ENABLING RAPID CONCEPTUAL DESIGN USING GEOMETRY-BASED MULTI-FIDELITY MODELS IN VSP

A Thesis
presented to
the Faculty of California Polytechnic State University
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Aerospace Engineering

by
Joel B. Belben
March 2013

© 2013
Joel B. Belben
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	Enabling Rapid Conceptual Design Using Geometry-Based Multi-Fidelity Models in VSP
AUTHOR:	Joel B. Belben
DATE SUBMITTED:	March 2013
COMMITTEE CHAIR:	Rob A. McDonald, Ph.D., Associate Professor, Aerospace Engineering
COMMITTEE MEMBER:	Nick J. Brake, Staff Aeronautical Engineer, Lockheed Martin
COMMITTEE MEMBER:	Kurt Colvin, Ph.D., Professor, Industrial and Manufacturing Engineering
COMMITTEE MEMBER:	David D. Marshall, Ph.D., Associate Professor, Aerospace Engineering

ABSTRACT

Enabling Rapid Conceptual Design Using Geometry-Based Multi-Fidelity Models in VSP

Joel B. Belben

The purpose of this work is to help bridge the gap between aircraft conceptual design and analysis. Much work is needed, but distilling essential characteristics from a design and collecting them in an easily accessible format that is amenable to use by inexpensive analysis tools is a significant contribution to this goal. Toward that end, four types of reduced-fidelity or *degenerate* geometric representations have been defined and implemented in VSP, a parametric geometry modeler. The four types are degenerate surface, degenerate plate, degenerate stick, and degenerate point, corresponding to three-, two-, one-, and zero-dimensional representations of underlying geometry, respectively.

The information contained in these representations was targeted specifically at lifting line, vortex lattice, equivalent beam, and equivalent plate theories, with the idea that suitability for interface with these methods would imply suitability for use with many other analysis techniques. The ability to output this information in two plain text formats—comma separated value and Matlab script—has also been implemented in VSP, making it readily available for use.

A modified Cessna 182 wing created in VSP was used to test the suitability of degenerate geometry to interface with the four target analysis techniques. All four test cases were easily completed using the information contained in the degenerate geometric types, and similar techniques utilizing different degenerate geometries produced similar results.

The following work outlines the theoretical underpinnings of degenerate geometry and the fidelity-reduction process. It also describes in detail how the routines that create degenerate geometry were implemented in VSP and concludes with the analysis test cases, stating their results and comparing results among different techniques.

ACKNOWLEDGEMENTS

I would like to first and foremost express my sincere gratitude to my fiancée, Laurel Hammang. I don't know what I would have done without the love and support you've provided throughout my educational career. I only hope that I can repay at least a fraction of what you've given now that the shoe's on the other foot.

I also owe a special debt of gratitude to my advisor Dr. Rob McDonald for suggesting the topic, immeasurable help and guidance along the way, and trusting that I would get this done in a timely manner even though I was moving halfway across the country.

Contents

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**
1.1 A Primer on VSP . . . 2
1.2 Justification for Degenerate Geometry . . . 5
1.3 Target Analysis Types . . . 6
1.3.1 'Stick' Types . . . 7
1.3.2 'Plate' Types . . . 8
1.3.3 Additional Types . . . 10

2 Degenerate Geometry Definitions **11**
2.1 Degenerate Surface . . . 12
2.2 Degenerate Plate . . . 14
2.3 Degenerate Stick . . . 20
2.4 Degenerate Point . . . 26

3 Implementation in VSP **28**
3.1 Class Structure & Code definitions . . . 28
3.1.1 DegenSurface . . . 36
3.1.2 DegenPlate . . . 38
3.1.3 DegenStick . . . 43
3.1.4 DegenPoint . . . 52
3.2 Components Covered . . . 52
3.3 Output File Types . . . 54
3.3.1 CSV File . . . 55
3.3.2 M File . . . 58

4 Demonstration Cases, Results, and Conclusions **62**
4.1 Demonstration Cases . . . 62
4.1.1 Vortex Lattice: AVL . . . 63
4.1.2 Lifting Line Theory . . . 66
4.1.3 Equivalent Plate: ELAPS . . . 69
4.1.4 Equivalent Beam Theory . . . 71
4.2 Conclusions . . . 73

Bibliography **75**

A Matlab Scripts **78**

B Target Analysis Input Files **85**

List of Tables

3.1 VSP components by category. 52

4.1 Comparison of key quantities predicted by AVL and lifting line theory. 67
4.2 Material properties used for ELAPS test case. 70
4.3 Comparison of ELAPS-calculated component properties with degenerate point. 70

List of Figures

1.1 VSP design parameter groups allow quick and meaningful modification of component geometry. . . . 3
1.2 Visualization of wing plate and stick representations. . . . 7

2.1 Cessna 182 model showing body (blue) and surface (red) type components. . 12
2.2 Cirrus SR22 model, showing degenerate surface representation. . . . 13
2.3 An example degenerate surface normal vector on a wing section. . . . 14
2.4 Mapping between discretized surface points and degenerate plate points for a wing section. . . . 15
2.5 Degenerate Plate attribute definition using an airfoil section. . . . 15
2.6 Transformation from body component to degenerate plate using a right circular cylinder. . . . 17
2.7 Cessna 182 degenerate plate three view. . . . 19
2.8 Degenerate stick model of Boeing 747 wing. . . . 21
2.9 Shell representation of an airfoil section using rectangles to model thickness. . 23
2.10 Transformation from body component to degenerate stick using a right circular cylinder. . . . 26

3.1 An overview of the relationship between the Vehicle class and component geometries. . . . 29
3.2 Composition of the DegenGeom class in VSP. . . . 30
3.3 A leading edge view of a wing tip with rounded end cap. Cross-sections are shown in different colors. . . . 31
3.4 Sequence for creating degenerate geometry from the VSP main screen. . . . 31
3.5 VSP's internal degenerate geometry creation process. . . . 32
3.6 VSP unreflected geometry storage ordering for both body and surface component types. . . . 33
3.7 Point and cross-section ordering for different types of reflected symmetry using a Fuselage 2 VSP component. . . . 34
3.8 An example degenerate surface normal vector on a wing section. . . . 37
3.9 Ordering of degenerate surface nodes and computation of camber line points on an airfoil section. . . . 39
3.10 Point indexing used in creating two degenerate plates for body type components. 40
3.11 Computation of degenerate plate nodes from camber line nodes via vector projection. . . . 41
3.12 Traversal of surface nodes to obtain maximum thickness for both fuselage and airfoil cross-sections. . . . 44
3.13 Calculation of maximum thickness location along chord for an airfoil section. 45
3.14 Sweep angle at wing quarter chord. . . . 47
3.15 Airfoil section showing vectors used in computation of cross-section area normal direction. . . . 47
3.16 VSP external storage component showing optional pylon and two fin surfaces. 53
3.17 Five surfaces which comprise the VSP engine component. . . . 54

3.18 Matlab structure format for degenerate geometry. 60
3.19 Format of additional Matlab structs describing propeller and point mass properties. 61

4.1 Geometry plots of Cessna wing from both VSP and AVL. 65
4.2 Trefftz Plane plot of Cessna wing force coefficients from AVL. 65
4.3 Modified Cessna wing used in lifting line theory analysis. 66
4.4 Aerodynamic properties of a modified Cessna wing computed with lifting line theory. 67
4.5 Comparison AVL and lifting line theory drag polars. 68
4.6 Comparison AVL and lifting line theory lift curves. 68
4.7 Geometry plots of Cessna wing and plate representation from VSP and plate in ELAPS. 70
4.8 Wing deflection computed by ELAPS for end-loaded wing 71
4.9 Plot of wing geometry in VSP and the beam representation composed of leading edge nodes. 71
4.10 Deflection on end-loaded wing treated as simple beam. 73

Chapter 1

Introduction

Aircraft design is comprised of three main stages: conceptual, preliminary, and detailed [1]. The conceptual phase is characterized by the exploration of large design spaces, which generally translates to large geometric variation among candidate designs [2]. Traditionally, concept geometry has borrowed heavily from past aircraft and relied on historical regressions to establish a sensible baseline configuration. The reasoning lies in the inherent iterative nature of design, the incredible resources consumed by CAD-style geometry modeling, and the fact that analysis can be both computationally and monetarily expensive.

Iterative solutions start with a first "guess" and generally converge better if this first guess is "good", which in the present context may be taken to mean close to the final solution. This methodology inevitably favors traditional designs, which are known to work, and hence serve as good starting points in the iterative process. The perhaps unintended consequence is that tradition heavily influences the eventual concept. Moreover, the approach is flawed both pedagogically and from the standpoint of commercial competition as it:

1. Disincentivizes novel ideas since they are viewed as more likely to fail and will waste resources on analysis, and
2. Necessarily does not explore the full design space; it is not a true requirements-driven process.

Relying on past designs to guide future development can be a powerful pedagogical tool if the focus is on the relationship between requirements—explicit or derived—and the eventual product. However, technological advances and seemingly minor differences in the role a vehicle is to fill may take the concept in an entirely different direction if it is allowed to do so.

The same argument may be made in a commercial context, though with the added complication of influences external to the design group such as budget, company management, and the regulatory and competitive environment. Ideally, requirements alone should drive a design, but they often don't. Part of the problem stems from inability to produce the multiple geometries needed without a large investment of man-hours. Neglecting external influences, the inability to inexpensively analyze those potential designs comprises the remainder.

To efficiently produce multiple geometries requires recognition of what differentiates them beyond physical dimensions. Differences from a design standpoint represent variations in metrics that affect performance—parameters such as wing aspect ratio, sweep, dihedral, or fuselage fineness ratio. Small changes in these quantities may represent non-trivial modification of geometric dimensions and affect relationships among components. Fortunately, Vehicle Sketch Pad (VSP), which was designed specifically with rapid aircraft conceptual design in mind, enables quick creation and modification of design concepts using high-level parameters, automatically adjusting component geometry accordingly.

The other challenge is timely analysis of each geometry with an eye toward overall design feasibility and a fidelity appropriate to the conceptual design phase. Inexpensive, yet accurate analysis tools exist, many of them open source, but thus far there has not been a general solution for interfacing these tools with a geometry modeler. Bridging the gap between a geometry modeler like VSP and inexpensive analysis tools will help mitigate the risk in attempting non-traditional designs by drastically reducing the time from an idea to an understanding of its feasibility. Translating a design concept into a general, distilled geometric representation suitable for multi-physics, multi-fidelity analysis and implementing the ability to generate and write out this geometry from VSP is the way to bridge this gap, and serves as motivation for the present work.

1.1 A Primer on VSP

VSP is a parametric geometry modeler created specifically to aide in aircraft conceptual design. It has been developed in various forms over the course of some twenty years at NASA by J.R. Gloudemans and others [2, 3, 4]. Recent release under NASA Open Source Agreement (NOSA) version 1.3 has facilitated access for users across the globe [3].

VSP's true power lies in its parametric nature. Pre-defined aircraft components are

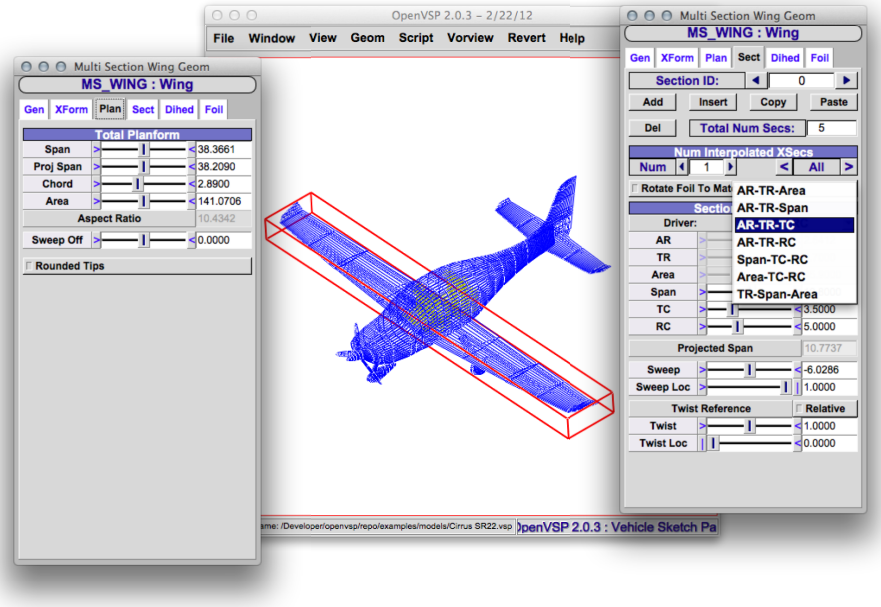

Figure 1.1: VSP design parameter groups allow quick and meaningful modification of component geometry.

easily modified using high-level design parameters in addition to geometric dimensions. With this parametric approach, VSP enables a designer to explore a wide array of aircraft configurations, creating multiple geometries in a fraction of the time required for just one using traditional CAD programs.

Figure 1.1 shows two tabs used to modify VSP's `MS_WING` component, providing a small example of how design parameters are quickly varied with simple sliders and numerical input boxes. Increasing wingspan will automatically increase area, aspect ratio, and any other affected quantities.

For each wing section, a designer has the choice of one of seven geometry driver groups. Each group is composed of three independent parameters, and adjusting any of them automatically adjusts dependent parameters. As an example, figure 1.1 shows the group Aspect Ratio–Taper Ratio–Tip Chord selected from the drop down menu. Adjusting any of these will automatically update the remainder—Area, Span, and Root Chord. Components can be reflected across planes of symmetry with the click of a button, cross-sections modified for sweep, twist, airfoil section with a few simple menus and sliders.

By design, the degree of control is reduced when compared to CAD programs. User-defined parts are not currently supported in VSP (though they are rumored to be on the

horizon), leaving only a handful of stock components to choose among. This may appear limiting, and depending on the context, can be. In the conceptual phase however, it is actually a benefit. Freeing the designer from the minutia of dimensioning every segment of every part and adjusting each of those dimensions to change, for example, sectional airfoils or twist angle, enables them to concentrate on the real task at hand.

To be sure, arbitrary geometry with a high degree of precision is necessary for machine drawings, but for exploration of high-level design concepts this degree of control is a hinderance rather than a help. Millimeter precision, rivet placement and fillets don't determine in a gross sense whether or not a wing produces enough lift, and use of this level of fidelity may well distract from the question of if it does.

Much in the same fashion, using high-fidelity analysis tools like CFD and FEA in the conceptual design phase can be a misuse of resources. Though these tools should not be explicitly dismissed, full-blown FEA and CFD are clearly not appropriate choices when trying to figure out if a wing aspect ratio or tip deflection is in the right range. Something like a vortex lattice or lifting line method would be an inexpensive and sufficiently accurate substitute for CFD, and instead of FEA, a quick equivalent beam technique will give wing tip deflections.

Current VSP outputs include high-quality CFD meshes and wetted volume and area reports, but no method of interfacing with other analysis programs. Similarly, commercial CAD software includes export for CFD meshers or FEA programs, but no efficient method of capturing geometric characteristics needed for other analysis techniques. The advent and development of high-fidelity analysis tools together with increased computational power has driven the toolset of choice to the most complex options, evidenced by the fact that most commercial modeling packages are set up to interface with meshing tools. Even if one invests the large amount of time necessary to create a number of design concepts using CAD, obtaining the information for analysis programs other than CFD or FEA can be both time-consuming and arduous.

A general method of capturing geometric information to enable inexpensive, multi-fidelity analysis techniques is needed. This should include the information necessary for analysis, but should not include the original full-fidelity model. It should create a reduced-order or *degenerate* geometric representation, serving as a conduit from design to analysis.

Incorporating the ability to write out this degenerate geometric information from VSP will enable a new design methodology that leans less on the past and concentrates more

on the requirements at hand. It will facilitate exploration of non-traditional concepts while simultaneously decreasing design cycle time. It will also break down the barriers to entry in the conceptual design and analysis process. Coupling a freely-available, open-source geometry modeler to classic analysis techniques—many of which are open-source or easily programmed—encourages participation by more individuals and organizations who may not have otherwise had the resources or opportunity to do so. It perhaps goes without saying that increasing participation in any field generally leads to innovation and technological progress, which benefits all involved.

A final point, though hardly an afterthought: optimization algorithms present an ideal method for balancing competing objectives and have the potential to play a powerful role in conceptual design. To truly realize their potential, they should also be a time-saving device, which requires inexpensive objective function calls. For purposes of conceptual design, inexpensive objective function calls means inexpensive analysis techniques and the ability to interface the optimization scheme to these techniques. Coupling a scriptable geometry modeler like VSP to analysis tools through degenerate geometry is an ideal path to achieving this goal. In fact, effort on a VSP plugin for *Model Center*, an optimization environment, is well underway at Phoenix Integration [5].

1.2 Justification for Degenerate Geometry

Invariably, analysis involves the use of equations or models which attempt to explain natural phenomena in a common language (mathematics). Moreover, real-world objects are described in terms of their macroscopic characteristics. These models are never true representations of underlying physics or material composition, but instead are chosen for their accuracy within some set of constraints.

We analyze aerodynamic phenomena for a wide range of temperatures and densities using a continuum assumption. It's not correct, but the effects of ignoring individual molecular contributions to pressure and other thermodynamic quantities are negligible under most circumstances and indeed, including them would be an unnecessary waste of resources.

For the same range of densities and temperatures, but at a much smaller length scale, ignoring the molecular nature of a fluid and a solid with which it is interacting would give worthless results. This implied variance in the value of fidelity is rarely addressed explicitly, but is encompassed by the question that the best engineers ask when confronted with a

choice of analysis techniques: is this the right tool for the job? The question is really asking: is this the appropriate level of fidelity for the answers I seek?

Multi-physics, multi-fidelity models have been in use since the inception of modern aerodynamic and structural analysis. Most debuted as the contemporary state of the art. With the advent of computing, higher-fidelity tools were developed, but the simplified techniques persisted because of their ease of use, time-efficiency, and relatively accurate results. The importance of these tools in conceptual design is paramount. If an engineer is trying to decide on wing length, airfoil section choices or some gross sizing parameter, full CFD and FEA analyses are not only unnecessary, but the wrong choice.

Preliminary lift and induced drag distributions are predicted accurately with Prandtl's lifting line theory [6], and recent additions to the technique have made it useful for wings with sweep and dihedral [7]. Similarly, reduced-fidelity analysis using AVL or other vortex lattice methods [8], equivalent beam theory [9], and equivalent plate representations [10, 11, 12, 13, 14] have all been shown to provide excellent accuracy at significantly reduced expense versus CFD or FEA.

