry related information will be added to this. The loader for loading the important values from this file is more complex than loader for the *.crvp* or *.crvd* file format.

4.2.2 .crvd - Curvepoint custom data

This simple file format is the author's design. It contains all the values needed from *.csv* file so the user doesn't need to edit the *.csv* file at all. It also contains values needed for the three-dimensional geometry.

Here is an example of some *.crvd* file:

```
#CRVD_1.0

// full_centriflow.csv

N0 = 9
N2 = 18
d1t = 0.0932
d1h = 0.0326
d2 = 0.171
b = 0.0075

gamma1h = 25.2
gamma1t = 54
gamma2 = 40

// Settings

z1/d2 = 0.34
bladecut = 0.3
bladethick/d2 = 0.005

// Settings (default-meridional)

angle_hle = 0.34
angle_hte = 0.3
angle_tle = 0.34
angle_tte = 0.3

// EOF
```

Every *.crvd* must start with #CRVD_1.0. Comments are allowed with // like in C++.

4.2.3 .crvp - A geometry data created by Curvepoint

This relatively simple format is also the author's design. This *.crvp* file contains all the data needed for creating the 3D geometry and Curvepoint Viewer uses this file as an input. This will be converted to the IGES if needed. This format supports the B-splines and can also draw points using coordinates.

Here is an example of *.crvp* file:

```
#CRVP_1.0

[b-spline]
examplecurve1
[/b-spline]

[vertex]
examplevertex1
[/vertex]

examplecurve1.ctrl 0 = 0.0000 0.0000 0.0000
examplecurve1.ctrl 1 = 0.2500 0.0000 0.0000
examplecurve1.ctrl 2 = 0.7500 1.0000 0.0000
examplecurve1.ctrl 3 = 1.0000 1.0000 0.0000

examplecurve1.knot 0 = 0
examplecurve1.knot 1 = 0
examplecurve1.knot 2 = 0
examplecurve1.knot 3 = 0
examplecurve1.knot 4 = 1
examplecurve1.knot 5 = 1
examplecurve1.knot 6 = 1
examplecurve1.knot 7 = 1


examplevertex1.vertex 0 = +0.01630000 +0.05810000 +0.00000000
examplevertex1.vertex 1 = +0.01629999 +0.05810000 +0.00002048
examplevertex1.vertex 2 = +0.01629995 +0.05810000 +0.00004097
examplevertex1.vertex 3 = +0.01629988 +0.05810000 +0.00006145
examplevertex1.vertex 4 = +0.01629979 +0.05810000 +0.00008193
```

Every *.crvp* must start with #CRVP_1.0. Comments are allowed with // like in C++.

4.2.4 .crvm - A meridional projection data

This format is almost like the .crvp but the main difference is that it has no z-coordinate. It also has no [b-spline] section because for the meridional projection there must always be the *hub* and the *tip*. There are no limits for the B-spline so there can be an arbitrary amount of control points.

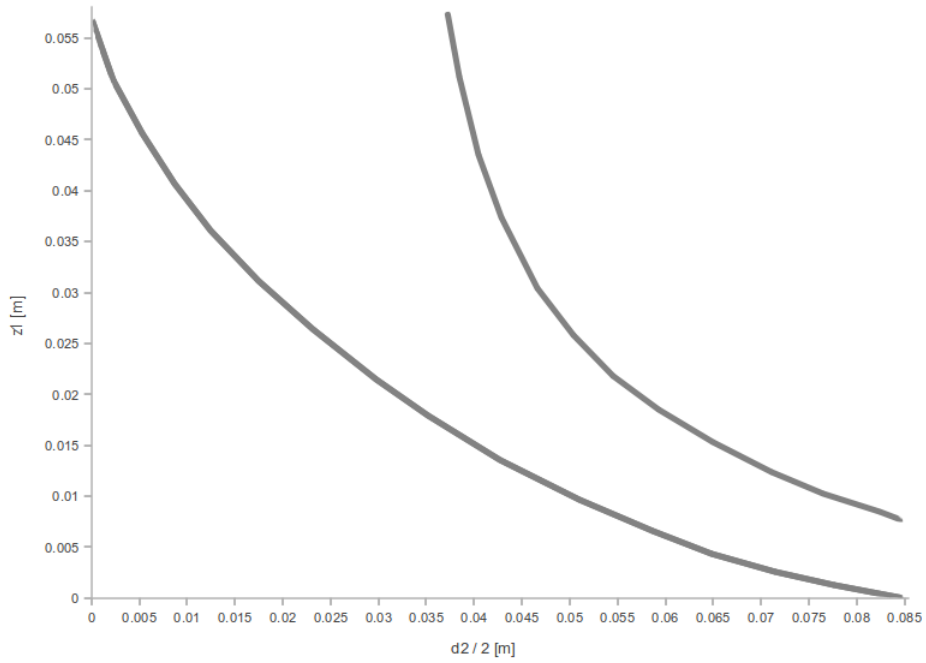


Figure 8: A meridional projection.

