The TIGL Geometry Library

German Aerospace Center

Simulation and Software Technology (SC)

Distributed Systems and Component Software

Version 2012 / 11

Authors: Arne Bachmann, Markus Litz
Martin Siggel, Markus Kunde
Date: June 16, 2014

DLR Deutsches Zentrum für Luft- und Raumfahrt

Contents

1 The TIGL Geometry Library **3**
1.1 What is TIGL 3
1.2 TIGLViewer 4

2 Installation **5**

3 Using TIGL **6**
3.1 Usage Example 6
3.1.1 Open a CPACS configuration 6
3.1.2 Querying the geometrical structure 6
3.1.3 Example how to use TIGL out of python scripts 6

A Theory **8**
A.1 Segment coordinate transform 8
A.1.1 Parametrization of the chord surface 8
A.1.2 Projecting a point onto the chord surface 9
A.2 Component segment coordinate transform 10
A.2.1 Extending leading and trailing edges 11
A.2.2 Calculating η values of the corners 11
A.2.3 Calculating 3D coordinates of the (η, ξ) pair 12

Chapter 1

The TIGL Geometry Library

1.1 What is TIGL

In order to perform the modeling of wings and fuselages as well as the computation of surface points effectively, a geometry library was developed in C++. The library provides external interfaces for C and FORTRAN. Some of the requirements of the library were:

- Ability to read and process the information stored in a CPACS file for wings and fuselages,
- Possibility to extend to engine pods, landing gear and other geometrical characteristics, e.g.,
- Ability to build up the three-dimensional airplane geometry for further processing,
- Ability to compute surface points in cartesian coordinates by using parameters such as ξ, ζ, η, segment number,
- Possibility to be expanded by additional functions such as area or volume computations,
- Possibility to export the airplane geometry in the IGES format.

The developed library uses the Open Source software OpenCASCADE to represent the airplane geometry by B-spline surfaces in order to compute surface points and also to export the geometry in the IGES format. OpenCASCADE is a development platform written in C++ for CAD, CAM, and CAE applications which has been continuously developed for more than ten years. The functionality covers geometrical primitives (for example points, vectors, matrix operations), the computation of B-spline surfaces and boolean operations on volume models.

Apart from the already specified requirements above, the geometry library offers query functions for the geometry structure. These functions can be used

for example to detect how many segments are attached to a certain segment, which indices these segments have, or how many wings and fuselages the current airplane configuration contains. This functionality is necessary because not only the modeling of simple wings or fuselages but also the description of quite complicated structures with branches or flaps is targeted.

1.2 TIGLViewer

In order to review the geometry information of the central data set a visualization tool, TIGLViewer, was developed. The TIGLViewer allows the visualization of the used airfoils and fuselage profiles as well as of the surfaces and the entire airplane model. Furthermore, the TIGLViewer can be used to validate and test the implemented functions of the geometry library, for example the calculation of points on the surface or other functions to check data that belong to the geometry structure.

Chapter 2

Installation

First you need to install the most recent version of Open Cascade to your computer. You could find a direct download link in the CPACSIntegration-Teamsite:

`http://sites.kp.dlr.de/sc/CPACSIntegration/default.aspx`

Please be sure to answer "'yes"' when asked if the open cascade setup should set the environment variables. The *$PATH* variable is extended and a new environment variable *$CASROOT* is created which points to the OpenCASCADE installation directory.

Since OpenCASCADE 6.3.0 there is no longer a installer for Linux. For most distributions exist native installation packages (*.deb or *.rpm format).

Chapter 3

Using TIGL

3.1 Usage Example

3.1.1 Open a CPACS configuration

This is how to open a CPACS configuration. Please mention that first *TIXI* has to be used to open a CPACS configuration.

3.1.2 Querying the geometrical structure

Below you find some example how to query the geometrical structure of a CPACS configuration.