These tools are useful, accurate, and inexpensive, and none of them use a full-fidelity representation of the geometry under analysis. There exists a need to develop a general definition of the types of geometry that these and other tools like them use.

1.3 Target Analysis Types

Four target analysis techniques—vortex lattice, equivalent plate, equivalent beam, and lifting line theory—helped guide the creation and definition of degenerate geometry. They were chosen to encompass structures and aerodynamics, since these are primary fields of concern in conceptual design. They can be divided into two categories of fidelity in geometric representation: "stick" and "plate" types. The former represents underlying geometry as one-dimensional, while the latter treats it as two-dimensional.

Figure 1.2 shows a visualization of plate and stick representations of a wing. Lifting line and equivalent beam theories treat the wing as a one-dimensional 'stick', like the one shown. Also, something very similar to the plate representation is used in vortex lattice analysis and equivalent plate theory.

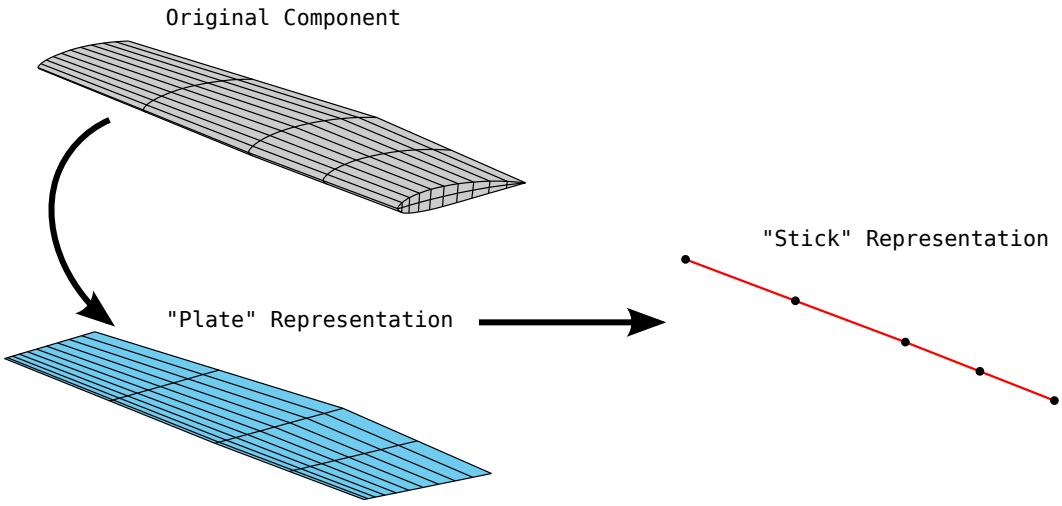

Figure 1.2: Visualization of wing plate and stick representations.

1.3.1 'Stick' Types

What a stick model of an aircraft component actually is depends on the component in question. For aerodynamic analysis of a wing, lifting line theory can be thought of as treating the wing as a stick. There isn't really a stick representation of a fuselage for aerodynamics models, but specifying cross-sectional area at stations along the length is a useful way to obtain a simple drag buildup.

From a structural standpoint, a wing or fuselage may be treated as a continuum beam and analyzed to obtain bending and torsional behaviors [9]. More detail on these two methods follows.

Lifting line theory

Prandtl's lifting line theory was developed during the period 1911-1918 and according to John D. Anderson, Jr., Professor Emeritus at University of Maryland:

> "[It is] The first practical theory for predicting the aerodynamic properties of a finite wing... The utility of Prandtl's theory is so great that it is still in use today for preliminary calculations of finite-wing characteristics." [6]

The beauty of Prandtl's theory is that it requires minimal geometric and aerodynamic characteristics and is extremely computationally efficient. For angles of attack within the linear lift curve slope region, all that is needed to solve for lift and induced drag at stations along the wing is the geometric angle of attack (α), the zero-lift angle of attack ($\alpha_{L=0}$),

chord length (c), wing span (b), and platform area (S). If sectional airfoil data is also included, this method can be extended to the entire range of angles of attack (including the stall region) and will provide C_L values that are accurate to within 20% [6]. Interestingly enough, though lifting line theory was developed from incompressible, irrotational potential flow theory, it has more recently been extended to treat the subject of supersonic flows [15].

Equivalent Beam Theory

VSP is capable of modeling of ribs, spars, and stringers, but at present these models target high-fidelity FEA, so a structural stick model of a wing can only really encompass cross-sectional area moments of inertia and center of gravity location. Specifying these, and other properties at nodes along the span is a basic finite-element formulation which forms the basis for equivalent beam methods (see [9] and [16] for two examples).

Even a simple one-dimensional beam model can give good preliminary estimates of static deformation behavior. Equivalent beam techniques have been used successfully in aeroelastic studies, and experimental validation has shown their accuracy for aspect ratios as low as 3 [17]. Even so, there is disagreement as to the meaning of an elastic axis and whether shear center, flexural center, or center of twist should be used in its definition [18]. Often one or more of these are considered the same, which further complicates matters. In either case, the definitions and calculation methods vary, and the result can be highly dependent on things like sectional skin thickness and depending on the problem, the idea of an elastic or flexural axis may not even be valid [19, 20]. For these reasons, elastic axis calculations are not considered in defining stick types for equivalent beam analysis.

1.3.2 'Plate' Types

Both equivalent plate structural analysis and vortex lattice methods treat objects as two-dimensional "plates". Equivalent plate structural analysis has been studied extensively, expanded upon, and used in tool design by Gary Giles at NASA Langley for more than twenty years. The fruit of his labors is a program called Equivalent Laminated Plate Solution (ELAPS). ELAPS will serve as a target analysis program and will be assumed to represent a standard for equivalent plate methods.

Though there are many ways to implement a vortex lattice method, Athena Vortex Lattice (AVL) serves as the *de facto* standard. Because it is freely available, popular, and

yields excellent results, AVL will serve as a target analysis program and a surrogate for all vortex lattice methods.

Both of these codes are primarily concerned with wings and wing-like objects, and were not designed with analysis of fuselages in mind [8]. For this reason, their role in guiding the creation of Degenerate Geometry is limited to wing-like components.

Vortex Lattice/AVL

Vortex lattice methods are many and varied in their formulation[6]. Most treat a three-dimensional wing as a flat "plate" of horseshoe vortices with appropriate boundary conditions imposed for camber. Theoretical development of vortex lattice methods is beyond the scope of the current discussion, but reference [21] is an excellent resource for the curious reader.

Suffice to say that a plate representation of a wing for the purposes of vortex lattice analysis should include nodes of a discretized wing planform and information about the camber line slope and location in relation to those nodes. Also, vortex lattice analysis is designed with thin lifting surfaces in mind and hence has no real treatment of fuselages or other similar components [21]. Though AVL makes provisions for inclusion of fuselages, the user manual recommends omitting them if at all possible [8].

Equivalent Plate/ELAPS

Equivalent plate structural analysis as envisioned by Gary Giles, requires only minimal information about the structure being analyzed. All geometric information is given in the form of polynomials in a global coordinate system. Equations for camber line location, sectional thickness, and skin thickness (one for each layer of a composite skin) are defined for each wing segment [10, 11]. Though later versions of ELAPS are capable of modeling fuselage structures, their analysis is based on ring and shell equations, not by treating them as equivalent plates [14].

Equivalent plate analysis has proven itself quite accurate when compared to finite element methods not only for static deflections, but also in predicting component natural frequencies. One study found a roughly 1% difference for the first two modes and less than 6.5% for up to 7 modes at a cost of about 1.5% of the time necessary to run FEM [10]. Stress distributions and displacements were almost indistinguishable and also computed in a fraction of the

time [10, 11]. Additionally, equivalent plate methods have been coupled with aerodynamic solvers to perform static aeroelastic analysis and validated against wind tunnel testing [22].

1.3.3 Additional Types

If plates and sticks are two- and one-dimensional representations of three-dimensional objects, then three- and zero-dimensional representations would complete the spectrum. A three-dimensional degenerate representation of a three-dimensional object is simply a discretized surface represented by nodes that exist on the actual body. Vortex lattice methods like AVL compute a plate representation internally and actually expect something like surface nodes for input, rather than a pre-computed plate. The same may very well be true of other analysis programs, so a degenerate surface should be included to ensure compatibility.

A zero-dimensional object is a geometric point, but full components are often treated as point masses at their respective centers of gravity for inertia or energy calculations. Moreover, drag buildups may rely on wetted area, and simple textbook lookups or hand calculations are often concerned with component properties like mass moment of inertia or volume. The purpose of degenerate geometry is to provide information to service the largest selection of reduced-fidelity analysis techniques, so a degenerate point type is also included.

Chapter 2

Degenerate Geometry Definitions

Aircraft components are broken up into two main categories for the purpose of defining degenerate geometric representations. *Surfaces* are objects such as wings, stabilizers, aerodynamic pylons, etc. They are essentially wing-like objects, lifting or non-lifting (symmetric). *Bodies* encompass everything else from fuselages to propellor spinners and landing gear. The inspiration for this classification system was taken directly from Athena Vortex Lattice (AVL), which categorizes components in precisely this manner [8]. Bodies may play a potentially large role in overall drag, but are not suitable for (basic) aeroelastic studies or to the computation of lift. This classification is also convenient because certain characteristics can be assumed about each geometry type. Surfaces for instance, will have one dimension much smaller than the other two, meaning that qualities like thickness are easily defined.

Figure 2.1 shows a Cessna 182 model with body components in blue and surface components in red. Notice that the wing and main gear struts are surface types since they are made from aerodynamic, wing-like shapes, whereas the landing gear wheels and pants are body types. Though not shown in this view, the nose gear strut is also a body type, since it was modeled as a right circular cylinder.

Four levels of degenerate geometry have been defined. They are, in order of decreasing fidelity: *surface*, *plate*, *stick*, and *point*, corresponding to three-, two-, one-, and zero-dimensional representations, respectively. Both surface and body types can be distilled into any one of these four, though their definitions are slightly different depending on the

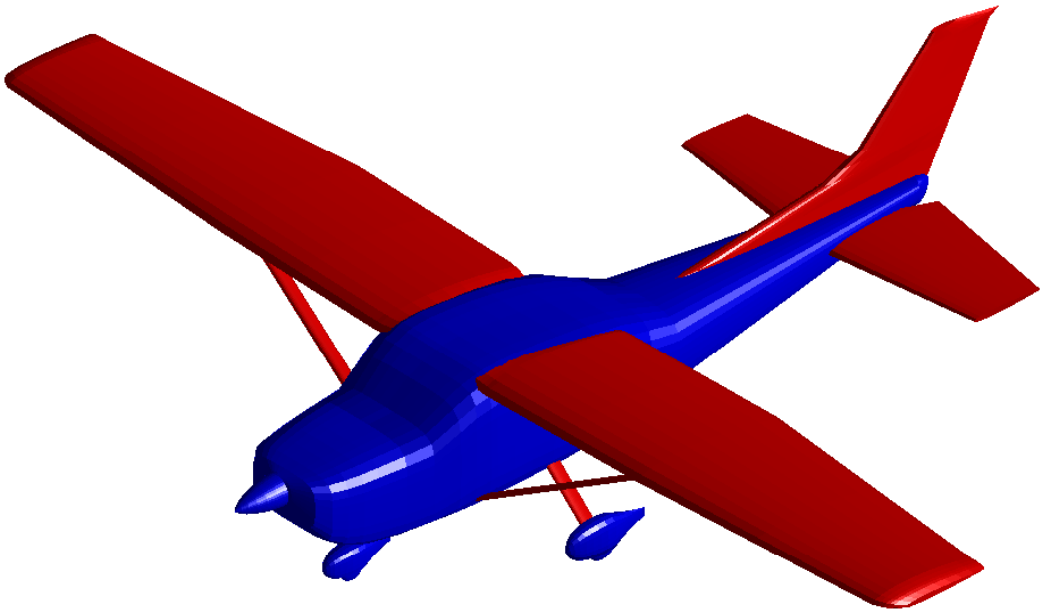

Figure 2.1: Cessna 182 model showing body (blue) and surface (red) type components.

category of the original geometry. The following sections define each of these degenerate geometries, for both surface and body type components. In each of these analyses, it is assumed that a coordinate system is adopted whereby positive x is aft down the fuselage, positive y is out the right wing, and positive z is up.

2.1 Degenerate Surface

An object's true geometry is three-dimensional, continuous on a macroscopic scale, and may be closely approximated by, but is never entirely amenable to mathematical description. In fact, describing an object mathematically is the first stage in reducing it's fidelity from true geometry to some tractable characterization. Doing so generally requires selecting points that comprise the object and connecting them in a piecewise fashion with curves of a desired order, the simplest of which are straight lines. These control points then, can be thought of as a reduced-fidelity representation of true, underlying geometry. In fact a degenerate surface has been defined such that it contains a collection of these control points. Connected together, they form a surface which approximates the original object.

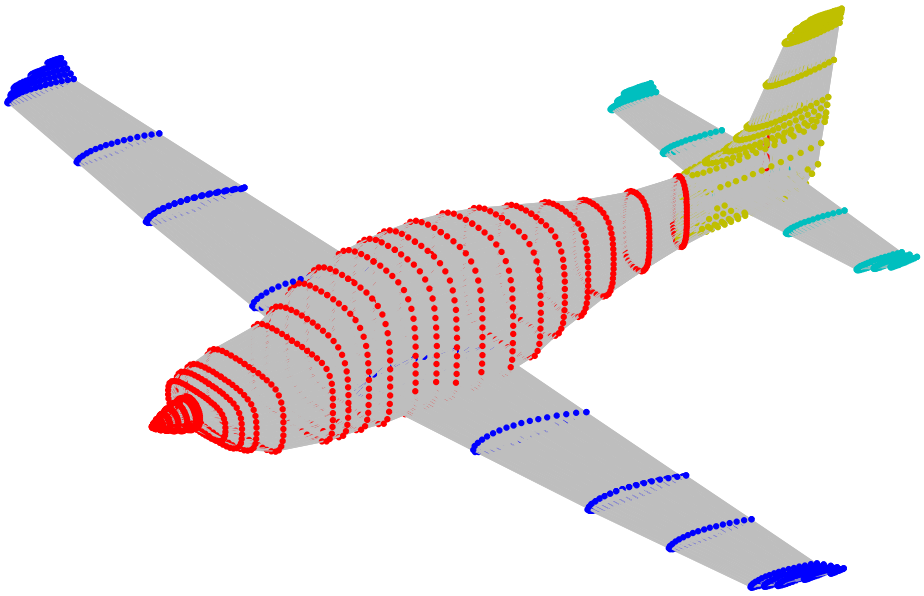

Figure 2.2: Cirrus SR22 model, showing degenerate surface representation.

Figure 2.2 shows this discretization on a Cirrus SR22, with each component shown in a different color. Note that the points have been grouped into cross-sections along each component, and though not obvious in the figure, each cross-section contains the same number of points within any component. The level of fidelity (or accuracy of representation) of any cross-section is limited only by how many control points are used, with the model approaching the actual object (in a continuum or macroscopic sense) as the number of cross-sections and control points becomes infinite.

A natural question is what form this collection of control points takes. The answer is a simple vector of coordinates ordered by cross-section. If a component has p cross-sections and q points per cross-section, then there are $m = p \times q$ control points and the vector of these control points is

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]^\intercal \tag{2.1}$$

where each $\mathbf{x}_i$ is a cartesian coordinate triplet (x_i, y_i, z_i). The number of cross-sections and points per cross-section are included as a means of effectively using this information. Degenerate surface also provides outward surface normal vectors in the form

$$\mathbf{X}_n = [\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2, \cdots, \hat{\mathbf{n}}_r]^\intercal \tag{2.2}$$

where each $\hat{\mathbf{n}}_i$ is a unit vector $(n_{x_i}, n_{y_i}, n_{z_i})$ describing the outward-facing surface normal

direction. Since a point has no single outward direction, normal vectors are defined using surrounding points. If control points within a cross-section are indexed by i and cross-sections by j, then a normal vector $\mathbf{n}_{ij} = \mathbf{t}_1 \times \mathbf{t}_2$, where $\mathbf{t}_1 = \mathbf{x}_{i,j+1} - \mathbf{x}_{ij}$ and $\mathbf{t}_2 = \mathbf{x}_{i+1,j} - \mathbf{x}_{ij}$ and $\hat{\mathbf{n}}_{ij} = \mathbf{n}_{ij}/\|\mathbf{n}_{ij}\|$, as shown in figure 2.3. Though the normal vector is shown at the node where its defining vectors intersect, it is in fact a best estimate of the red surface's normal direction. It is shown at the node to emphasize how it is determined and because the red panel is not necessarily planar. Note also that for each cross-section, there will be one less normal vector than control point, and the last cross-section will have no normal vectors. The length of $\mathbf{X}_n$ is $r = (p-1)(q-1)$. Note also that though degenerate geometries are categorized as either surface or body types, both types can be described with the same degenerate surface definition.

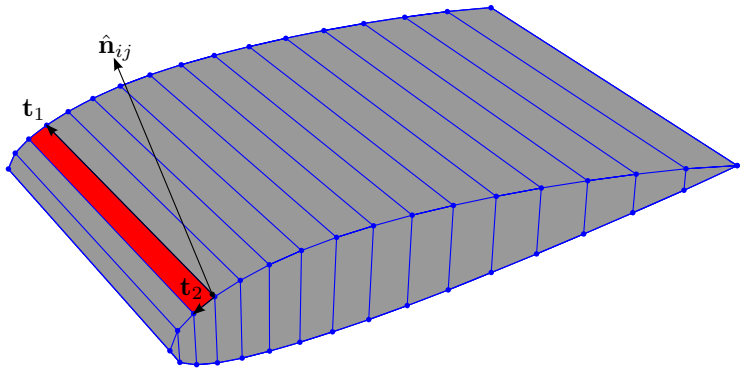

Figure 2.3: An example degenerate surface normal vector on a wing section.

VSP stores a component internally as a collection of nodes, with each one described by both a cartesian coordinate triplet and by a parametric coordinate (u, w), where u describes a cross-section's location "along" a component, and w describes each point's location "around" a cross-section. Parametric coordinates such as these can be extremely useful in mapping between unique discretizations of the same component (for two different analysis techniques, for example). These coordinates have been included in the VSP output (discussed in detail in Chapter 3) in the form of two vectors: $\mathbf{u}$ and $\mathbf{w}$.