Here is a complete *.crvm* file and the source for the figure 8:

```
#CRVM_1.0

hub.ctrl 0 = 0.0163 0.0581
hub.ctrl 1 = 0.0223 0.0297
hub.ctrl 2 = 0.0567 0.0040
hub.ctrl 3 = 0.0855 0.0000

hub.knot 0 = 0
hub.knot 1 = 0
hub.knot 2 = 0
hub.knot 3 = 0
hub.knot 4 = 1
hub.knot 5 = 1
hub.knot 6 = 1
hub.knot 7 = 1

tip.ctrl 0 = 0.0466 0.0581
tip.ctrl 1 = 0.0506 0.0294
tip.ctrl 2 = 0.0582 0.0174
tip.ctrl 3 = 0.0855 0.0075

tip.knot 0 = 0
tip.knot 1 = 0
tip.knot 2 = 0
tip.knot 3 = 0
tip.knot 4 = 1
tip.knot 5 = 1
tip.knot 6 = 1
tip.knot 7 = 1
```

Every *.crvm* must start with #CRVM_1.0. Comments are allowed with // like in C++.

4.2.5 .crvg - A gamma distribution

This file format works just like the previus .crvm format. The only difference is the first line which is used to detect the file format. In the .crvg this line is *#CRVG_1.0*. An example of a gamma distribution can be seen in the figure 9.

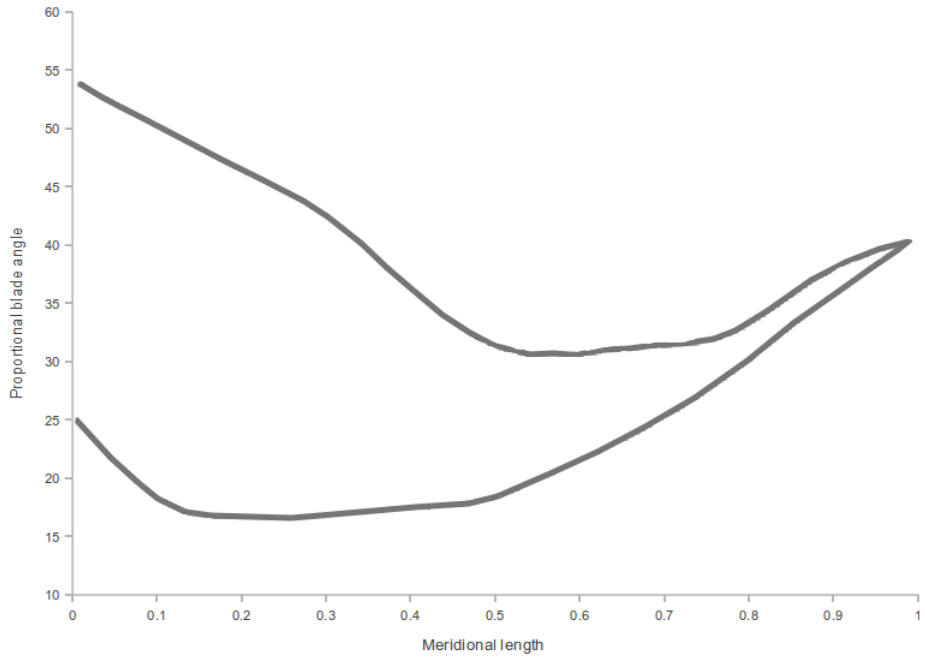


Figure 9: A gamma distribution.

The source for the figure 9 is not printed here. It works similar to the previous examples. However, every *.crvg* file must start with #CRVG_1.0. Comments are allowed with // like in C++.

4.3 IGES export file format

An export format of Curvepoint is the IGES. That's well known format by the CAD softwares so it's possible for them to view a geometry produced by Curvepoint.