3.1.3 Example how to use TIGL out of python scripts

This is a small sample script that does nothing more than opening an CPACS data set and exporting it as an iges file.

```
from ctypes import *
from os import *

# define handles
cpacsHandle = c_int(0)
tixiHandle = c_int(0)
filename   = "./cpacs_example.xml"
exportName = "./cpacs_example.iges"

# open TIXI and TIGL shared libraries
import sys
if sys.platform == 'win32':
    TIXI = cdll.TIXI
    TIGL = cdll.TIGL
else:
    TIXI = CDLL("libTIXI.so")
    TIGL = CDLL("libTIGL.so")
```

```
# Open a CPACS configuration file. First open the CPACS-XML file
# with TIXI to get a tixi handle and then use this handle to open
# and read the CPACS configuration.
tixiReturn = TIXI.tixiOpenDocument(filename, byref(tixiHandle))
if tixiReturn != 0:
    print 'Error: tixiOpenDocument failed for file: ' + filename
    exit(1)

tiglReturn = TIGL.tiglOpenCPACSConfiguration(tixiHandle, "VFW-614", byref(cpacsHandle))
if tiglReturn != 0:
    TIXI.tixiCloseDocument(tixiHandle)
    print "Error: tiglOpenCPACSConfiguration failed for file: " + filename
    exit(1)

#-----------------------------------
# Export CPACS geometry as IGES file.
#-----------------------------------
print "Exporting CPACS geometry as IGES file..."
TIGL.tiglExportIGES(cpacsHandle, exportName)
```

Appendix A

Theory

A.1 Segment coordinate transform

For the following calculations we define the following points of the wing segment: $\vec{p}_1$ is the innermost point of the leading edge, $\vec{p}_2$ the outermost point of the leading edge, $\vec{p}_3$ the innermost point of the trailing edge, and $\vec{p}_4$ the outermost point of the trailing edge.

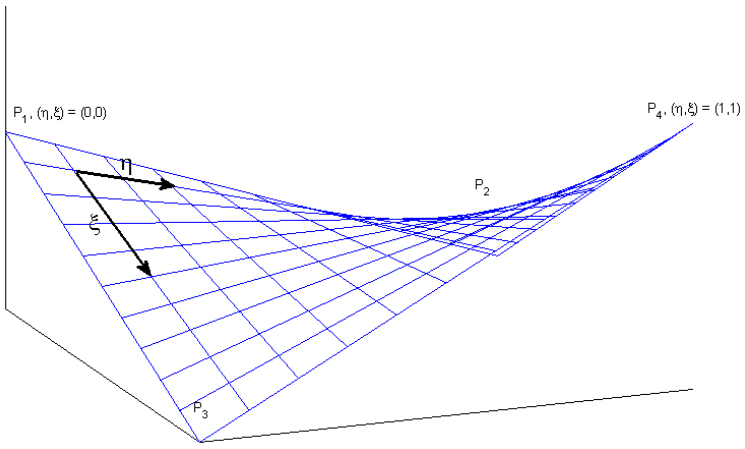

Figure A.1: Mathematically, a the chord face of a wing segment is a bilinear surface

A.1.1 Parametrization of the chord surface

For simplicity, lets rename $\alpha := \eta$, $\beta := \xi$. Each point $\vec{p}$ on the surface then be expressed by

$$\vec{p}(\alpha,\beta) = (\vec{p_1}(1-\alpha) + \vec{p_2}\alpha)\,(1-\beta) \tag{A.1}$$
$$+\,(\vec{p_3}(1-\alpha) + \vec{p_4}\alpha)\,\beta \tag{A.2}$$
$$= \alpha\,\underbrace{(-\vec{p_1}+\vec{p_2})}_{\vec{a}} + \beta\,\underbrace{(-\vec{p_1}+\vec{p_3})}_{\vec{b}} + \alpha\beta\,\underbrace{(\vec{p_1}-\vec{p_2}-\vec{p_3}+\vec{p_4})}_{\vec{c}} + \underbrace{\vec{p_1}}_{\vec{d}} \tag{A.3}$$

This formula defines the transformation between the segments coordinates (α, β) to cartesian coordinates (x, y, z). Mathematically, this form is called called bilinear surface. To sum up:

$$\begin{aligned}\vec{p}(\alpha,\beta) &= \alpha\vec{a} + \beta\vec{b} + \alpha\beta\vec{c} + \vec{d}, \quad \text{with} \\ \vec{a} &= -\vec{p_1} + \vec{p_2} \\ \vec{b} &= -\vec{p_1} + \vec{p_3} \\ \vec{c} &= \vec{p_1} - \vec{p_2} - \vec{p_3} + \vec{p_4} \\ \vec{d} &= \vec{p_1}\end{aligned} \tag{A.4}$$

A.1.2 Projecting a point onto the chord surface

The projection of a point $\vec{x}$ onto the surface is defined as the point $\vec{p}(\alpha, \beta)$ so that $\vec{p}(\alpha, \beta) - \vec{x}$ is orthogonal to the surface. Mathematically it can be shown, that the orthogonality requirement is equivalent to finding the point $\vec{p}(\alpha, \beta)$ that has a smallest distance to $\vec{x}$.