2.2 Degenerate Plate

The next step in fidelity reduction is to represent a three-dimensional object as two-dimensional. Since surface type components have one dimension much smaller than the other two, it is quite natural to collapse them down on that dimension. In fact, this two-dimensional

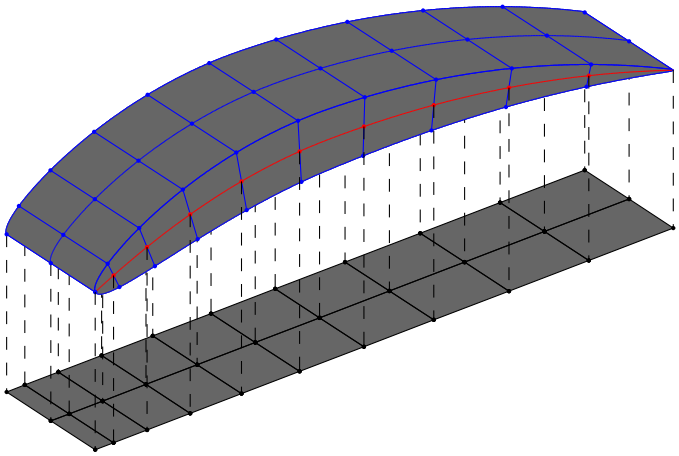

Figure 2.4: Mapping between discretized surface points and degenerate plate points for a wing section.

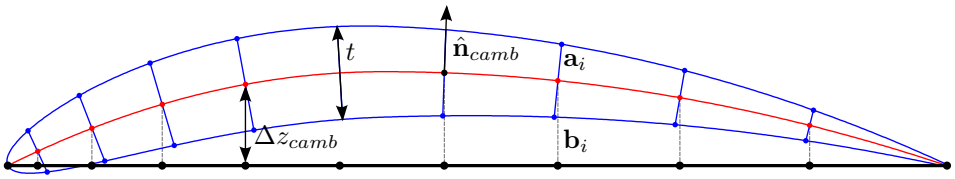

Figure 2.5: Degenerate Plate attribute definition using an airfoil section.

plate representation was inspired by both equivalent plate structural analysis and vortex lattice aerodynamic analysis, which are primarily concerned with wings and wing-like components [8, 12]. For this reason, degenerate plate's definition relies on airfoil nomenclature and general geometry. Unlike degenerate surfaces, degenerate plates need separate definitions for surface and body type components. For simplicity, a surface definition is provided first.

To create a degenerate plate, an object is first discretized into a series of cross-sections, which are each represented by a number of coordinate points, essentially a degenerate surface without normal vectors. These points are then collapsed down to a planar representation as shown in figure 2.4.

Figure 2.5 shows details of how these points are mapped from a single cross-section to a plate. First, the midpoints between corresponding upper and lower nodes are calculated via

$$\mathbf{X}_{camb} = \frac{1}{2}\left(\mathbf{X}_{top} + \mathbf{X}_{bottom}\right) \tag{2.3}$$

where the $\mathbf{X}$s are vectors of coordinate points (x, y, z) defining the upper and lower surfaces.

These midpoints form the camber line shown in red. The computed camber points are then projected onto the sectional chord line—formed by connecting the leading and trailing edges—using vector projection. If $\mathbf{x}_{te}$ and $\mathbf{x}_{le}$ are the trailing and leading edge coordinates, respectively, then a normalized vector along the chord pointing from the trailing edge to the leading edge is given by

$$\hat{\mathbf{c}} = \frac{\mathbf{x}_{le} - \mathbf{x}_{te}}{\|\mathbf{x}_{le} - \mathbf{x}_{te}\|} \tag{2.4}$$

If a vector from the trailing edge to a camber point $\mathbf{a}$ is given by $\mathbf{a}'$, then the degenerate plate point $\mathbf{b}$ corresponding to $\mathbf{a}$ is given by

$$\mathbf{b} = \mathbf{x}_{te} + \hat{\mathbf{c}} \times (\mathbf{a}' \cdot \hat{\mathbf{c}}) \tag{2.5}$$

At each node $\mathbf{b}$, degenerate plate also reports the distance to $\mathbf{a}$ as a magnitude,

$$\Delta z_{camb} = \|\mathbf{a} - \mathbf{b}\| \tag{2.6}$$

thickness of the original section t, which is simply the distance between corresponding top and bottom points of the original cross-section,

$$t = \|\mathbf{X}_{top} - \mathbf{X}_{bottom}\| \tag{2.7}$$

and the camber line normal vector

$$\hat{\mathbf{n}}_{camb} = \frac{\mathbf{X}_{top} - \mathbf{X}_{bottom}}{\|\mathbf{X}_{top} - \mathbf{X}_{bottom}\|} \tag{2.8}$$

Additionally, for each cross-section a plate normal vector $\hat{n}_{plate}$ is given, which provides the original orientation of the cross-section and defines the direction from the plate points b to the camber line points a. This is computed using any corresponding points a and b via

$$\hat{\mathbf{n}}_{plate} = \frac{\mathbf{a} - \mathbf{b}}{\|\mathbf{a} - \mathbf{b}\|} \tag{2.9}$$

Once again, the VSP parametric coordinates, though not part of the definition of degenerate plate, are reported in the output at each station and collected in vectors: $\mathbf{u}$, $\mathbf{w}_{top}$, and $\mathbf{w}_{bottom}$. Note that in this instance a top and bottom w are reported since each plate point corresponds to two nodes on the original geometry.

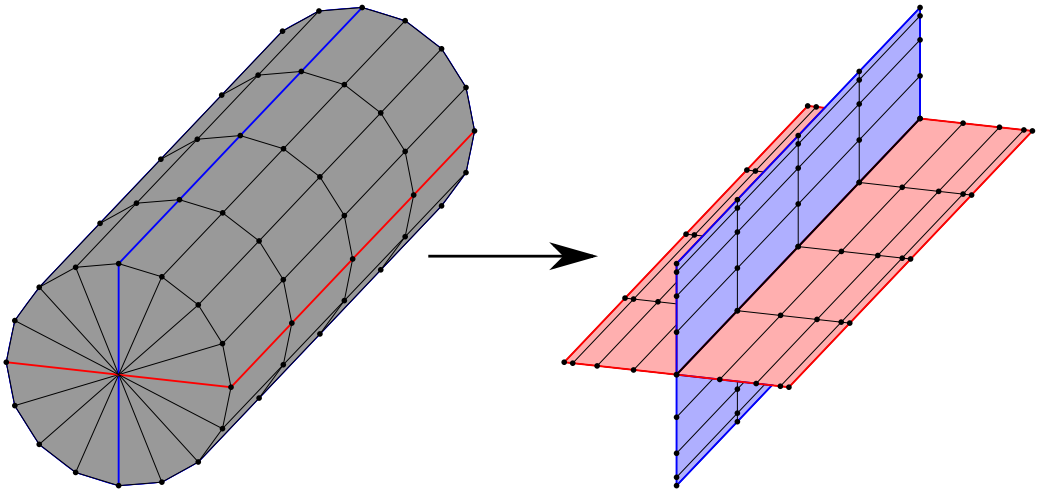

Figure 2.6: Transformation from body component to degenerate plate using a right circular cylinder.

Having sufficiently defined a degenerate plate representation of wing-like components, a question arises concerning what this definition is for something like a fuselage. Defining a general plate orientation to represent thick bodies is decidedly difficult. If an object is axisymmetric, then any section which bisects it into two symmetric pieces would make an appropriate plate. However, the vast majority of aircraft body parts are not axisymmetric and so a different approach is necessary. Since it is virtually impossible to state, in a general sense the "least dominant" dimension for arbitrary body geometry, it was decided that collapsing a part along two separate dimensions was the best alternative. In this manner, a truer representation of the original geometric characteristics is preserved, while still reducing fidelity. This means that degenerate plates composed of body geometry actually contain two plate objects. This is easiest to visualize using a right circular cylinder as shown in figure 2.6. The two plates are defined such that they equally divide the number of discretized nodes in each cross-section. This means that though the plates will nominally be orthogonal, if the nodes are unequally distributed (i.e more on the left than right half, etc.) the plates will assume non-orthogonal orientations. Additionally, since this is done on a per cross-section basis the plates' locations can vary along a body, meaning that its degenerate plate representation is not necessarily planar in a cartesian coordinate system. Aside from creating two degenerate plates from each component, the definitions of thickness, distance to camber line, et cetera are all analogous to surface type degenerate plates. Figure 2.7 shows a Cessna 182 degenerate plate model 3 view, with each component shown in a different color. All

components are surface types and hence collapse down to single plates, with the exception of the fuselage, which is represented by two plates.

It might be of interest to note that the degenerate geometry definitions say nothing about how various components are connected. In fact, figure 2.7 shows that they may not intersect at all. This is beneficial as it provides an engineer, who will have insight into the geometry and analysis techniques being used, with freedom to choose or ignore connections between components and specify their material or aerodynamic properties as appropriate.

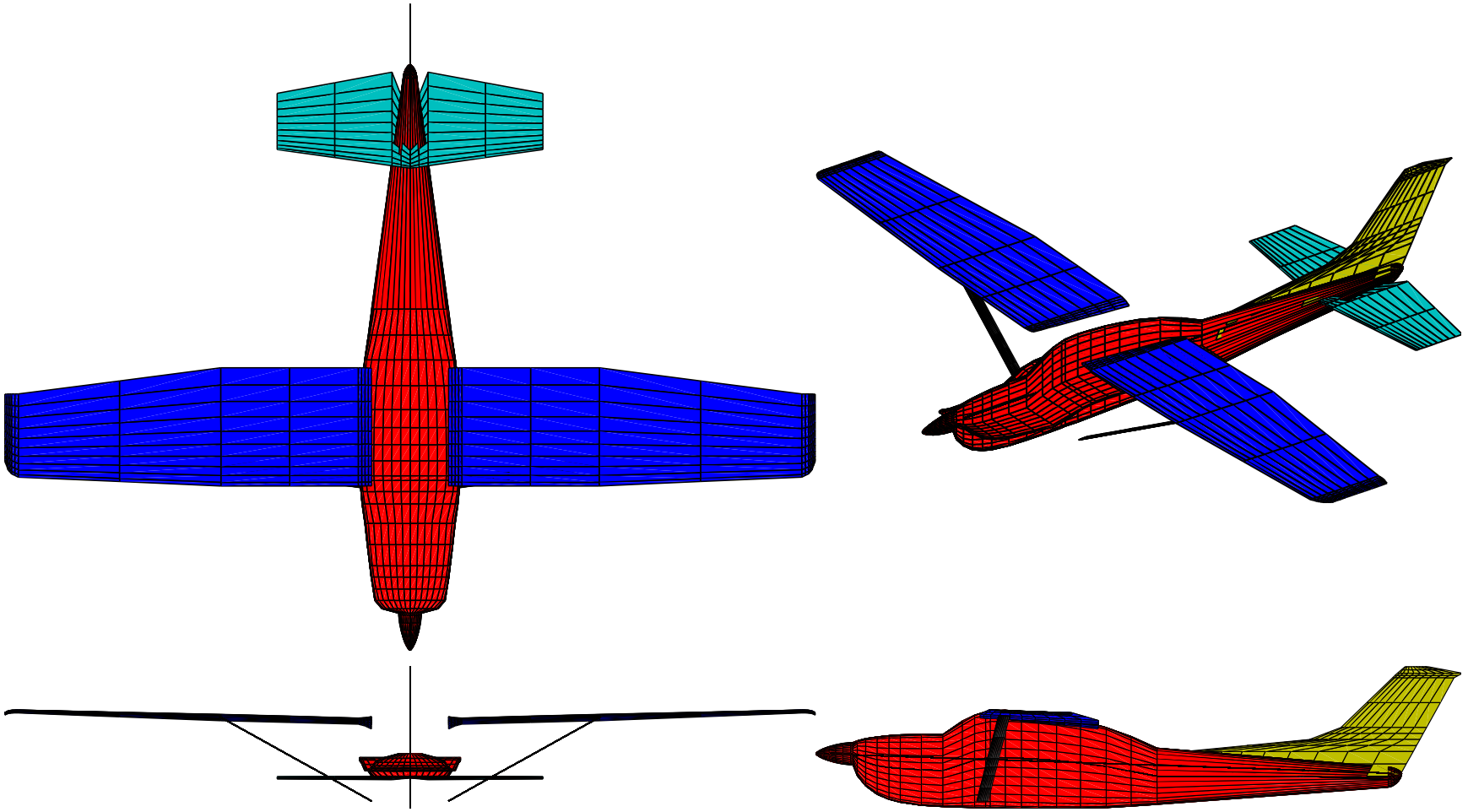

Figure 2.7: Cessna 182 degenerate plate three view.

2.3 Degenerate Stick

Degenerate stick reduces fidelity further from degenerate plate, creating a one-dimensional representation, where each point on the degenerate stick corresponds to a cross-sectional slice of the actual geometry. Like degenerate plate, degenerate stick relies on airfoil nomenclature, but has separate definitions for surfaces and bodies. Once again, the surface definition is presented first, followed by extension to the body definition.

If a degenerate plate is a degenerate surface collapsed to a plane, then a degenerate stick is a degenerate plate collapsed to a line. To create a degenerate stick, an object is first discretized into cross-sections, with each one corresponding to a degenerate stick node. Figure 2.8 shows these nodes connected together (red line) along with the original wing. Though the nodes and connecting line shown are an easy way to visualize a degenerate stick, the points that define it are actually the leading and trailing edge points from the original component (shown in blue). As with other degenerate geometries, node locations are collected in vectors

$$\mathbf{X}_{le} = [\mathbf{x}_{le,1}, \mathbf{x}_{le,2}, \cdots, \mathbf{x}_{le,p}]^\intercal \tag{2.10}$$

$$\mathbf{X}_{te} = [\mathbf{x}_{te,1}, \mathbf{x}_{te,2}, \cdots, \mathbf{x}_{te,p}]^\intercal \tag{2.11}$$

where each $\mathbf{x}_i$ is a cartesian coordinate triplet corresponding to one of the p discretized cross-sections. Degenerate sticks also report maximum thickness non-dimensionalized by chord length, the maximum thickness location as a fraction of the chord length, chord length, cross-sectional area, an area normal vector, and top and bottom perimeters, as shown in figure 2.8. The maximum thickness is defined as the maximum of the distances between adjacent top and bottom nodes from the degenerate surface representation

$$t_{max} = \max\left\{\|\mathbf{X}_{top} - \mathbf{X}_{bottom}\|\right\} \tag{2.12}$$

or alternately, as the maximum thickness reported for the degenerate plate nodes of the same cross-section. The maximum thickness location is the location of the camberline point at max thickness projected onto the chord line and is reported as a fraction of chord length. This is the same as finding the chord location of the degenerate plate node with the maximum thickness. The top and bottom perimeters are found by assuming that each set of adjacent nodes is connected by a straight line and then adding up the length of each of

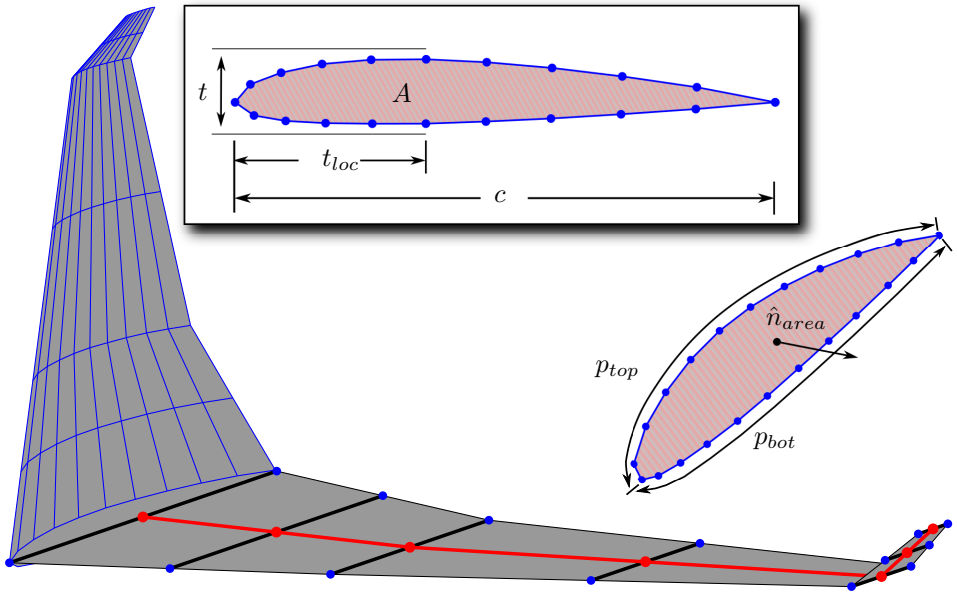

Figure 2.8: Degenerate stick model of Boeing 747 wing.

those lines. The cross-sectional area is found via equation (2.17), the details of which are given below. Since a cross-section is planar, an area normal vector can be found by taking the cross product of two vectors defined by any points within the cross-section. Not shown in figure 2.8, but included in degenerate stick is the quarter chord sweep angle, defined as positive for rearward sweep. Note that this angle is defined in the x-y plane, so that a component whose quarter chord locations align with the y-axis has a sweep angle of zero.

Of interest in equivalent beam structural analysis are sectional moments of inertia, specifically those resisting lift and drag and a torsional moment of inertia. These can be provided in a general manner for both solid and thin-walled "shell" sections without knowing the units or wall thickness properties. Since creating a degenerate stick requires discretization of a component into cross-sections composed of points, each cross-section can be treated as a polygon defined by q points. Research by Steger [23] resulted in formulae for moments of arbitrary order for polygons. Those for area moments about the origin assuming a cross-section confined to the x-z plane are given by

$$J_{xx} = \frac{1}{12}\sum_{i=1}^{q}(x_{i-1}z_i - x_i z_{i-1})(z_{i-1}^2 + z_{i-1}z_i + z_i^2) \tag{2.13}$$

$$J_{zz} = \frac{1}{12}\sum_{i=1}^{q}(x_{i-1}z_i - x_i z_{i-1})(x_{i-1}^2 + x_{i-1}x_i + x_i^2) \tag{2.14}$$

The parallel axis theorem is employed in order to get moments of inertia about cross-section

centroid:

$$J^*_{xx} = J_{xx} - A\bar{z}^2 \tag{2.15}$$

$$J^*_{zz} = J_{zz} - A\bar{x}^2 \tag{2.16}$$

where $(\bar{x}, \bar{z})$ is the cross-sectional centroid location and A the area. These can be found via

$$A = \frac{1}{2}\sum_{i=1}^{q} x_{i-1}z_i - x_i z_{i-1} \tag{2.17}$$

$$\bar{x} = \frac{1}{6a}\sum_{i=1}^{q}(x_{i-1}z_i - x_i z_{i-1})(x_{i-1} + x_i) \tag{2.18}$$

$$\bar{z} = \frac{1}{6a}\sum_{i=1}^{q}(x_{i-1}z_i - x_i z_{i-1})(z_{i-1} + z_i) \tag{2.19}$$

For all components, it is assumed that lift acts in the z-direction, and drag in the x-direction. Equations (2.15) and (2.16) then correspond to resistance to lift and drag, respectively. The resistance to torsion J^*_{yy} is about the y-axis and is simply the sum of J^*_{xx} and J^*_{zz}. All solid cross-section inertias are given in units to the fourth power.