4.3.1 History of IGES

In the 1979 when mechanical CAD systems were relatively young, the IGES project was started. Before that it was difficult to share data among the CAD users because there weren't any standards for CAD. Therefore, a group of users and vendors of CAD started to create the first national standard for the CAD data exchange. There were some meetings with CAD vendors, that included ComputerVision, Applicon and Gerber, and soon after, an open meeting was held at National Academy of Sciences on October 10, 1979. About 200 people attended to herald the birth of the IGES standard. (National Institute of Standards and Technology, 2010)

4.3.2 ASCII File format

There are fixed and compressed versions of ASCII file format. Also, there is a binary format but it's not used anymore. In this project, only the fixed one is used because it's obviously clearer than the compressed one. The IGES file consists of 80-column lines which are grouped into sections. That 80-column *limit* is derived from the punched card era. There are five sections S, G, D, P and T which will be described soon. Section-specific data is in the columns 1-72, an identifying letter code in the column 73, and an ascending sequence number in the columns 74-80. All of these are numbers start at 1 and are incremented by 1 for each line. Columns 73 and 74-80 could be considered as a special kind of line number. (U.S. Product Data Association, 1996, p. 9)

This format is very strictly and therefore one error in the file *could* ruin the whole file. When designing the Curvepoint exporting feature, one must be extra careful.

4.3.3 Sections S, G, D, P and T

The first two sections are called Start Section (S) and Global Section (G). Start Section has been reserved for human-readable prologue to the file and it contains one or more lines. There is only one data field in the columns 1-72. Start Section basically contains comments and other information. Global Section is relatively complex and definitely isn't *normal*-human-readable until getting familiar with the IGES standard. It contains information describing the preprocessor and some information needed by the postprocessor to handle the IGES formatted file. There are also text strings in this section like an author name which is represented as "7HTIIHALA". The number before the letter "H" means the number of characters in the string. (U.S. Product Data Association, 1996, p. 16)

In a section called Directory Entry (D), one line has been divided into ten fields and two lines form an entity. Therefore, the entity consists of 20 fields. In the columns 1-72, every field length is eight characters and those are right-justified. (U.S. Product Data Association, 1996, p. 23)

The next section is Parameter Data Section (P). This contains the data associated with the each entity. It's not meaningful here to describe this section precisely but it's worth a mention that the data is free-formatted. This section contains also most of the information about the geometry. (U.S. Product Data Association, 1996, p. 32)

The last section is Terminate Section (T). There's only one line here and it is the last sequenced line of the file. (U.S. Product Data Association, 1996, p. 34)

5 PROGRAMMING THE SOFTWARE

In this section, the programming methods are explained. First thing to do is to program the loaders that can load the values from the text files described in the previous section. After that, the Cox-de Boor recursion formula is implemented in C++. Then we move on defining the three-dimensional geometry and saving that information to the *.crvp* file (see section 4.2.3). Finally the *.crvp* is converted to IGES. There is also a program called Curvepoint Viewer which can be used for viewing the three-dimensional geometry but its details are not revealed here because it's an optional software (see section 6.4).

The list of the source code files can be found in appendix 1.

5.1 Loaders

All the loaders source code can be found in *loader.cc*. Parsing methods are quite simple and C++ Strings library is primarily used for them. The format of the files where the values are loaded is simple: $parameter = value$. As an exception of this, the Centriflow file needs more advanced method for the parsing.

As an example, we can see an excerpt of the source code from *loader.cc* in the figure 10. It loads values from a *.crvd* file. This is probably the simplest method to load values ($parameter = value$) from the text file. The return vector in this example function contains all the values found in the text file and their validity is checked later.

5.2 Cox-de Boor C++ implementation

The source code for the Cox-de Boor C++ implementation can be found in *coxdeboor.cc*. It might be difficult to understand the code at a first glance but it's quite clear when examined in depth. The Cox-de Boor is well explained in the section 3.2.2.

```
/*
 *  This can be found in csvdata.hh
 *
 *  class Crvddata
 *  {
 *     public:
 *     std::string param;
 *     float value;
 *  };
 */

std::vector<Crvddata> load_from_crvd(std::string file)
{
        std::vector<Crvddata> return_data; // Return values pushed to vector
        Crvddata temp_data;
        std::ifstream filestr;
        std::string tmpstr, delimiter, parameter;
        double tmpdouble;

        filestr.open(file.c_str());
        if (!filestr.is_open()) {  // Handling the error
                throw (static_cast<std::string>("\nUnable to open crvd file.\n"));
        }
        filestr >> tmpstr;
        if (tmpstr != "#CRVD_1.0") // Check the header of the file
                file_format_error(); // Handles the error

        while (1) {
                tmpstr.clear();
                delimiter.clear();
                parameter.clear();
                filestr >> tmpstr;
                if (filestr.eof()) break; // Stop if EOF occurs
                if (tmpstr == "//") { // Handling C++ comments
                        getline(filestr, tmpstr); // Skip the line
                        tmpstr.clear();
                        continue;
                }
                parameter = tmpstr;
                filestr >> delimiter;
                if (delimiter != "=") error_in_file(); // Must be equal sign
                if (!(filestr >> tmpdouble)) error_in_file(); // Error if the value is not float
                temp_data.param = tmpstr; // Parameter (string)
                temp_data.value = tmpdouble; // Number (float)
                return_data.push_back(temp_data); // Value is pushed to return vector
        }
        filestr.close();
        return return_data;
}
```