This is of course an optimization problem which can be defined as:

$$\min_{\alpha,\beta} f(\alpha,\beta) := \min_{\alpha,\beta} \|\vec{p}(\alpha,\beta) - \vec{x}\|^2 \tag{A.5}$$

This is an almost quadratic problem (it is only quadratic, if $\vec{c} = 0$) and can be thus solved with Newton's optimization method. Newtons method is an iterative procedure. In our case, we can adapt it as follows:

$(\alpha;\beta)_0 \leftarrow (0;0)$
$k \leftarrow 0$
while *not converged* **do**
  $(\alpha,\beta)_{k+1} \leftarrow (\alpha,\beta)_k - s[\nabla^2 f(\alpha,\beta)]^{-1} \cdot \vec{\nabla} f(\alpha,\beta), \quad \text{with } s \leq 1$
  $k \leftarrow k + 1$
end

Algorithm 1: Newton's optimization algorithm

In order to perform this projection, we need the gradient $\vec{\nabla} f(\alpha, \beta)$ and the hessian matrix $\nabla^2 f(\alpha, \beta)$, which are defined as the first and second order derivative of $f(\alpha, \beta)$.

Gradient Using the chain rule, we get for the gradient:

$$\vec{\nabla} f(\alpha,\beta) = 2J_p(\alpha,\beta)^T \cdot (\vec{p}(\alpha,\beta) - \vec{x}), \tag{A.6}$$

where $J_p(\alpha, \beta)$ is the Jacobian of $\vec{p}(\alpha, \beta)$. In components this is:

$$\begin{aligned}\frac{\partial f}{\partial \alpha}(\alpha, \beta) &= 2\left(\frac{\partial \vec{p}}{\partial \alpha}(\alpha, \beta)\right)^T (\vec{p}(\alpha, \beta) - \vec{x}) \\ &= 2\left(\vec{a} + \beta\vec{c}\right)^T (\vec{p}(\alpha, \beta) - \vec{x}) \end{aligned} \tag{A.7}$$

and

$$\begin{aligned}\frac{\partial f}{\partial \beta}(\alpha, \beta) &= 2\left(\frac{\partial \vec{p}}{\partial \beta}(\alpha, \beta)\right)^T (\vec{p}(\alpha, \beta) - \vec{x}) \\ &= 2\left(\vec{b} + \alpha\vec{c}\right)^T (\vec{p}(\alpha, \beta) - \vec{x}) \end{aligned} \tag{A.8}$$

Hessian A derivative of the gradient gives as the Hessian matrix:

$$\nabla^2 f(\alpha, \beta) = 2J_p(\alpha, \beta)^T J_p(\alpha, \beta) + 2(\nabla^2 \vec{p}(\alpha, \beta))^T \cdot (\vec{p}(\alpha, \beta) - \vec{x}) \tag{A.9}$$

Thus for the diagonal elements of the Hessian, we get

$$\frac{\partial^2 f}{\partial \alpha^2}(\alpha, \beta) = 2(\vec{a} + \beta\vec{c})^T(\vec{a} + \beta\vec{c}) = 2\|\vec{a} + \beta\vec{c}\|^2 \tag{A.10}$$

$$\frac{\partial^2 f}{\partial \beta^2}(\alpha, \beta) = 2(\vec{b} + \alpha\vec{c})^T(\vec{b} + \alpha\vec{c}) = 2\|\vec{b} + \alpha\vec{c}\|^2 \tag{A.11}$$

and for the off-diagonals

$$\frac{\partial^2 f}{\partial \alpha \partial \beta}(\alpha, \beta) = 2(\vec{a} + \beta\vec{c})^T(\vec{b} + \alpha\vec{c}) + 2(\vec{p}(\alpha, \beta) - \vec{x})^T\vec{c}. \tag{A.12}$$

Due to the symmetry of the Hessian, we get the same result for the other off-diagonal element $\frac{\partial^2 f}{\partial \beta \partial \alpha}(\alpha, \beta)$.

A.2 Component segment coordinate transform

The coordinate transform of component segment coordinates is somewhat more complicated than the segment coordinate transform. To simplify the procedure, the component segment should consist of only one segment for now. The segment is again defined by its corner points $\vec{p}_1 \ldots \vec{p}_4$. Figure A.2 should help understanding the forthcoming calculations.