Degenerate stick also gives cross-sectional center of mass (gravity), which for a solid cross-section of constant density coincides with the centroid. The center of gravity coordinates then are given by equations (2.18) and (2.19) or

$$X_{cg} = (\bar{x}, \bar{y}, \bar{z}) \tag{2.20}$$

where $\bar{y}$ is simply the y-location of the cross-section.

If a cross-section composed of q points is instead treated as a shell with small thickness, then each of the $n = q - 1$ internodal line segments can be treated as a rectangle. Figure 2.9 shows an airfoil section broken up into rectangles, with thickness exaggerated to show detail. For each rectangle, if the length is b and the thickness t, then the moments of inertia about its centroid c can be found via

$$J^*_{xx} = \frac{bt^3}{12} \tag{2.21}$$

$$J^*_{zz} = \frac{b^3 t}{12} \tag{2.22}$$

$$J^*_{xz} = 0 \tag{2.23}$$

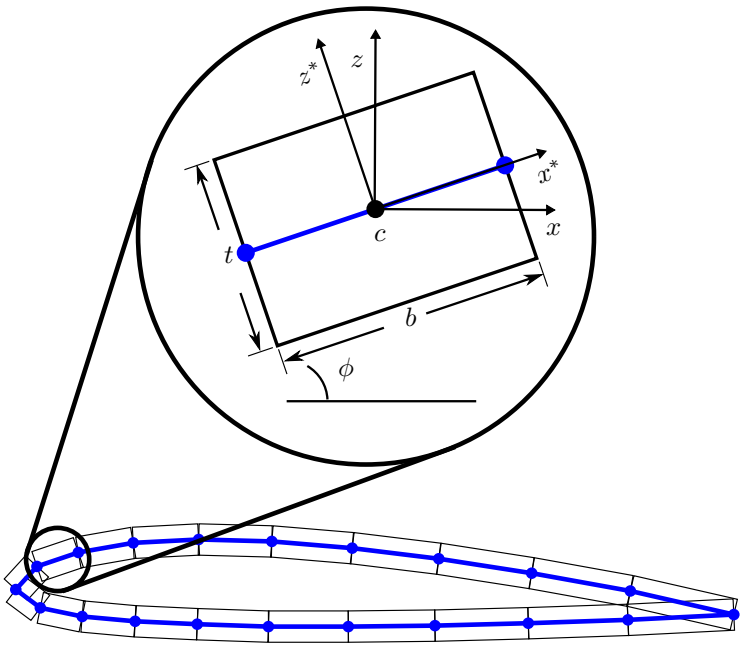

Figure 2.9: Shell representation of an airfoil section using rectangles to model thickness.

Rotating these inertias by ϕ so that they align with the global coordinate system and applying the parallel axis theorem, gives the contribution to cross-sectional inertia of each discretized segment

$$J_{xx} = \frac{J^*_{xx} + J^*_{zz}}{2} + \frac{J^*_{xx} - J^*_{zz}}{2}\cos(2\phi) - J^*_{xz}\sin(2\phi) + btd_z^2 \tag{2.24}$$

$$J_{zz} = \frac{J^*_{xx} + J^*_{zz}}{2} - \frac{J^*_{xx} - J^*_{zz}}{2}\cos(2\phi) + J^*_{xz}\sin(2\phi) + btd_x^2 \tag{2.25}$$

where d_x and d_z are the distances from c to the cross-section centroid along the x-axis and z-axis, respectively. Substituting equations (2.21), (2.22), and (2.23) into (2.24) and (2.25) yields equations of the form

$$J_{xx} = a_1 t^3 + a_2 t \tag{2.26}$$

$$J_{zz} = a_3 t^3 + a_4 t \tag{2.27}$$

where the coefficients a_1, a_2, a_3, and a_4 are given by

$$a_1 = \frac{b}{24}\left[1 + \cos\left(2\phi\right)\right] \qquad a_3 = \frac{b}{24}\left[1 - \cos\left(2\phi\right)\right]$$

$$a_2 = \frac{b^3}{24}\left[1 - \cos\left(2\phi\right)\right] + bd_z^2 \qquad a_4 = \frac{b^3}{24}\left[1 + \cos\left(2\phi\right)\right] + bd_x^2$$

Notice that for small thickness—a critical assumption—the t terms a_2 and a_4, which include the parallel axis theorem, dominate the t^3 terms. Though the choice was made not to, one would be justified in leaving out the t^3 terms if desired.

The total cross-sectional inertia is simply the sum of the contributions of each segment and is given in the form

$$J_{xx,tot} = A_1 t^3 + A_2 t \tag{2.28}$$

$$J_{zz,tot} = A_3 t^3 + A_4 t \tag{2.29}$$

where the coefficients are now the sum of the respective coefficients from each segment

$$A_1 = \sum_{i=1}^{n} a_{1,i} \qquad A_3 = \sum_{i=1}^{n} a_{3,i}$$
$$A_2 = \sum_{i=1}^{n} a_{2,i} \qquad A_4 = \sum_{i=1}^{n} a_{4,i}$$

Recall that since these are aligned with the global coordinate axes and lift is assumed to act in the z-direction, $J_{xx,tot}$ and $J_{zz,tot}$ are cross-sectional resistances to bending due to lift and drag, respectively. The resistance to torsion is $J_{yy,tot}$ and is the sum of $J_{xx,tot}$ and $J_{zz,tot}$ or

$$J_{yy,tot} = (A_1 + A_3)\, t^3 + (A_2 + A_4)\, t \tag{2.30}$$

Degenerate stick reports these four coefficients A_1, A_2, A_3, and A_4 for each cross-section so that shell inertia is defined as a function of thickness and may be recovered by simply inserting t into equations (2.28), (2.29), and (2.30). Note that approximating the surface as a series of rectangles relies on a thin wall assumption so that the error due to overlapping segments is small. Most shell structures for aerospace applications fit this criterion.

The center of gravity for an object of composite shapes, again assumed to be confined to the x-z plane, may be found via

$$\bar{x} = \frac{1}{M} \sum_{i=1}^{n} x_i m_i \tag{2.31}$$

$$\bar{z} = \frac{1}{M} \sum_{i=1}^{n} z_i m_i \tag{2.32}$$

where (x_i, z_i) is the location of each rectangle's center of mass,

$$M = \sum_{i=1}^{n} m_i \tag{2.33}$$

and $\bar{y}$ is the simply the y-location of the cross-section. The mass of each rectangle is given by the product of its area and density or

$$m_i = \rho_i A_i \tag{2.34}$$

where the area, as shown in figure 2.9 is given by

$$A_i = b_i t \tag{2.35}$$

Substituting equations (2.33), (2.34), and (2.35) into equations (2.31) and (2.32) results in

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i \rho_i b_i t}{\sum_{i=1}^{n} \rho_i b_i t} \tag{2.36}$$

$$\bar{z} = \frac{\sum_{i=1}^{n} z_i \rho_i b_i t}{\sum_{i=1}^{n} \rho_i b_i t} \tag{2.37}$$

The thickness is assumed to be the same for all rectangles. If the density is also assumed not to vary between rectangles, then all ρ_i are equal to one value, ρ which can be factored out of the summations and divided out along with t. The final expressions for the center of gravity location then become

$$\bar{x} = \frac{1}{B} \sum_{i=1}^{n} x_i b_i \tag{2.38}$$

$$\bar{z} = \frac{1}{B} \sum_{i=1}^{n} z_i b_i \tag{2.39}$$

where

$$B = \sum_{i=1}^{n} b_i \tag{2.40}$$

The center of gravity then can be specified without regard to the thickness of the shell. Obviously the same caveat applies that in order to represent a shell as a series of rectangles, the thickness must be small.

Just like the other Degenerate Geometries, when implemented in VSP (see Chapter 3) the internal parametric coordinates are appended to the output. For degenerate stick, only

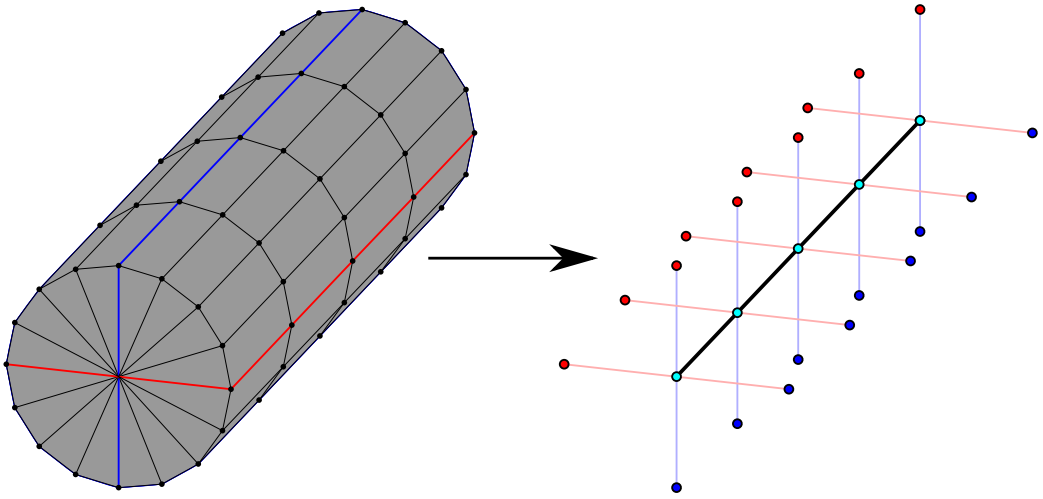

Figure 2.10: Transformation from body component to degenerate stick using a right circular cylinder.

the coordinate u is reported, since each node representing a cross-section corresponds to some u coordinate along a component.

As mentioned previously, the definition of degenerate stick needs some extension to deal with body type components. In a similar manner to degenerate plate, this is accomplished by collapsing underlying geometry down along two separate (nominally orthogonal) directions and reporting two sets of information for each cross-section. This is again most easily shown using a right circular cylinder. Figure 2.10 shows how a body component is transformed into a degenerate stick. The "leading edge" points are shown in blue, while the "trailing edge" points are shown in red. Note that the degenerate stick shown in black with black and teal nodes actually has two sets of data reported at each node, or there are two degenerate sticks overlying one another.

2.4 Degenerate Point

The final step in fidelity reduction is to treat a component as zero-dimensional, or a geometric point. Degenerate point does this for shell or solid components, reporting the center of gravity location for each of these two cases. Additional information that degenerate point contains is the component volume, area, wetted volume, wetted area, and mass moments of inertia for a solid and shell.

The wetted properties make degenerate point unique among the degenerate representations, in that it is the only one that relies on other components for information. External

component geometry is needed to find what portions of the area and volume of the component of interest are intersected.

Six inertia values are given for both a solid and shell representation of each component: I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz}, with the products of inertia assumed symmetric (i.e. $I_{xy} = I_{yx}$). For solid components, these are given per density and for shell components per surface density. To get moments of inertia, simply multiply by density ρ, or by density and thickness, ρ and t depending on which set of inertias, solid or shell, are desired.

There are multiple discretization methods for obtaining three-dimensional properties like those found in degenerate point. A recommended approach is that outlined by Dobrovolskis, in which a triangular surface mesh is created and used to define tetrahedra, which together comprise the volume of the original object [24]. A similar method already existed in VSP and was used in the computation of degenerate geometry properties (see section 3.1.4).

Chapter 3

Implementation in VSP

Vehicle Sketch Pad has been developed over the course of many years and has taken a few different forms [3, 4]. Though it is written in C++, work was begun before much of what is colloquially called the Standard Template Library (STL) existed. For instance, dynamically-sized container classes didn't exist and needed to be created [25]. Additionally, though VSP doesn't adhere very strictly to accepted object-oriented design patterns (Model-View-Controller, Factory, etc.), and despite its evolutionary growth, the class inheritance structure is well organized [26]. This is important as it allowed degenerate geometry computation and write-out capabilities to be implemented in relatively short order and with the ability to easily add support for future components, which is one of the primary benefits of object-oriented languages.

The purpose of this chapter is to give an overview of VSP's code structure and organization, and how degenerate geometry fits into that structure. It will also explain some of the lower level details of how various degenerate types are computed, what VSP components can be distilled into these types, and how future, user-defined components can interface with the degenerate geometry routines. Finally, the output file types for degenerate geometry will be discussed.

3.1 Class Structure & Code definitions

Vehicle Sketch Pad is centered around the vehicle, both in principle and in practice. Whether in batch mode or using the UI, the top level class is `Vehicle`, which holds references to all component geometries, and which are active and/or top level (no parents). It also holds a

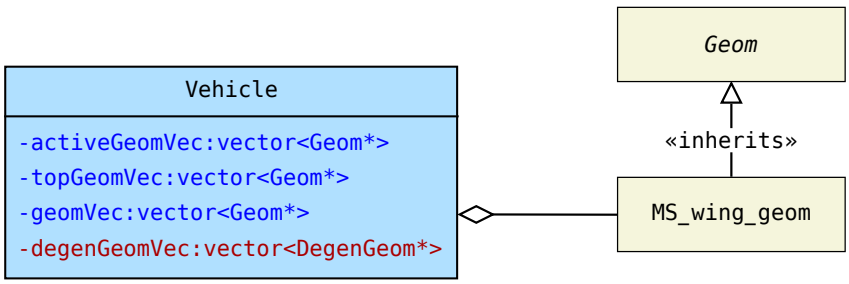

Figure 3.1: An overview of the relationship between the Vehicle class and component geometries.

fair amount of information about GUI components like a reference to the `ScreenMgr` class (which manages GUI screens), which components are highlighted, etc. Some of this may change as a standalone geometry kernel is created, but `Vehicle` will likely still act as a top-level class that tracks component geometry. Figure 3.1 shows the relationship between `Vehicle` and the various component geometries that make up the vehicle. References to all components are contained in `geomVec`, with subsets contained in `activeGeomVec` and `topGeomVec`. Every component is actually a subclass of the abstract `Geom` class, each with fields, methods and attributes specific to itself.

`Vehicle` also includes methods for file read/write, mass properties, and the `CompGeom` routines, among others. This arrangement is reasonable since all component geometry is visible at the `Vehicle` level. The motivation for implementing the methods responsible for degenerate geometry creation and write-out was much the same: since all component geometry is accessible from `Vehicle`, it is easy to loop though each component and create its corresponding degenerate geometry from `Vehicle` as well.

Each set of degenerate geometries that is created is contained in an instance of the class `DegenGeom`. As shown in figure 3.2, `DegenGeom` contains four C-structs corresponding to the four degenerate geometry types and an enumeration which tags a component as either a body or surface type. The difference between surfaces and bodies and definitions of each of the degenerate geometric types is discussed in chapter 2. Before covering the specifics of how each degenerate geometry is created, it is best to look at how geometry is stored in VSP and a high-level view of the path taken and function calls made to create degenerate geometry.

Every component type in VSP is a subclass of the `Geom` class. VSP stores each component geometry in instances of a class called `Xsec_surf` as two-dimensional arrays of point locations grouped into cross-sections. In other words, every component at its core is simply a

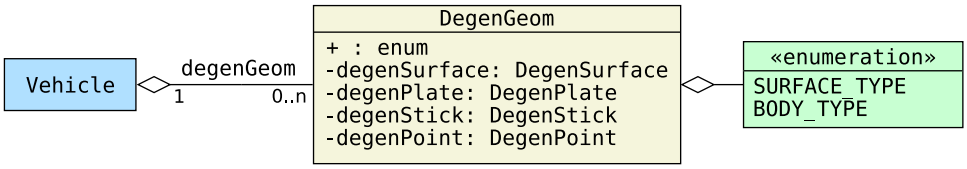

Figure 3.2: Composition of the DegenGeom class in VSP.

collection of point locations stored in an instance of `Xsec_surf`, or a ready-made degenerate surface. The methods to create all types of degenerate geometry except degenerate point belong to `Xsec_surf`. The write-out capabilities are implemented here as well. Degenerate point creation is handled at the `Vehicle` level, since all components need to be visible to calculate wetted volume and area.

VSP has a few items that either don't contain geometry or aren't really components that comprise the vehicle. Because of this they are not part of the degenerate geometry computations. They are provided here with some description:

- `BLANK`: doesn't consist of any geometry. Provided as a convenient way to group components.
- `CABIN_LAYOUT`: allows blocking for internal configurations. Also doesn't consist of any geometry.
- `MESH_GEOM`: a mesh representation of an existing VSP component, created whenever meshing is required (CompGeom, Mass Prop, etc.).

`MS_WING` is a fourth item that requires a little special treatment. `MS_WING` is actually unique among the predefined VSP components in that it has the option for rounded end caps. This wouldn't necessarily be a problem except that the end caps are created using four "cross-sections" that are not in fact planar, cross-sectional slices. Figure 3.3 shows a leading edge view of a wing tip. The gray portion, bounded by the blue and green cross-sections, is the wing tip without an end cap. Adding a rounded end cap, shown in red, causes the red, cyan, magenta, and black "cross-sections" to be added. Since the definitions of degenerate plate and degenerate stick assume that discretized cross-sections are planar slices of underlying geometry, the end cap cross-sections cannot be distilled into these types. To rectify the problem the end caps, if they exist, are removed prior to computation of degenerate surface, degenerate plate, and degenerate stick, and replaced afterward. Adding rounded end caps does not appreciably change the wing span, and hence removing them does not appreciably

Figure 3.3: A leading edge view of a wing tip with rounded end cap. Cross-sections are shown in different colors.