Figure 10: An excerpt of the source code from *loader.cc*.

5.3 Defining the geometry

The source code in the file *convert.cc* contains mainly all the code for calculating the three-dimensional geometry. In addition to this, *thick.cc* contains the code for defining the thickness of the blades. These two routines are the most important part of this software. The thickness distribution was quite hard to implement. There should be an easier way for that but the author didn't have enough time to reimplement this feature.

5.4 Converting to IGES

The programming process for *iges.cc* was straightforward but required a careful study of the IGES standard. It uses only one IGES entity, Linear Path entity (Type 106, Form 12). It's enough for drawing the whole three-dimensional geometry of the impeller.

The IGES files have fixed names: IMPELLER.IGS and IMPLSEGM.IGS. Latter shows only one segment of the impeller.

5.5 Miscellaneous

A program for plotting the B-splines, *curvepoin-plot*, uses functions from *coxdeboot.cc* and *loader.cc*. It is the simplest program and the source code is easy to read.

The header files *bsplinecurve.hh* and *csvdata.hh*, and *vertex.hh* are stand-alone files containing C++ classes. The *csvdata.hh* is the most important. It name refers just to *.csv* file but it contains also other information needed for the geometry.

6 USING THE SOFTWARE

This section can be considered as a manual for the Curvepoint.

6.1 Program: curvepoint

This is the main program of the Curvepoint software. It performs the calculations needed for the three-dimensional geometry.

It is takes four files, *.crvd*, *.crvm*, *.crvg*, and *.crvt* as command line parameters. After the processing, it outputs the *.crvp* and IGES files. IGES files are what we are looking for when using the Curvepoint software. In appendix 2, there are some pictures of IGES opened in SolidWorks.

6.2 Program: curvepoint-init

This program is used at first. It takes one command line parameter which can either be the *.csv* or *.crvd*. If the *.csv* file is used as the parameter, *curvepoint-init*'s output is *.crvd* file. If the *.crvd* file is used as the parameter, *curvepoint-init* outputs three files: *.crvm*, *.crvg*, and *.crvt*.

The program, *curvepoint-init*, is therefore used for generating default values for all the necessary files. It is not needed if these files are written by hand or generated by some another program.

6.3 Program: curvepoint-plot

This program is used when plotting the B-splines found in *.crvm*, *.crvg*, and *.crvt*. It takes one of these files as a command line parameter and outputs simple file containing the xy-coordinates. These coordinates can be plotted to a curve by using, for example, Gnuplot program.

6.4 Program: curvepoint-viewer

This program is used for viewing the three-dimensional geometry. It takes *.crvp* file as a command line parameter.

This program is not an essential part of the Curvepoint software. It works only in X11 desktop environment and uses OpenGL drawing functions. It was mainly used when writing draw functions. In figure 11 is a screenshot of this program.

This program is, however, very interesting if programming and X11 desktop environment are concerned because it uses Xlib and OpenGL libraries directly. Usually, higher level libraries like GLUT or SDL are used when programming OpenGL.