First we need some definitions:

- Leading edge: $\vec{s}_v = \vec{p}_2 - \vec{p}_1$
- Trailing edge: $\vec{s}_h = \vec{p}_4 - \vec{p}_3$
- Projected leading edge: $\vec{n} = -\vec{s}_v$, with $n_x = 0$

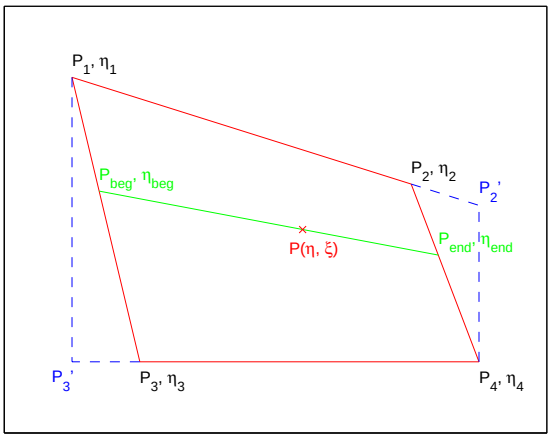

Figure A.2: Illustration for the calculation of the component segment coordinate transformation

A.2.1 Extending leading and trailing edges

Plane through $\vec{p}_4$ with normal vector $\vec{n}$. Calculate inersection with leading edge. Plane equation is:

$$(\vec{p} - \vec{p}_4) \cdot \vec{n} = 0 \tag{A.13}$$

Linear equation for the leading edge:

$$\vec{p} = \vec{p}_1 + \alpha(\vec{p}_2 - \vec{p}_1) \tag{A.14}$$

Inserting (A.14) into (A.13) yields:

$$\alpha_v = \frac{(\vec{p}_4 - \vec{p}_1) \cdot \vec{n}}{(\vec{p}_2 - \vec{p}_1) \cdot \vec{n}} \tag{A.15}$$

If $\alpha_v > 1$, the leading edge has to be extended, else the trailing edge must be extended. In the first case, we calculate the extended leading edge point $\vec{p}_2'$:

$$\vec{p}_2' = \vec{p}_1 + \alpha_v(\vec{p}_2 - \vec{p}_1) \tag{A.16}$$

In the other case ($\alpha_v < 1$), the intersection point with the trailing edge can be calculated in the same fashion.

Now, we apply the same method also to the inner section, getting the extended points $\vec{p}_1'$ and $\vec{p}_2'$.

A.2.2 Calculating η values of the corners

For the following calculations, we need to know the eta coordinates of the corner points $\vec{p}_1 \ldots \vec{p}_4$. Lets image a plane with the previously defined normal vector $\vec{n}$ that goes through one of these points. Without loss of generality, let this point be $\vec{p}_3$. This plane is then defined by the equation

$$(\vec{p} - \vec{p}_3) \cdot \vec{n} = 0 \tag{A.17}$$

Now we should find out, at which eta coordinate the plane intersects the leading edge, which is now parametrized as follows:

$$p = \vec{p}_1' + \eta(\vec{p}_2' - \vec{p}_1'). \tag{A.18}$$

Combining both equations leads to $\eta_3 = \frac{(\vec{p}_3 - \vec{p}_1') \cdot \vec{n}}{(\vec{p}_2' - \vec{p}_1') \cdot \vec{n}}$, or in general:

$$\eta_i = \frac{(\vec{p}_i - \vec{p}_1') \cdot \vec{n}}{(\vec{p}_2' - \vec{p}_1') \cdot \vec{n}}. \tag{A.19}$$

After doing these calculations (which have to be done only once), we can finally proceed to the coordinate transform.

A.2.3 Calculating 3D coordinates of the (η, ξ) pair

The easiest way to do the transformation is to move first in ξ direction. As the component segment is straight between the points $(\vec{p}_1, \vec{p}_3)$ and $(\vec{p}_2, \vec{p}_4)$ by definition, we move first along these lines:

$$\vec{p}_{beg}(\xi) = (1 - \xi)\vec{p}_1 + \xi\vec{p}_3 \tag{A.20}$$
$$\vec{p}_{end}(\xi) = (1 - \xi)\vec{p}_2 + \xi\vec{p}_4 \tag{A.21}$$

The corresponding eta coordinates of the points are

$$\eta_{beg}(\xi) = (1 - \xi)\eta_1 + \xi\eta_3 \tag{A.22}$$
$$\eta_{end}(\xi) = (1 - \xi)\eta_2 + \xi\eta_4 \tag{A.23}$$

Finally, we walk along the line defined by the new point $\vec{p}_{beg}$ and $\vec{p}_{end}$ (depicted green in figure A.2). We have to keep in mind their eta coordinates (which are not 0 and 1)! We get our final result

$$\vec{p}(\eta, \xi) = \vec{p}_{beg}(\xi) + \frac{\eta - \eta_{beg}(\xi)}{\eta_{end}(\xi) - \eta_{beg}(\xi)} \cdot (\vec{p}_{end}(\xi) - \vec{p}_{beg}(\xi)). \tag{A.24}$$