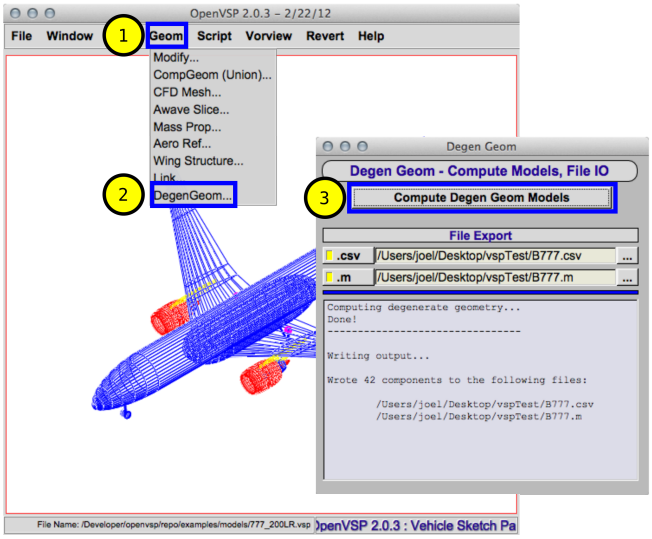

Figure 3.4: Sequence for creating degenerate geometry from the VSP main screen.

change the wing's degenerate representations. The exception of course is degenerate point. Since degenerate point relies on intersections with other components to define wetted volume and area and it is not adversely affected by non-planar cross-sections, wing end caps are retained for its computation.

Accessing degenerate geometry creation and write-out in VSP is very simple. Like other geometric information and conversion capabilities, degenerate geometry is under the Geom menu. The Degen Geom user interface (UI) allows a designer to compute all four degenerate geometry types and to output them in one or both of a comma-separated value (csv) file or a Matlab script (m-file). Figure 3.4 shows how to access these capabilities. Along with the compute button, there are file browser/selector and output enable buttons for both output types. There is also a status display screen, which gives basic information to the user including how many components were written and to what location(s) if output was enabled. The computed degenerate geometry isn't currently accessible except through the

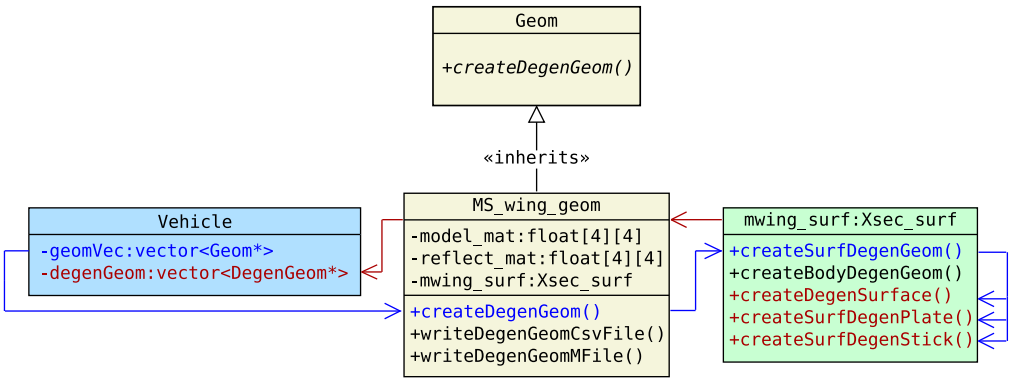

Figure 3.5: VSP's internal degenerate geometry creation process.

output files. However it is stored, so the upcoming API or other future capabilities will almost certainly expose degenerate geometry to access and/or manipulation.

When the user tells VSP to create degenerate geometry, the function calls are passed all the way down to `Xsec_surf`, which returns an instance of `DegenGeom` back up to `Vehicle` to be stored. An overview of this process is shown in figure 3.5 using an `MS_WING` component as an example. The top level `createDegenGeom()` method belongs to `Vehicle`. When it is invoked, `Vehicle` loops through the vector of all geometries, and calls the individual `createDegenGeom()` methods for each component that has its output flag enabled and isn't a `MESH_GEOM`, `BLANK`, or `CABIN_LAYOUT`. If a component is an instance of `MS_WING`, rounded end caps, if there are any, are first removed. Each component's `createDegenGeom()` contains a call to either `createSurfDegenGeom()` or `createBodyDegenGeom()` in its `Xsec_surf`. For instance, `MS_WING` is a surface type component whose points are stored in an instance of `Xsec_surf` called `mwing_surf`. `MS_WING`'s `createDegenGeom()` calls `mwing_surf`'s `createSurfDegenGeom()`. The instance of `DegenGeom` that's eventually passed back to `Vehicle` is created here.

It might be of interest to note that cleanup of `DegenGeom` objects is handled by `Vehicle`'s deconstructor. This allows degenerate geometry to persist as long as `Vehicle` does and since `Vehicle` already holds a vector of references to all degenerate geometries, it can easily release their allocated memory. Note that though degenerate geometry can be out of sync with the current `Vehicle`, there is no danger of accidentally accessing this information since it is recomputed before it is written to file. It should also be noted that two instances of `DegenGeom` are created for any component with symmetry, one for the original component and one for the reflected part.

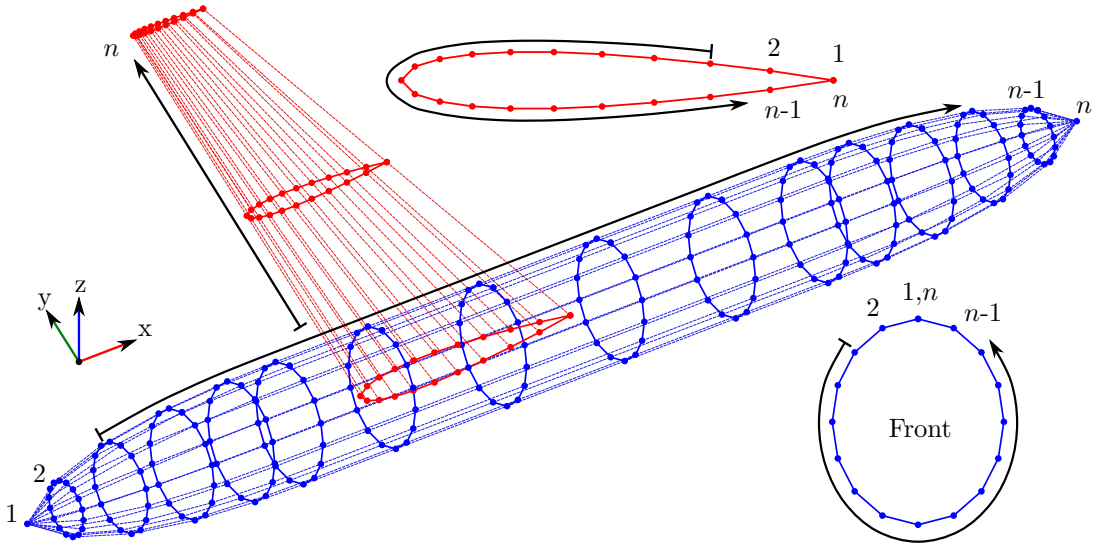

Figure 3.6: VSP unreflected geometry storage ordering for both body and surface component types.

Once an instance of `DegenGeom` has been created, it is tagged as either a `SURFACE_TYPE` or `BODY_TYPE` (see figure 3.2). The appropriate creation methods for degenerate surface, degenerate plate, and degenerate stick are then called. After these three degenerate types are created, a reference to their containing `DegenGeom` is passed back to `Vehicle`, which then moves on to the next component. Once all component geometries have had an associated `DegenGeom` created, modified versions of the `CompGeom` and `MassProp` routines are run, which mesh the entire aircraft and compute all component degenerate points at once.

The following sections will cover, in detail, how each type of degenerate geometry is created and stored. First, VSP's internal storage mechanism should be discussed. As previously mentioned, VSP stores the points comprising each component geometry as a two-dimensional array of cartesian coordinate triplets. Figure 3.6 shows the ordering of points within a cross-section and the ordering of cross-sections within a component for both a `FUSE2` (body) and an `MS_WING` (surface) in their default orientations.

The default orientation for a body component is with its longitudinal axis lying along the x-direction, with cross-section numbering starting at the smallest x value and increasing in the direction of increasing x. Looking along the x-axis in the positive direction, points within a cross-section are numbered starting at the top and increasing in a counter-clockwise direction. Note that the first and last points are coincident.

Since the creation of degenerate geometry relies on certain assumptions (like the default orientation of components), it is important for a designer to understand how the way they

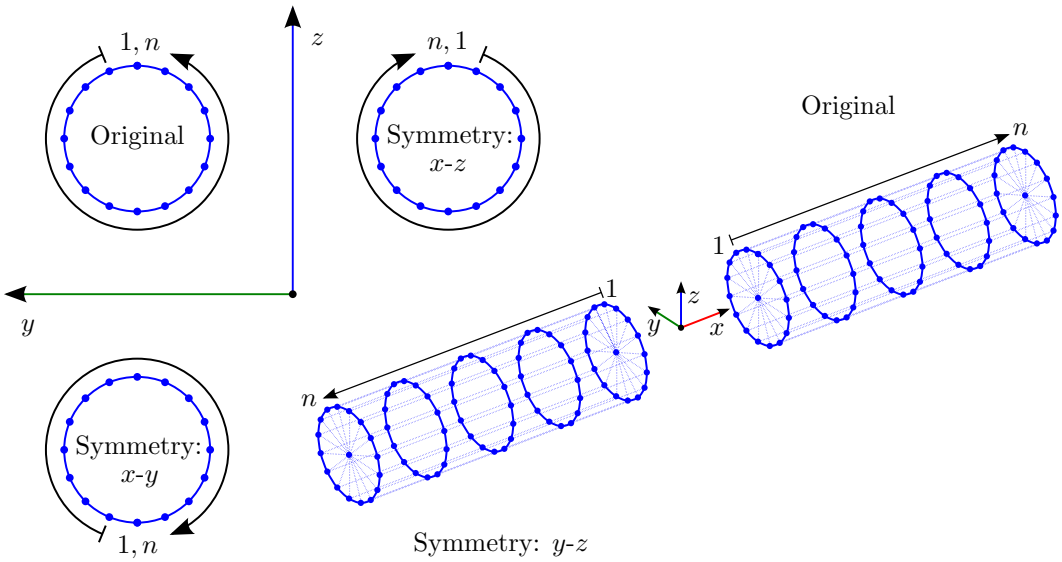

Figure 3.7: Point and cross-section ordering for different types of reflected symmetry using a Fuselage 2 VSP component.

create geometry can affect the validity of the degenerate geometry that results from their design. The numbering system mentioned above is only good for default component orientations. Rotations and reflections will change it. For instance, if a `FUSE2` is reflected across the y-z plane, the cross-section numbering will increase in the negative x-direction, but maintain its intra cross-sectional number order. If instead the same `FUSE2` were reflected across the x-z plane, the cross-section ordering would remain the same, but the point numbering would be reflected (increasing from the top in a clockwise direction when viewed from the front). Figure 3.7 shows this effect for different types of reflected symmetry.

The default orientation for a surface component is with its longitudinal axis lying along the y-direction, with cross-section numbering starting at the smallest magnitude of y values, nominally 0, and increasing along the y-axis either positive or negative. The wing shown in figure 3.6 starts at $y = 0$ and increases cross-section number along the positive y-direction. If it had a reflected mate (the left wing, assuming reflection across the x-z plane), that component would start at $y = 0$ and increase cross-section number along the *negative* y-direction. Points within surface cross-sections are numbered starting at the trailing edge and increasing across the top surface to the leading edge around the bottom surface back to the trailing edge. Once again, the first and last points are coincident.

Though it is assumed that aircraft is oriented such that the freestream (at zero angle of attack) is in the x-direction, it is important to note that the trailing edge of a wing (part of

degenerate stick's definition) is defined by its default orientation and so does not necessarily correspond to the downwind side. Just like the `FUSE2` example shown in figure 3.7, reflection across various planes (choosing a symmetry option in VSP) can affect point or cross-section ordering. If a wing is reflected across the y-z plane for instance, the "trailing" edge will actually be the upwind side. The point numbering is the same, namely "trailing" edge to "leading" edge to "trailing" edge, top to bottom. The same is true if a wing is rotated, say 180 degrees, about the z-axis. The points will be reported in their correct, rotated locations, but the "leading" edge points will actually be downwind and cross-section numbering will increase along the negative y-direction.

Each cross-section within a given component has the same number of points that define it, though this number can vary from component to component. This means that the fuselage beginning and end "points" labeled in figure 3.6 as cross-sections 1 and n actually contain 17 points each overlaying one another.

Additionally, though components can be rotated and translated from their default orientation, these movements don't affect how and where the points are stored. All rotation and translation information is stored in a 4×4 matrix for each component at the `Geom` level (this is shown for an `MS_WING` in figure 3.5 as the matrices `model_mat` and `reflect_mat`, for the primary half of the component and any reflected symmetry, respectively). The points in a cross-section can move (changing/scaling an airfoil section, for instance), but component-level rotation and/or translation does not affect the stored points. This is a double-edged sword. On the one hand it is incredibly important for degenerate geometry computation, as one can always assume component orientation and certain properties, which degenerate geometry relies on. On the other hand, unintended consequences can occur, like rotating a wing 180 degrees so that the leading edge is downstream and then basing subsequent analysis on the "leading" and "trailing" edges from degenerate stick.

The only real caveat is one that can be broadly applied to aircraft or any other engineering design: the designer must be aware of a process' underlying assumptions and what effect, if any, violation of those assumptions results in. He or she must also be aware of enough of a process' detail to know what's happening "under the hood", lest poor or invalid results are obtained. With this in mind, degenerate geometry computation will be looked at in detail, starting with the simplest, degenerate surface.

3.1.1 DegenSurface

Degenerate surface (defined in section 2.1) requires almost no computation as the majority of information is simply a record how VSP already stores its components internally (see figure 3.6). Degenerate surface information is stored in a C/C++ structure called `DegenSurface` that is a member variable of the class `DegenGeom`. The code definition is:

```
typedef struct {
        vector< vector<vec3d> > x;
        vector< vector<vec3d> > nvec;
        vector<double>          u;
        vector<double>          w;
} DegenSurface;
```

where `x` and `nvec` are two-dimensional arrays (actually `vectors` of `vectors`) of node locations and node normal vector components, respectively. If there are p cross-sections and q points per cross section, both `x` and `nvec` will be size $p \times q$. Both the cartesian coordinate triplets and normal vectors are of type `vec3d`, which is a VSP storage class specifically created to store coordinates or vector components. It has a two-dimensional analog called `vec2d`. Both of these storage classes have methods like dot and cross-product (3D only), normalization, magnitude, etc. The vectors `u` and `w` are VSP parametric coordinates, and are of length p and q, respectively. Each cross-section has a parametric coordinate u and each point within a cross-section has a parametric coordinate w, which is the same for a given point (say the second) in all cross-sections.

Recording `x`, `u`, and `w` is merely a matter of iterating over their respective variables and copying the information into corresponding `DegenSurface` members. Calculating `nvec`, as discussed in section 2.1, is a bit more involved and depends on whether or not a component is the original or reflected. Figure 3.8 shows an unreflected airfoil section, which is a surface type, with an example normal vector. The surface normal definition does not hinge on surface/body designation, but whether or not the component is reflected, so it is important to note that the component shown is not a reflected part.

At each degenerate surface node where a normal vector is to be recorded, two vectors are defined, labeled in figure 3.8 as $\mathbf{t}_1$ and $\mathbf{t}_2$. The inset shows the points that bound a given surface patch and how these nodes relate to t_1 and t_2. From the figure, the following

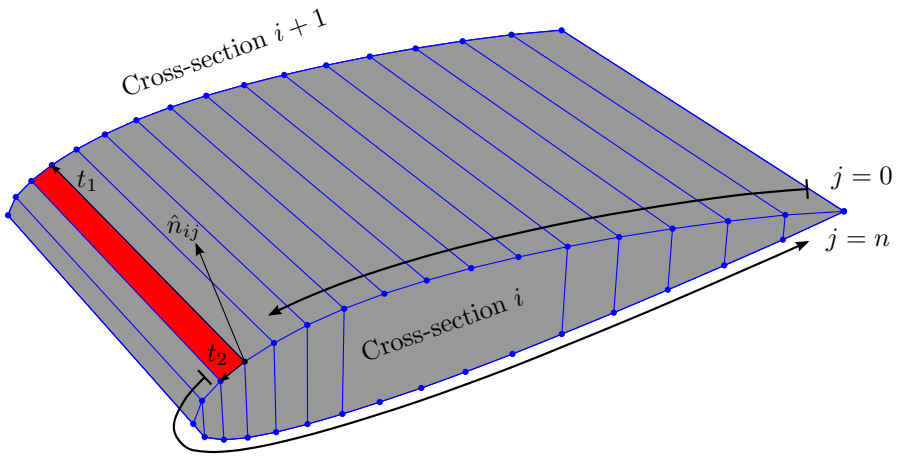

Figure 3.8: An example degenerate surface normal vector on a wing section.

definitions should be obvious:

$$\mathbf{t}_1 = x(i+1,j) - x(i,j) \tag{3.1}$$

$$\mathbf{t}_2 = x(i,j+1) - x(i,j) \tag{3.2}$$

The unit normal vector corresponding to node $x(i,j)$ is computed via vector cross product

$$n(i,j) = \frac{t_1 \times t_2}{\|t_1 \times t_2\|} \tag{3.3}$$

where the double vertical bars denote the vector norm.

Since each surface normal looks at adjacent nodes that are "forward" in point number or cross-section there is one less normal vector than node for each cross-section and no normal vectors for the last cross-section. Though not necessary, these "missing" vectors are padded with `NAN`s, making output file formatting uniform and easier to carry out.

For reflected parts one of the following things is true with respect to default orientations:

1. The numbering of nodes within cross-sections is reversed, or
2. The cross-section numbering is reversed

meaning that equation (3.3) gives inward-facing normals instead of outward-facing. The remedy is to cross $\mathbf{t}_1$ and $\mathbf{t}_2$ in the reverse order for all reflected components.