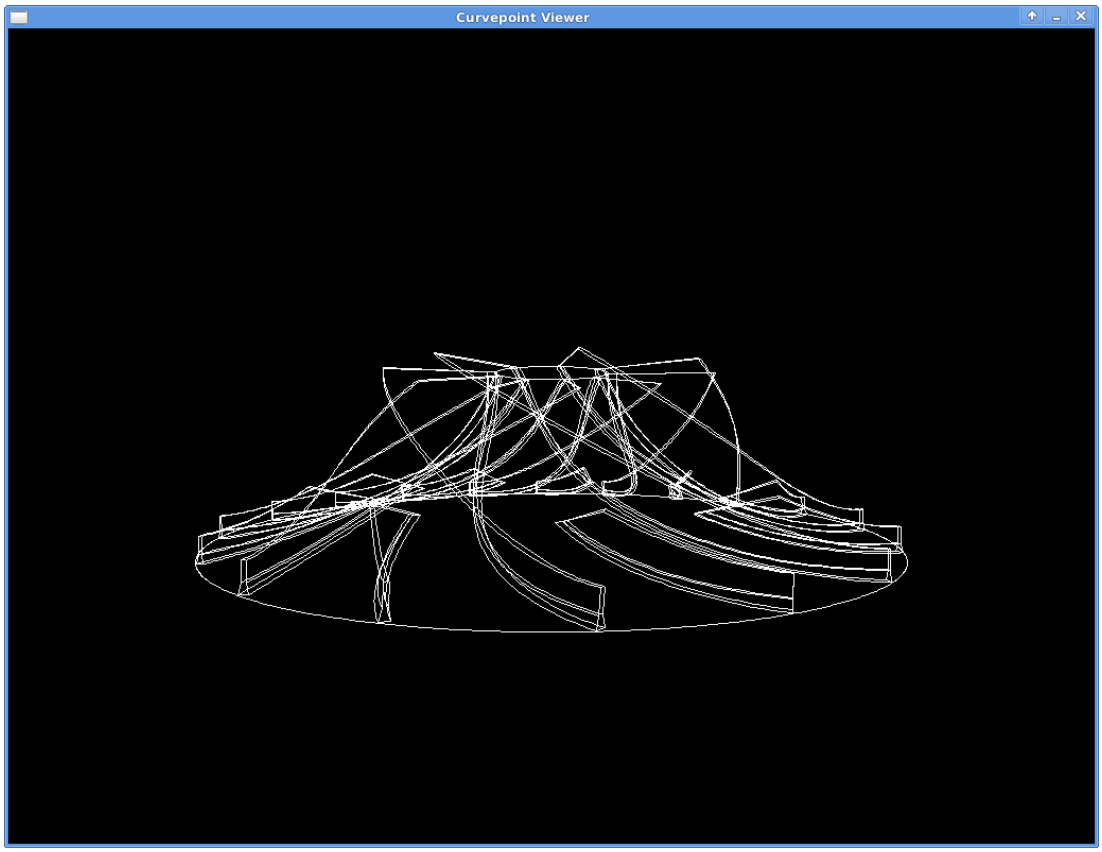


Figure 11: Curvepoint Viewer and an impeller.

7 CONCLUSIONS

The goal of this thesis was to develop and implement new computer software for defining the three-dimensional geometry for a centrifugal compressor impeller. In general, that goal was reached but there are still some room for further developing. The main problem is that the software is more complex than it should be. An object-oriented programming should have been used more efficiently.

Defining the three-dimensional geometry for an impeller is quite difficult. Especially defining the thickness of the blades became a big problem. In theory, it's quite easy but the method used here was not easy to implement. There are easier ways which should be used when developing this software further.

References

Andersson, F. (2003). *Bezier and B-spline Technology*. [Retrieved 5 July, 2010], url: http://www8.cs.umu.se/education/examina/Rapporter/461.pdf.

Boyce, M.P. (2003). *Centrifugal compressors: a basic guide*. PennWell Corporation. ISBN 0-87814-801-9.

Farin, G. (2002). *Curves and surfaces for CAGD: a practical guide*. Morgan Kauffmann Publishers. ISBN 1-55860-737-4.

Kerninghan, B. and Ritchie, D. (1988). *The C Programming Language, Second Edition*. Prentice Hall, Inc. ISBN 0-13-110370-9.

Larjola, J. (1988). *Radiaalikompressorin suunnittelun perusteet*. Aalef Oy Lappeenranta. ISBN 951-763-505-2.

NASA Glenn Research Center (2007). *CE18 Small Compressor Test Facility*. [Retrieved 6 July, 2010], url: http://www.grc.nasa.gov/WWW/5810/ce18.htm.

National Institute of Standards and Technology (2010). *Brief History of IGES*. [Retrieved 10 June, 2010], url: http://ts.nist.gov/standards/iges/about.cfm.

Shene, C.K. (2008). *Introduction to Computing with Geometry*. [Retrieved 1 July, 2010], url: http://www.cs.mtu.edu/∼shene/COURSES/cs3621/NOTES.