3.1.2 DegenPlate

Degenerate plate, defined in section 2.2, shares some similarities with degenerate surface in its code definition:

```
typedef struct {
    vector< vector<vec3d>  > x;
    vector< vector<double> > zcamber;
    vector< vector<vec3d>  > nCamber;
    vector< vector<double> > t;
    vector<vec3d>            nPlate;
    vector<double>           u;
    vector<double>           wTop;
    vector<double>           wBot;
} DegenPlate;
```

Both have a two-dimensional array (`vector` of `vectors`) holding node locations organized by cross-section, both have normal vectors, and both store the VSP parametric coordinates u and w. Unique to degenerate plate are two-dimensional arrays for distance to camber line from node (`zcamber`) and sectional thickness at node (`t`). It also has two sets of normal vectors, camber line and plate, and two different vectors of w coordinates, top and bottom.

Degenerate plate node locations are based on degenerate surface nodes. They are in fact camber line points projected onto a chord line. Because the leading and trailing edge points of an airfoil define the chord line, if the underlying geometry is a surface type component, then the leading and trailing edge points from degenerate surface are the same as the first and last points on the degenerate plate. If a cross-section is composed of q nodes, then there will be $r = (q+1)/2$ degenerate plate nodes and (using one-based indexing)

$$x_{plate}(i,1) = x_{surf}(i,1) \qquad x_{plate}(i,r) = x_{surf}(i,r) \tag{3.4}$$

where the index i indicates the ith cross-section. Note that the first node is the degenerate surface trailing edge and the rth node is the leading edge (see figure 3.6).

Recall that for a body type component, two plates are created. The following discussion treats first and second plate variables as separate, but the two are actually concatenated, making `DegenPlate` member variables twice as long for a body as for a surface.

Equation 3.4 holds for the (nominally) vertical plate. For the horizontal plate, the chord is rotated by one-quarter, or $s = (q-1)/4$, of the nodes, so the "trailing" and "leading"

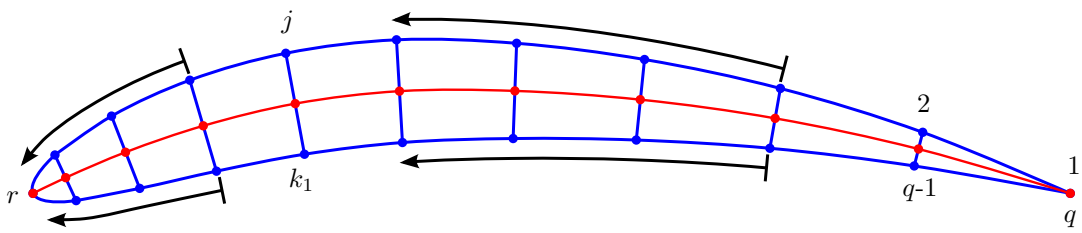

Figure 3.9: Ordering of degenerate surface nodes and computation of camber line points on an airfoil section.

edge points are respectively given by

$$x_{plate}(i,1) = x_{surf}(i,1+s) \qquad\qquad x_{plate}(i,r) = x_{surf}(i,r+s) \tag{3.5}$$

To compute the intermediate plate node locations, camber line nodes must first be obtained. A mean camber line is formed by the locus of points midway between the upper and lower surfaces, as measured normal to the mean camber line [27]. Fortunately, obtaining the points that define a section's mean camber line, whether surface or body type, merely entails marching from one side of a section to the other and finding the midpoint locations between pairs of opposing points.

Figure 3.9 provides a more clear description for an airfoil (surface) section. If camber line nodes are denoted x_{camb} and degenerate surface nodes x_{surf} then the jth camber line node of the ith cross-section is given by

$$x_{camb}(i,j) = \frac{1}{2}\left[x_{surf}(i,j) + x_{surf}(i,k_1)\right] \ ; \qquad\qquad j \in \{2,3,\ldots,r-1\} \tag{3.6}$$

where $k_1 = (q+1) - j$. Note that j only runs from node 2 to node $r-1$ because the first and last camber line points are coincident with the degenerate surface nodes.

Figure 3.10 shows the two degenerate plates created for a circular, body type cross-section along with point indexing. Equation 3.6 holds for the first, vertical cross-section, but for the second, the node indexing must be modified. The camber line points are instead given by

$$x_{camb}(i,j) = \frac{1}{2}\left[x_{surf}(i,j+s) + x_{surf}(i,k_2)\right] \ ; \qquad\qquad j \in \{2,3,\ldots,r-1\} \tag{3.7}$$

where $k_2 = \{(s+q-j) \mod (q-1)\} + 1$. Though it's not entirely clear upon inspection, this ensures that $k_2 \in \{s, s-1, \ldots, 1, q-1, q-2, \ldots, r+s+1\}$.

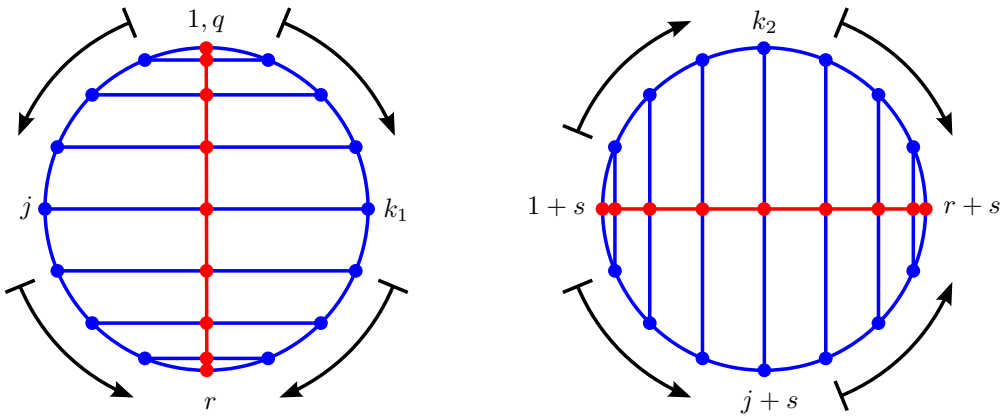

Figure 3.10: Point indexing used in creating two degenerate plates for body type components.

The next step is to project the camber line points onto the plate or plates, depending on component type. For a surface this is the sectional chord line, or that line drawn between the leading and trailing edge nodes. For a body, there are two: one described by the same formulae as for a surface, and one for the second, nominally orthogonal plate. The following steps will be described in terms of a surface component, but with two equations listed at every step. The first will be labeled with a subscript 1, and is for the surface, as described and also used to compute the first plate for a body type component. The second, labeled with a subscript 2 will exclusively pertain to body type components and be for computing the second plate.

Figure 3.11 shows how camber line nodes are projected onto a section's chord line. For each camber line node, a vector from the trailing edge to that node is given by

$$\mathbf{a}_1 = x_{camb}(i,j) - x_{surf}(i,1)\ ; \qquad j \in \{2,3,\ldots,r-1\} \tag{3.8}$$

$$\mathbf{a}_2 = x_{camb}(i,j+s) - x_{surf}(i,1+s)\ ; \qquad j \in \{2,3,\ldots,r-1\} \tag{3.9}$$

A unit vector pointing from the trailing edge to the leading edge is given by

$$\hat{c}_1(i) = \frac{x_{surf}(i,r) - x_{surf}(i,1)}{\|x_{surf}(i,r) - x_{surf}(i,1)\|} \tag{3.10}$$

$$\hat{c}_2(i) = \frac{x_{surf}(i,r+s) - x_{surf}(i,1+s)}{\|x_{surf}(i,r+s) - x_{surf}(i,1+s)\|} \tag{3.11}$$

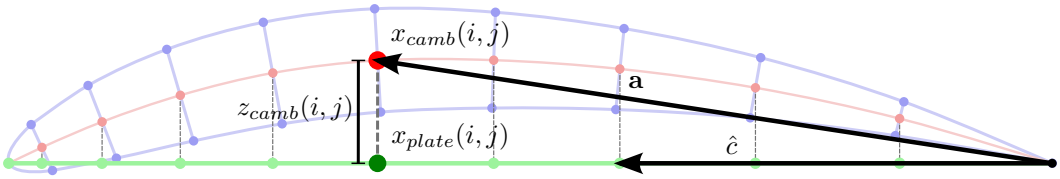

Figure 3.11: Computation of degenerate plate nodes from camber line nodes via vector projection.

and the jth plate point is computed via

$$x_{plate,1}(i,j) = x_{surf}(i,1) + (\mathbf{a}_1 \cdot \hat{c}_1(i))\,\hat{c}_1(i) \tag{3.12}$$

$$x_{plate,2}(i,j) = x_{surf}(i,1+s) + (\mathbf{a}_2 \cdot \hat{c}_2(i))\,\hat{c}_2(i) \tag{3.13}$$

The distances from these nodes to their corresponding camber line nodes, stored in the member variable `zcamber`, are computed for either surface or body components via

$$z_{camb}(i,j) = \|x_{camb}(i,j) - x_{plate}(i,j)\| \tag{3.14}$$

The values of the first and last point are set to zero, since by definition, the degenerate plate nodes at these locations coincide with the first and last camber line nodes.

Thickness at each degenerate plate node is simply the distance between corresponding degenerate surface "upper" and "lower" nodes. Thickness at the jth degenerate plate node is

$$t_1(i,j) = \|x_{surf}(i,j) - x_{surf}(i,k_1)\| \tag{3.15}$$

$$t_2(i,j) = \|x_{surf}(i,j+s) - x_{surf}(i,k_2)\| \tag{3.16}$$

where the indexes k_1 and k_2 are given by

$$k_1 \in \{q-1, q-2, \ldots, r+1\} \tag{3.17}$$

$$k_2 \in \{s, s-1, \ldots, 1, q-1, q-2, \ldots, r+s+1\} \tag{3.18}$$

Once again, the first and last points are set equal to zero.

The camber line normal vectors for the first and last node location are both set to the zero vector, since there is no defined direction from a location to itself. For the intermediate nodes, camber line normal vectors are in the direction of the thicknesses or

$$\hat{n}_{camb,1}(i,j) = \frac{x_{surf}(i,j) - x_{surf}(i,k_1)}{\|x_{surf}(i,j) - x_{surf}(i,k_1)\|} \tag{3.19}$$

$$\hat{n}_{camb,2}(i,j) = \frac{x_{surf}(i,j+s) - x_{surf}(i,k_2)}{\|x_{surf}(i,j+s) - x_{surf}(i,k_2)\|} \tag{3.20}$$

where k_1 and k_2 are given by equations (3.17) and (3.18), respectively.

Each cross-section has a plate normal vector that points toward the "top" of the original surface. Referring to figures 3.9 and 3.10, this is the j or $j+s$ side.

$$\hat{n}_{plate}(i) = \frac{\hat{c}(i) \times \hat{n}_{area}(i)}{\|\hat{c}(i) \times \hat{n}_{area}(i)\|} \tag{3.21}$$

where $\hat{c}$ is the chord vector, given by equation (3.10) or (3.11) and the cross-sectional area normal vector $\hat{n}_{area}$ is defined by the process given in section 3.1.3.

The VSP parametric coordinate u has one value for each cross-section and hence one value for each degenerate plate section. VSP stores these parametric coordinates in a vector called `uArray`, and degenerate plate merely copies it over for each node (twice in the case of a body type component)

$$u(i,j) = uArray(i)\ ; \qquad\qquad j \in \{1, 2, \ldots, r\} \tag{3.22}$$

Since each degenerate plate node corresponds to two nodes from the original VSP geometry, degenerate plate reports two w coordinates at each node, w_{top} and w_{bot}. These are obtained from the VSP variable `wArray` with the following formulae

$$w_{top,1}(i,j) = wArray(j) \qquad\qquad w_{bot,1}(i,j) = wArray(k_1) \tag{3.23}$$

$$w_{top,2}(i,j) = wArray(j+s) \qquad\qquad w_{bot,2}(i,j) = wArray(k_2) \tag{3.24}$$

where once again $j \in \{1, 2, \ldots, r\}$. The indexes k_1 and k_2 are expanded so the first and last plate nodes are captured

$$k_1 \in \{q, q-1, \ldots, r\} \qquad\qquad k_2 \in \{1+s, s, \ldots, 1, q-1, \ldots, r+s\} \tag{3.25}$$

For the first plate (or only surface plate), since there are two coincident nodes at the trailing edge, the top and bottom values for w at the first degenerate plate node will be unique, specifically 0 and 1, respectively. The final degenerate plate node, however is assigned from the location of the leading edge point, and hence the top and bottom values of w will be identical. For the second plate, the first and last nodes have only a single value of w and so w_{top} and w_{bot} are identical. If the index k_2 from equation (3.25) is examined, it is apparent that for the coincident node location halfway along the second degenerate plate (the two original "trailing" edge points) the value of w_{bot} is set to 0.

3.1.3 DegenStick

Degenerate stick is defined in section 2.3. Though its representation of underlying geometry is a reduction in fidelity from other degenerate forms, its member variable list is expanded:

```
typedef struct {
    vector<vec3d>                xle;
    vector<vec3d>                xte;
    vector<double>               toc;
    vector<double>               tLoc;
    vector<double>               chord;
    vector<double>               sweep;
    vector< vector<double> >     Ishell;
    vector< vector<double> >     Isolid;
    vector<vec3d>                xcgSolid;
    vector<vec3d>                xcgShell;
    vector<double>               area;
    vector<vec3d>                areaNormal;
    vector<double>               perimTop;
    vector<double>               perimBot;
    vector<double>               u;
} DegenStick;
```

Using the same convention as section 3.1.2, all discussion of degenerate stick variable computation will focus on surface type components, but with two sets of equations listed. The first, labeled with a subscript 1 will pertain to the single surface component degenerate stick and the first degenerate stick for a body type component. The second, labeled with a subscript 2, will pertain to the second body type component degenerate stick.

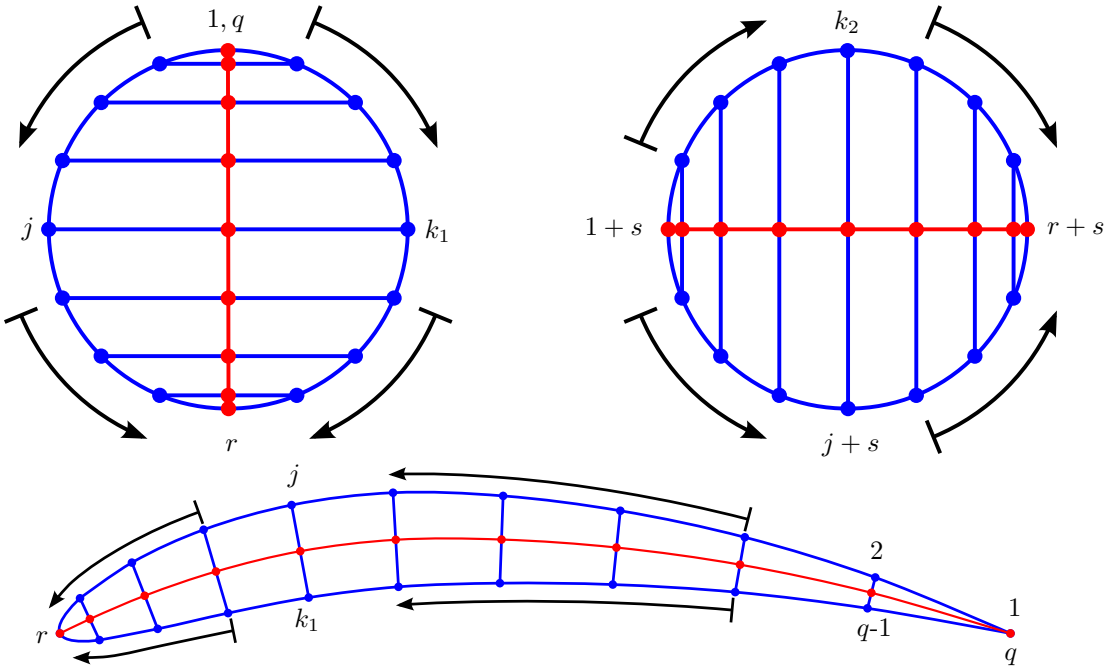

Figure 3.12: Traversal of surface nodes to obtain maximum thickness for both fuselage and airfoil cross-sections.

The variables `xle` and `xte` refer to sectional leading and trailing edges, respectively. They are obtained for the ith cross-section from the degenerate surface representation via

$$x_{le,1}(i) = x_{surf}(i, r) \qquad x_{le,2}(i) = x_{surf}(i, r+s) \tag{3.26}$$

$$x_{te,1}(i) = x_{surf}(i, 1) \qquad x_{te,2}(i) = x_{surf}(i, 1+s) \tag{3.27}$$

where the each cross-section has q nodes and

$$r = \frac{1}{2}(q+1) \qquad s = \frac{1}{4}(q-1) \tag{3.28}$$

Maximum thickness to chord, stored in the variable `toc`, is obtained by traversing pairs of degenerate surface nodes to obtain thicknesses, and then dividing the maximum value by the chord length. This traversal and thickness calculation is performed in the same manner as for degenerate plate. Figures 3.9 and 3.10 are repeated here as figure 3.12 for reference.

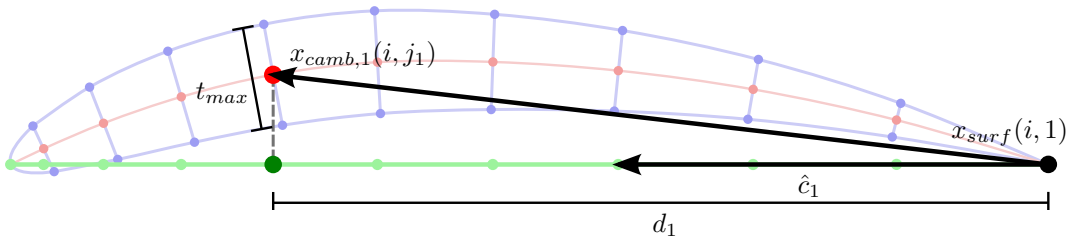

Figure 3.13: Calculation of maximum thickness location along chord for an airfoil section.

Chord is found via

$$c_1(i) = \|x_{surf}(i, r) - x_{surf}(i, 1)\| \tag{3.29}$$

$$c_2(i) = \|x_{surf}(i, r + s) - x_{surf}(i, 1 + s)\| \tag{3.30}$$

where r and s are given by equation (3.28). Maximum thickness to chord is obtained by

$$toc_1(i) = \frac{\max\{\|x_{surf}(i, j) - x_{surf}(i, k_1)\|\}}{c_1(i)} \tag{3.31}$$

$$toc_2(i) = \frac{\max\{\|x_{surf}(i, j + s) - x_{surf}(i, k_2)\|\}}{c_2(i)} \tag{3.32}$$

where $j \in \{2, 3, \ldots, r - 1\}$ and k_1 and k_2 are given by equations (3.17) and (3.18), respectively.

The maximum thickness location is given in percent chord and is found by first projecting the degenerate surface camber line point corresponding to the maximum thickness location down onto the sectional chord line as shown in figure 3.13. This gives the distance from the trailing edge to the maximum thickness location along the chord. If the camber line point corresponding to the max thickness locations for the two sticks are given by $x_{camb,1}(i, j_1)$ and $x_{camb,2}(i, j_2)$, then the distances d_1 and d_2 are found via

$$d_1(i) = [x_{camb,1}(i, j_1) - x_{surf}(i, 1)] \cdot \hat{c}_1(i) \tag{3.33}$$

$$d_2(i) = [x_{camb,2}(i, j_2) - x_{surf}(i, 1 + s)] \cdot \hat{c}_2(i) \tag{3.34}$$

where $\hat{c}_1$ and $\hat{c}_2$ are given by equations (3.10) and (3.11), respectively. Thickness location in percent chord is then given by

$$tLoc_1(i) = 1 - \frac{d_1(i)}{c_1(i)} \tag{3.35}$$

$$tLoc_2(i) = 1 - \frac{d_2(i)}{c_2(i)} \tag{3.36}$$

Quarter-chord sweep angle, like the degenerate surface normal vectors (section 3.1.1) depends on adjacent cross-sections for their computation, and hence there will be one less angle than the number of cross-sections reported. If there are p cross-sections, the following formulae assume $i \in \{1, 2, \ldots, p-1\}$. Computation starts with finding the coordinate location of the current section quarter chord via

$$qc_{crnt,1}(i) = x_{surf}(i, 1) + \frac{3}{4}c_1(i)\hat{c}_1(i) \tag{3.37}$$

$$qc_{crnt,2}(i) = x_{surf}(i, 1+s) + \frac{3}{4}c_2(i)\hat{c}_2(i) \tag{3.38}$$

and the quarter chord location of the adjacent cross-section via

$$qc_{next,1}(i) = x_{surf}(i+1, 1) + \frac{3}{4}c_1(i+1)\hat{c}_1(i+1) \tag{3.39}$$

$$qc_{next,2}(i) = x_{surf}(i+1, 1+s) + \frac{3}{4}c_2(i+1)\hat{c}_2(i+1) \tag{3.40}$$

The next step is to get a vector from the current to adjacent cross-section quarter chord location and to zero out the z-component. The reasoning is the same given in previous sections, namely that it is assumed the upstream location is in the $-x$-direction, and that z is up. For aerodynamic purposes, the sweep angle is then confined to the x-y plane and is that angle off of the y- or $-y$-axis (depending on which direction is outboard) toward the x-axis.

Next, the signed angle between this vector and the appropriate side of the y-axis is obtained. If the y-component of the vector to the next cross-section is negative, the appropriate axis is the negative y-axis and the angle sign convention is positive counter-clockwise about the z-axis. If instead it is positive, the appropriate axis is the positive y-axis and the angle sign convention is positive clockwise about the z-axis. Figure 3.14 shows inboard section sweep angles for both a left and right wing. Angles are reported in degrees, with the pth sweep angle set to `NAN`.

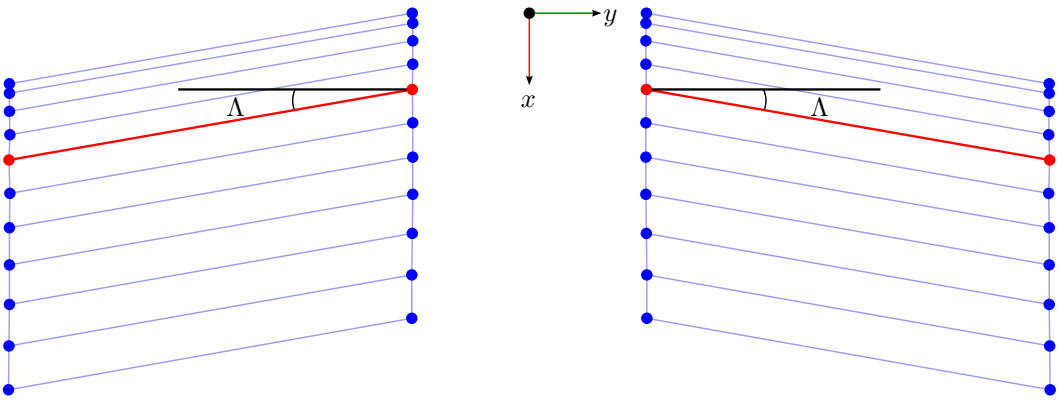

Figure 3.14: Sweep angle at wing quarter chord.

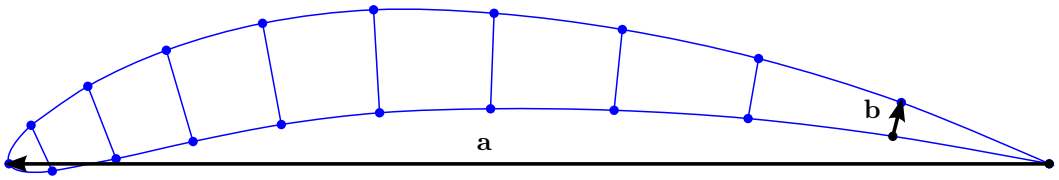

Figure 3.15: Airfoil section showing vectors used in computation of cross-section area normal direction.

Cross-section area normal vectors are computed by taking the cross product of two vectors within a section. The choice is arbitrary since any two vectors in a planar cross-section will give a vector normal to that section. For consistency, a chord vector pointing from the trailing to leading edge and a vector between the bottom and top points directly adjacent to the trailing edge were chosen, as shown in figure 3.15. The area normal vector is $\hat{n}_{area} = \mathbf{b} \times \mathbf{a}$, where

$$\mathbf{a}(i) = x_{surf}(i, r) - x_{surf}(i, 1) \tag{3.41}$$

$$\mathbf{b}(i) = x_{surf}(i, 2) - x_{surf}(i, q - 1) \tag{3.42}$$

These formulae hold for both surface and body type components.

Cross-section area and moments of inertia and center of gravity location for solid cross-sections are found using the work of Carsten Steger [23] as discussed in section 2.3. In order to calculate cross-section area, the points are first rotated (denoted by the prime symbol) so that the area normal aligns with the y-axis, ensuring that the cross-section is entirely in

the x-z plane. The area is then found via

$$a(i) = \frac{1}{2}\sum_{j=1}^{q-1} x'_{surf,x}(i,j)x'_{surf,z}(i,j+1) - x'_{surf,x}(i,j+1)x'_{surf,z}(i,j) \tag{3.43}$$
$$+ \frac{1}{2}\left[x'_{surf,x}(i,q)x'_{surf,z}(i,1) - x'_{surf,x}(i,1)x'_{surf,z}(i,q)\right]$$

where the subscripts x and z denote those respective coordinate components.

For solid components, both surface and body type, the cross-sectional center of gravity is found by first rotating the degenerate surface nodes into the x-z plane. The cg components are then found via

$$\bar{x}'(i) = \frac{1}{6}\sum_{j=1}^{q-1}\left[x'_{surf,x}(i,j) + x'_{surf,x}(i,j+1)\right] \tag{3.44}$$
$$+ \frac{1}{6}\left[x'_{surf,x}(i,q) + x'_{surf,x}(i,1)\right]$$
$$\bar{z}'(i) = \frac{1}{6}\sum_{j=1}^{q-1}\left[x'_{surf,z}(i,j) + x'_{surf,z}(i,j+1)\right] \tag{3.45}$$
$$+ \frac{1}{6}\left[x'_{surf,z}(i,q) + x'_{surf,z}(i,1)\right]$$

The y-component is identical for all nodes since the cross-section was rotated into the x-z plane, so $\bar{y}' = x'_{surf,y}(i,1)$ is used. The points are then rotated back into their original orientation, giving the correct center of gravity location.

Three moments of inertia are reported for solid components. They are two bending moments and one torsional. For a surface type component the two bending moments of inertia are about the x- and z-axes and represent resistance to bending due to drag and lift, respectively. The torsional moment of inertia is about the y-axis and represents resistance due to a pitching moment. These are computed via

$$I_{xx}(i) = \frac{a(i)}{12}\sum_{j=1}^{q-1}\left[x^2_{surf,z}(i,j) + x_{surf,z}(i,j)x_{surf,z}(i,j+1) + x^2_{surf,z}(i,j+1)\right] \tag{3.46}$$
$$+ \frac{a(i)}{12}\left[x^2_{surf,z}(i,q) + x_{surf,z}(i,q)x_{surf,z}(i,1) + x^2_{surf,z}(i,1)\right] + a(i)\bar{z}^2(i)$$
$$I_{zz}(i) = \frac{a(i)}{12}\sum_{j=1}^{q-1}\left[x^2_{surf,x}(i,j) + x_{surf,x}(i,j)x_{surf,x}(i,j+1) + x^2_{surf,x}(i,j+1)\right] \tag{3.47}$$
$$+ \frac{a(i)}{12}\left[x^2_{surf,x}(i,q) + x_{surf,x}(i,q)x_{surf,x}(i,1) + x^2_{surf,x}(i,1)\right] + a(i)\bar{x}^2(i)$$
$$I_{yy}(i) = I_{xx}(i) + I_{zz}(i) \tag{3.48}$$

where $a(i)$ is given by equation (3.43), and $\bar{x}(i)$ and $\bar{z}(i)$ by the rotated results of equations (3.44) and (3.45), respectively.

A similar process is followed for body type components except that the bending moments of inertia are about the y- and z-axes and torsion is about the x-axis, and so are computed via

$$I_{yy}(i) = \frac{a(i)}{12}\sum_{j=1}^{q-1}\left[x^2_{surf,z}(i,j) + x_{surf,z}(i,j)x_{surf,z}(i,j+1) + x^2_{surf,z}(i,j+1)\right] \tag{3.49}$$
$$+ \frac{a(i)}{12}\left[x^2_{surf,z}(i,q) + x_{surf,z}(i,q)x_{surf,z}(i,1) + x^2_{surf,z}(i,1)\right] + a(i)\bar{z}^2(i)$$
$$I_{zz}(i) = \frac{a(i)}{12}\sum_{j=1}^{q-1}\left[x^2_{surf,y}(i,j) + x_{surf,y}(i,j)x_{surf,y}(i,j+1) + x^2_{surf,y}(i,j+1)\right] \tag{3.50}$$
$$+ \frac{a(i)}{12}\left[x^2_{surf,y}(i,q) + x_{surf,y}(i,q)x_{surf,y}(i,1) + x^2_{surf,y}(i,1)\right] + a(i)\bar{y}^2(i)$$
$$I_{xx}(i) = I_{yy}(i) + I_{zz}(i) \tag{3.51}$$

The area is computed by rotating the cross-section into the y-z plane:

$$a(i) = \frac{1}{2}\sum_{j=1}^{q-1} x'_{surf,y}(i,j)x'_{surf,z}(i,j+1) - x'_{surf,y}(i,j+1)x'_{surf,z}(i,j) \tag{3.52}$$
$$+ \frac{1}{2}\left[x'_{surf,y}(i,q)x'_{surf,z}(i,1) - x'_{surf,y}(i,1)x'_{surf,z}(i,q)\right]$$

The cross-sectional centroid z coordinate is given by equation (3.45), y-coordinate is

$$\bar{y}(i) = \frac{1}{6}\sum_{j=1}^{q-1}\left[x_{surf,y}(i,j) + x_{surf,y}(i,j+1)\right] \tag{3.53}$$
$$+ \frac{1}{6}\left[x_{surf,y}(i,q) + x_{surf,y}(i,1)\right]$$

and this time the x-coordinate is $\bar{y}' = x'_{surf,x}(i,1)$.

Moments of inertia of thin-walled components are computed as functions of wall-thickness and are reported as coefficients of these functions. The description of underlying assumptions and relevant formulae are given in section 2.3 and equations (2.28), (2.29), and (2.30). Figure 2.9 shows how the thin section is discretized into rectangles of thickness t. Assuming that the index $q+1$ refers to node 1, each line segment connecting adjacent degenerate

surface nodes can be expressed as the vector

$$\mathbf{b}(i,j) = x_{surf}(i,j+1) - x_{surf}(i,j) \tag{3.54}$$

The magnitude of this vector is

$$b(i,j) = \|x_{surf}(i,j+1) - x_{surf}(i,j)\| \tag{3.55}$$

The distance from the midpoint between each set of adjacent nodes to the cross-section center of gravity

$$d(i,j) = \frac{1}{2}\left[x_{surf}(i,j+1) + x_{surf}(i,j)\right] - \bar{x}(i) \tag{3.56}$$

The moment of inertia function coefficients for a surface component are then found via

$$A_1(i) = \frac{1}{24}\sum_{j=1}^{q}\left\{b(i,j)\left[1+\cos\left(2\phi(i,j)\right)\right]\right\} \tag{3.57}$$

$$A_2(i) = \frac{1}{24}\sum_{j=1}^{q}\left\{b^3(i,j)\left[1-\cos\left(2\phi(i,j)\right)\right] + b(i,j)d_z^2(i,j)\right\} \tag{3.58}$$

$$A_3(i) = \frac{1}{24}\sum_{j=1}^{q}\left\{b(i,j)\left[1-\cos\left(2\phi(i,j)\right)\right]\right\} \tag{3.59}$$

$$A_4(i) = \frac{1}{24}\sum_{j=1}^{q}\left\{b^3(i,j)\left[1+\cos\left(2\phi(i,j)\right)\right] + b(i,j)d_x^2(i,j)\right\} \tag{3.60}$$

$$A_5(i) = A_1(i) + A_3(i) \tag{3.61}$$

$$A_6(i) = A_2(i) + A_4(i) \tag{3.62}$$

where $\phi(i,j)$ is the angle of $\mathbf{b}(i,j)$ with respect to the positive x-axis. The bending and torsional moments of inertia can be recovered if wall thickness t is specified via

$$I_{xx}(i) = A_1(i)t^3 + A_2(i)t \tag{3.63}$$

$$I_{zz}(i) = A_3(i)t^3 + A_4(i)t \tag{3.64}$$

$$I_{yy}(i) = A_5(i)t^3 + A_6(i)t \tag{3.65}$$

For a body type component, the moments of inertia are instead calculated in the y-z plane. The function coefficients A_1-A_3, A_5, and A_6 are identical except that $\phi(i,j)$ is now

the angle of $\mathbf{b}(i,j)$ off of the positive y-axis and the remaining function coefficient is

$$A_4(i) = \frac{1}{24}\sum_{j=1}^{q}\left\{b^3(i,j)\left[1+\cos\left(2\phi(i,j)\right)\right] + b(i,j)d_y^2(i,j)\right\} \tag{3.66}$$

The two bending and one torsional moment of inertia are now about the y-, z-, and x-axes, respectively and are recovered via

$$I_{yy}(i) = A_1(i)t^3 + A_2(i)t \tag{3.67}$$

$$I_{zz}(i) = A_3(i)t^3 + A_4(i)t \tag{3.68}$$

$$I_{xx}(i) = A_5(i)t^3 + A_6(i)t \tag{3.69}$$

The cross-sectional center of gravity for shell components is found by first rotating the degenerate surface nodes into the x-z plane (once again denoted by the prime symbol). The cg components are then found via

$$\bar{x}'(i) = \frac{\sum_{j=1}^{q}\frac{1}{2}\left[x'_{surf,x}(i,j) + x'_{surf,x}(i,j+1)\right]b(i,j)}{\sum_{j=1}^{q}b(i,j)} \tag{3.70}$$

$$\bar{z}'(i) = \frac{\sum_{j=1}^{q}\frac{1}{2}\left[x'_{surf,z}(i,j) + x'_{surf,z}(i,j+1)\right]b(i,j)}{\sum_{j=1}^{q}b(i,j)} \tag{3.71}$$

The y-component is identical for all nodes since the cross-section was rotated into the x-z plane, so $\bar{y} = x_{surf,y}(i,1)$ is used. The points are then rotated back into their original orientation, giving the correct center of gravity location.

The top and bottom perimeters are found by summing up line segments connecting adjacent degenerate surface nodes within a cross-section, or

$$p_{top}(i) = \sum_{j=1}^{r-1} b(i,j) + \frac{1}{2}b(i,q) \tag{3.72}$$

$$p_{bot}(i) = \sum_{j=r}^{q-1} b(i,j) + \frac{1}{2}b(i,q) \tag{3.73}$$

Once again the VSP parametric coordinate u is simply copied over for each cross-section

$$u(i) = \texttt{uArray[i]}$$

where `uArray` is the VSP variable for storing the parametric coordinates u.

3.1.4 DegenPoint

The final degenerate geometry type is degenerate point, which is defined in section 2.4. The code definition is:

```
typedef struct {
    vector< double >              vol;
    vector< double >              volWet;
    vector< double >              area;
    vector< double >              areaWet;
    vector< vector< double > >    Ishell;
    vector< vector< double > >    Isolid;
    vector< vec3d >               xcgShell;
    vector< vec3d >               xcgSolid;
} DegenPoint;
```

Degenerate point properties require discretizing components into tetrahedra or triangular surface meshes. For the wetted area and volume, all components' meshes need to be intersected. Reference [24] gives a treatment of how to discretize arbitrary polyhedra into tetrahedra.

Fortunately, the routines to calculate degenerate point's member variables already existed in VSP and had been in use for some time. The volume and area and their wetted counterparts were obtained using the `CompGeom` routines. The moments of inertia and cg location came from a modified version of the `MassProp` routines that calculated inertia per density (ρ) for the solid and per surface density (ρt) for the shell.

3.2 Components Covered

VSP currently has 12 components available for use from the geometry browser window. All 12 are subclasses of the top-level geometry class `Geom`. Table 3.1 lists these components by category: surface, body, or neither.

Table 3.1: VSP components by category.