U.S. Product Data Association (1996). *Initial Graphics Exchange Specification (IGES 5.3)*. [Retrieved 10 June, 2010], url: http://www.uspro.org/documents/IGES5-3_forDownload.pdf.

Wirzenius, A. (1978). *Keskipakopumput*. Kustannusyhtymä. Tampere.

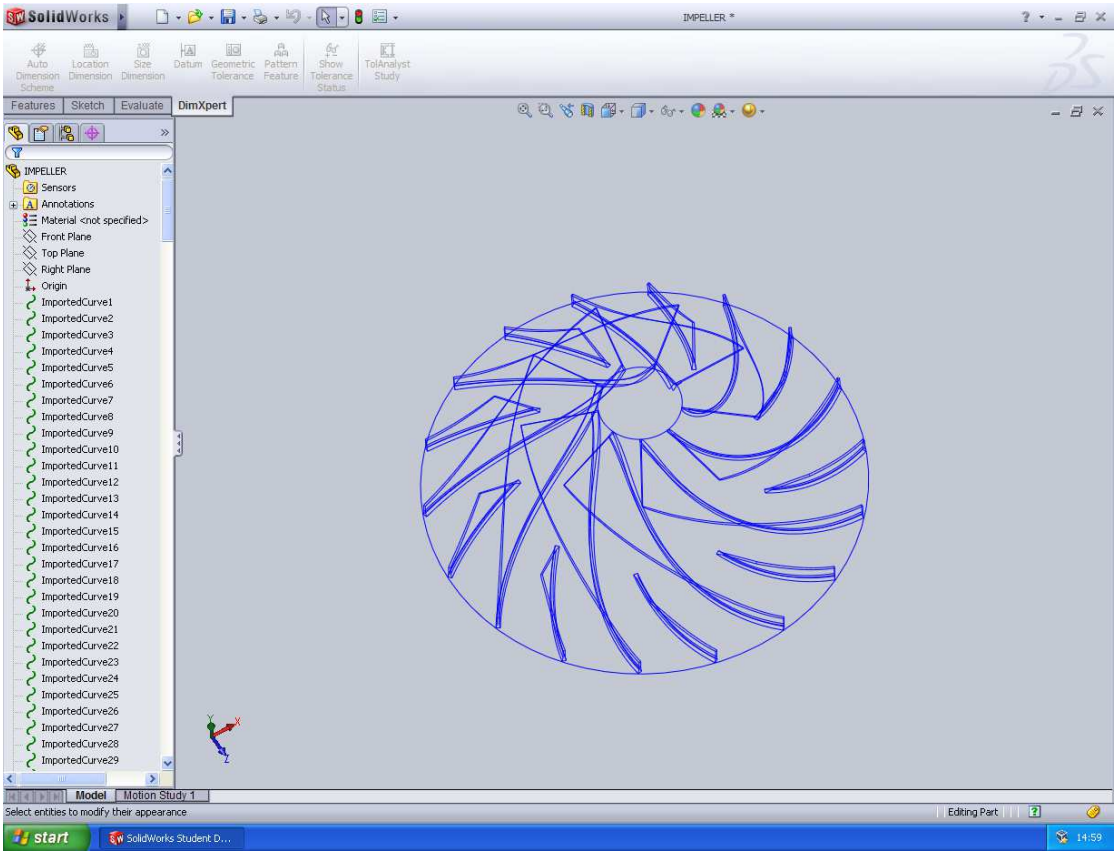
APPENDIX 1: Curvepoint source code files

The official version of Curvepoint 1.0
MD5 (curvepoint-1.0.tar.gz) = 1754d51eef0f9e0bc7ee3e6d28082c44

lines	*filename*	
64	bsplinecurve.hh	
55	bspliney.cc	
13	bspliney.hh	
157	checkdata.cc	
16	checkdata.hh	
991	convert.cc	
18	convert.hh	
102	coxdeboor.cc	
15	coxdeboor.hh	
90	csvdata.hh	
63	draw.cc	
17	draw.hh	
342	iges.cc	
12	iges.hh	
346	init.cc	- main function for *curvepoint-init*
584	loader.cc	
50	loader.hh	
59	main.cc	- main function for *curvepoint*
134	plot.cc	- main function for *curvepoint-plot*
331	thick.cc	
31	thick.hh	
26	vertex.hh	
183	viewer.cc	- main function for *curvepoint-viewer*

total lines: **3699**

APPENDIX 2.1: IGES opened in SolidWorks (IMPELLER.IGS)



APPENDIX 2.2: IGES opened in SolidWorks (IMPESEGM.IGS)