Surface		Body		Uncategorized
MS_WING	PROP*	POD	EXT_STORE*	BLANK
DUCT	HWB	FUSE	ENGINE*	CABIN_LAYOUT
		HAVOC	FUSE2	

* Composites of more than one `Xsec_surf`

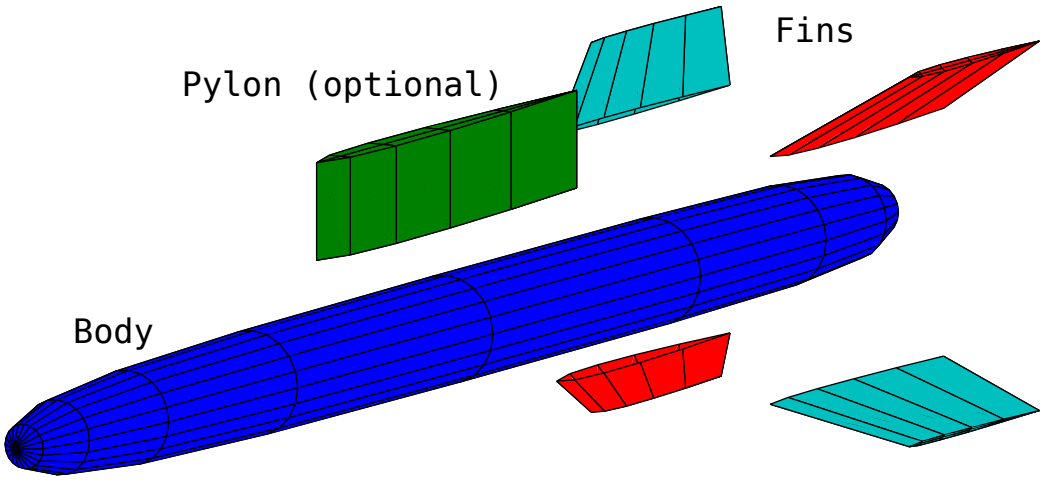

Figure 3.16: VSP external storage component showing optional pylon and two fin surfaces.

The two uncategorized components, `BLANK` and `CABIN_LAYOUT` are primarily used for model design and layout and are not strictly part of the actual vehicle. Any `CABIN_LAYOUT` component is ignored when calculating degenerate geometry. `BLANK` components however, can be designated as point masses in VSP. Location and mass information is recorded and output for any that are point masses and all others are ignored.

`MS_WING` is the prototypical surface type component and `DUCT` and `HWB` are essentially just modified versions of it. Each of these components has its geometry held in a single instance of `Xsec_surf` and all have their entire geometry taken into account when computing degenerate geometry.

`POD`, `FUSE`, `HAVOC`, and `FUSE2` are much the same: each store their geometry in one instance of `Xsec_surf`, and that entire geometry gets written out in the various degenerate forms. The starred components in table 3.1 require a little special attention.

`PROP` is composed of multiple instances of `Xsec_surf`, one for each blade. As these are all identical, only one degenerate geometry, corresponding to one blade is computed and output. However, rotation center, a vector denoting axis of rotation, and the number of blades is also output, meaning the original propeller could be reconstructed if need be.

`EXT_STORE`, shown in figure 3.16, is composed of four `Xsec_surfs`, one each for the body (blue), pylon (green), and two fins (red and cyan). There is an option to turn on/off the pylons and if the tank type is selected there are no fins. In any case, the most relevant portion is the body section, and only that is converted to degenerate geometry, the other parts are ignored.

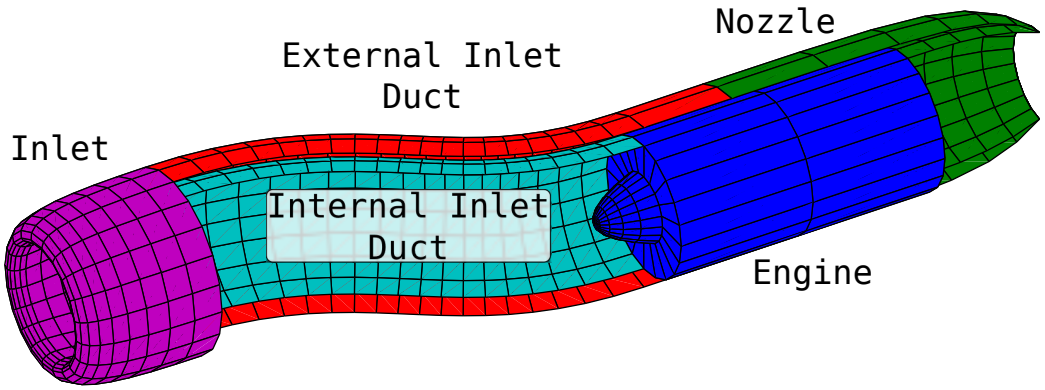

Figure 3.17: Five surfaces which comprise the VSP engine component.

`ENGINE`, shown in figure 3.17, is composed of five `Xsec_surfs`: an inlet (magenta), two portions of an inlet duct (red and cyan), a nozzle (green), and the main engine itself (blue). It was decided early on that the engine would not be supported by degenerate geometry routines for the following reasons:

1. `ENGINE` is not suitable for simple aerodynamic or structural analysis.
2. Degenerate surface assumes a certain discretization method that `ENGINE` may not comply with.
3. Effort is currently underway to replace `ENGINE` with an updated and more capable version [28].

Toward the first point, inexpensive aerodynamic analysis on an engine might simply look at its contribution to drag. For that, a simple `FUSE2` can be used. Also, though engines are structural components, they are not typically part of the aircraft's structural load paths. Toward the second point, the ducts would likely be classified as surface components, while the engine would be a body, leaving the classification of `ENGINE` undefined. The portion that is computed and output by the degenerate geometry routines is the engine section, which comprises the main core, and will suffice for a simple drag buildup.

3.3 Output File Types

VSP has the capability to write out degenerate geometry information in two file formats, a comma-separated value file and a Matlab script. The purpose behind the csv file was

to create a collection of degenerate geometry information that was easily parsable in any number of languages or could be loaded into Excel and manually manipulated.

The m-file was created primarily as a means of writing degenerate geometry information directly into Matlab variables, skipping the need for parsing. Since Matlab is often used for engineering analysis, this potentially puts the necessary information into the analysis program directly. The m-file also holds enormous benefits for design students, as their analysis tools very likely center around Matlab. The following sections describe these two formats.

3.3.1 CSV File

The csv file has a keyword-driven format and is an easily human-readable, ready-made Excel file, complete with comments. The symbol # denotes a comment line, and is followed by a comma-separated list of descriptive column headings (i.e. what comes below).

The file has two header lines, a title and a blank line, followed by a comment line stating that the next line will be the number of components contained within the file.

```
# DEGENERATE GEOMETRY CSV FILE     |     title depicting file contents
                                   |     blank line
# NUMBER OF COMPONENTS             |     comment line
42                                 |     number of components
...
```

The title is mainly to identify the file to any human reading it, say in Excel. The number of components (which does not include point masses) could be useful for storage preallocation during parsing.

If any `BLANK` components in VSP have been designated as point masses, they are listed immediately following the header lines and are designated by the keyword `BLANK_GEOMS`, followed by the number of point masses, a comment line, and then one line for each containing its name, location, and mass:

```
BLANK_GEOMS, 3
# Name, xLoc, yLoc, zLoc, Mass
Blank_0, 2.50, 3.25, 0.00, 155
...
```

These lines (if they exist) are followed by degenerate geometry information for each component. The keyword `BODY` or `SURFACE` denotes the beginning of a component, and is

followed by the component name. If the component is a propeller, the next two lines are a comment line and a line beginning with the keyword PROP followed by propeller-specific information:

```
SURFACE, Prop_0
# Propeller, Num Blades, xLoc, yLoc, zLoc, nRotX, nRotY, nRotZ
       PROP,          4,    5,    5,    3,     0,     1,     0
...
```

where the location given is the center of rotation, and the vector (nRotX, nRotY, nRotZ) is a normalized vector denoting the axis of rotation.

If the component is not a propeller, the next line following the BODY or SURFACE keyword is a comment line, followed by the type of degenerate geometry and the number of cross-sections for all types except degenerate point. For degenerate surface and degenerate plate, the number of points per cross-section is also included. The order is not crucial, but each component's degenerate geometry is listed in order of degenerate surface, degenerate plate, degenerate stick, and degenerate point. These are specified by the keywords FULL_SURFACE, PLATE, STICK, and POINT, respectively. The first two, degenerate surface and degenerate plate have the following structure:

```
BODY (or SURFACE), Fuselage
#DegenGeom Type, nXsecs, nPnts/Xsec
FULL_SURFACE, 24, 41
#x, y, z, xn, yn, zn, u, w
 0, 1, 3, 0.5, 0.2, 0.3, 0, 1
...
#DegenGeom Type, nXsecs, nPnts/Xsec
PLATE, 48, 21
#xn, yn, zn
  0,  1,  0
...
#x, y, z, zCamb, t, nCambX, nCambY, nCambZ, u, wTop, wBot
 0, 1, 3, 0, 1, .5, 2.3, 0, 1, 0, 0.35, 0.45, 0.55
 ...
```

The first line states that the component is a body type named 'Fuselage'. The comment line serves as headers for the information directly beneath. FULL_SURFACE denotes that the following is a degenerate surface, and the numbers indicate that there are 24 cross-sections composed of 41 points each. This means that there will be $24 \times 41 = 984$ lines of information following (excluding any comment lines). Each line thereafter has a node coordinate location (x,y,z), surface normal vector coordinates (xn,yn,zn) corresponding to that node (could

be NANs) and the VSP parametric coordinates u and w.

After all 984 nodes have been reported, there is an additional comment line followed by the keyword PLATE, the number of cross-sections, and the number of points per cross-section. Since there are 48 cross-sections, the next 48 lines will be plate normal vector components for each of those cross-sections (xn,yn,zn). After the plate normals (and another comment line for the humans), there will be $48 \times 21 = 1008$ lines with the degenerate plate node coordinates (x,y,z), the distance to the camber line from those nodes (zCamb), the thickness (t), direction to camber line from node (nCambX,nCambY,nCambZ), and VSP parametric coordinates (u,wTop,wBot).

Degenerate stick has too many columns to effectively show in the present format, and so its structure will be explained instead.

```
#DegenGeom Type, nXsecs
STICK, 48
# xle, yle, zle, xte, yte, zte, xcgSolid, ycgSolid, ...
  0.6, 0.0, 1.0, 0.6, 0.0, 5.0,       0.6,       0.0, ...
```

The first line contains the keyword STICK and the number of cross-sections, which will be the number of lines following. Next is a comment line that serves as column headings for all information to follow. The fields on each line are, in order: leading edge node location (xle,yle,zle), trailing edge node location (xte,yte,zte), center of gravity location for a solid (xcgSolid,ycgSolid,zcgSolid), center of gravity location for a shell (xcgShell,ycgShell,zcgShell), maximum thickness to chord (toc), location of maximum thickness in percent chord (tLoc), chord length (chord), sweep angle (sweep, can be NAN), shell moment of inertia coefficients (A1,A2,A3,A4,A5,A6), solid moments of inertia (Ixx,Izz,Iyy for surfaces or Iyy,Izz,Ixx for bodies), area, area normal vector components (areaNormalX, areaNormalY, areaNormalZ), top and bottom perimeters, and finally the VSP parametric coordinate u.

The final degenerate geometry type, degenerate point, is denoted by the keyword POINT and is only one line. The fields, in order are volume (vol), wetted volume (volWet), area (area), wetted area (areaWet), shell mass moments of inertia followed by solid mass moments of inertia (Ixx,Iyy,Izz,Ixy,Ixz,Iyz), shell center of gravity location followed by solid center of gravity location (xcg,ycg,zcg).

After degenerate point, either another component will begin with the keyword BODY or SURFACE or the end of file will be reached. There is no keyword signifying the end of file.

3.3.2 M File

The m-file output is a script that loads all degenerate geometry information into a Matlab structure. The resulting variable, called `degenGeom` is a $1 \times n$ struct of structs, where each field corresponds to a component of the original geometry (fuselage, etc.). Note that when a component has reflected symmetry, it is split into two components for output. For instance a component named `Wing` with reflected symmetry (presumably x-z) will be output in two different structures with names `Wing` and `Wing_refl`.

Figure 3.18 shows the composition of the struct `degenGeom` and its members. Each member contains a string named '`type`' which is either '`SURFACE`' or '`BODY`', and a string named '`name`' which contains the name of the original component. Each one also contains four structs, corresponding to the four degenerate geometry types.

Both `degenGeom.surf` and `degenGeom.plate` contain a struct of structs called `sect` which corresponds to the original component cross-sections. Referring to figure 3.18 and assuming `'Fuselage'` is the tenth component in `degenGeom`, to access the fourth surface normal vector from the second cross-section one would execute the following Matlab command:

```
degenGeom(10).surf.sect(2).Xn(4,:)
```

Indexing into the other variables is accomplished in a similar manner. It should be pointed out that any variables in figure 3.18 with sizes like `<2px3>` or `<1x2r>` are that size because the example component is a `BODY` (meaning that two plates and two sticks are constructed). The same variables would respectively be `<px3>` and `<1xr>` for `SURFACE` components.

One additional area that deserves clarification is that of the inertias in `degenGeom.stick` and `degenGeom.point`. Both shell and solid moments of inertia given in `degenGeom.point` are listed in the order I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, and I_{yz}. As previously mentioned, these are mass moments of inertia *per density* or *per surface density* for solid and shell, respectively. To obtain the mass moment of inertia for a solid about the x-axis for the ith component, one would execute the Matlab command

```
degenGeom(i).point.Isolid(1)*rho
```

where `rho` is the user-supplied component density.

Shell inertias in `degenGeom.stick` are the coefficients `A1-A6` given in equations (3.57) to (3.62) and (3.66). Referring once again to figure 3.18, the Matlab command

```
degenGeom(i).stick.Ishell(3,1)*t^3 + degenGeom(i).stick.Ishell(3,2)*t
```

where `t` is the user-supplied shell thickness, would return I_{xx} or I_{yy} for the third cross-section of the ith component, depending on if it was a SURFACE or BODY.

Much in the same manner each row of `degenGeom.stick.Isolid` contains the area moments of inertia I_{xx}, I_{zz}, I_{yy} for surfaces and I_{yy}, I_{zz}, I_{xx} for bodies. To get the area moment of inertia for the same section above if it were instead a solid, one would execute

```
degenGeom(i).stick.Isolid(3,1)
```

In addition to `degenGeom`, two other structs—`blankGeom` and `propGeom`—are created when there are point masses or propellers present. Point masses are collected in `blankGeom` which is a struct of structs, each corresponding to a point mass. Similarly, `propGeom` is a struct of structs corresponding to each PROP component.

Figure 3.19 shows the member fields of each of these two structs. The point mass information is simply name, location, and mass. The fields `propGeom.rotCenter` and `propGeom.rotVec` correspond to the propeller's rotation center location and a vector about which it rotates. The field `propGeom.idx` is the index in `degenGeom` corresponding to the propeller information given. For instance if `propGeom(4).idx = 17`, then the component whose degenerate geometric information is contained in `degenGeom(17)` is a propeller with the rotation center, etc. specified in `propGeom(4)`.

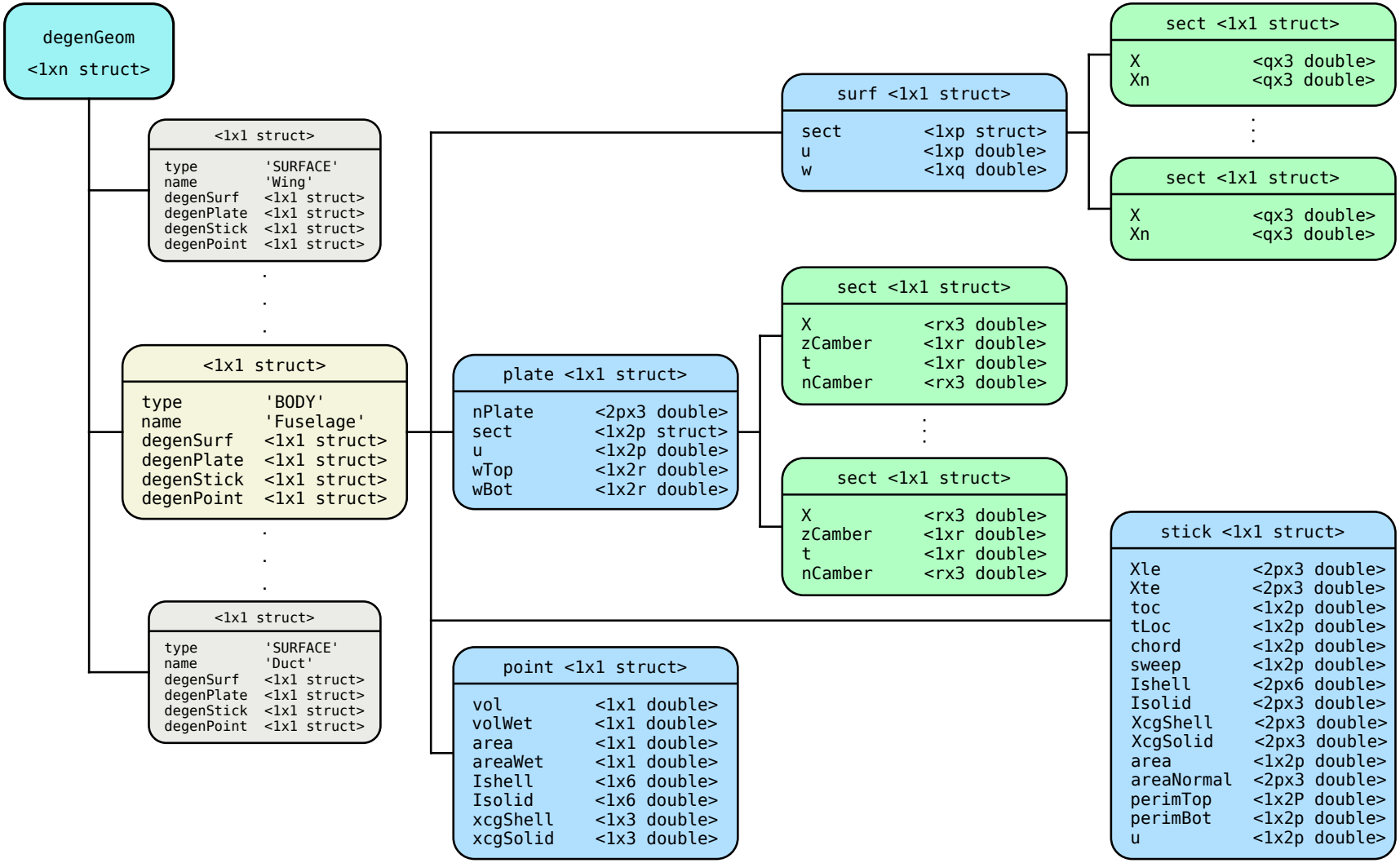

Figure 3.18: Matlab structure format for degenerate geometry.

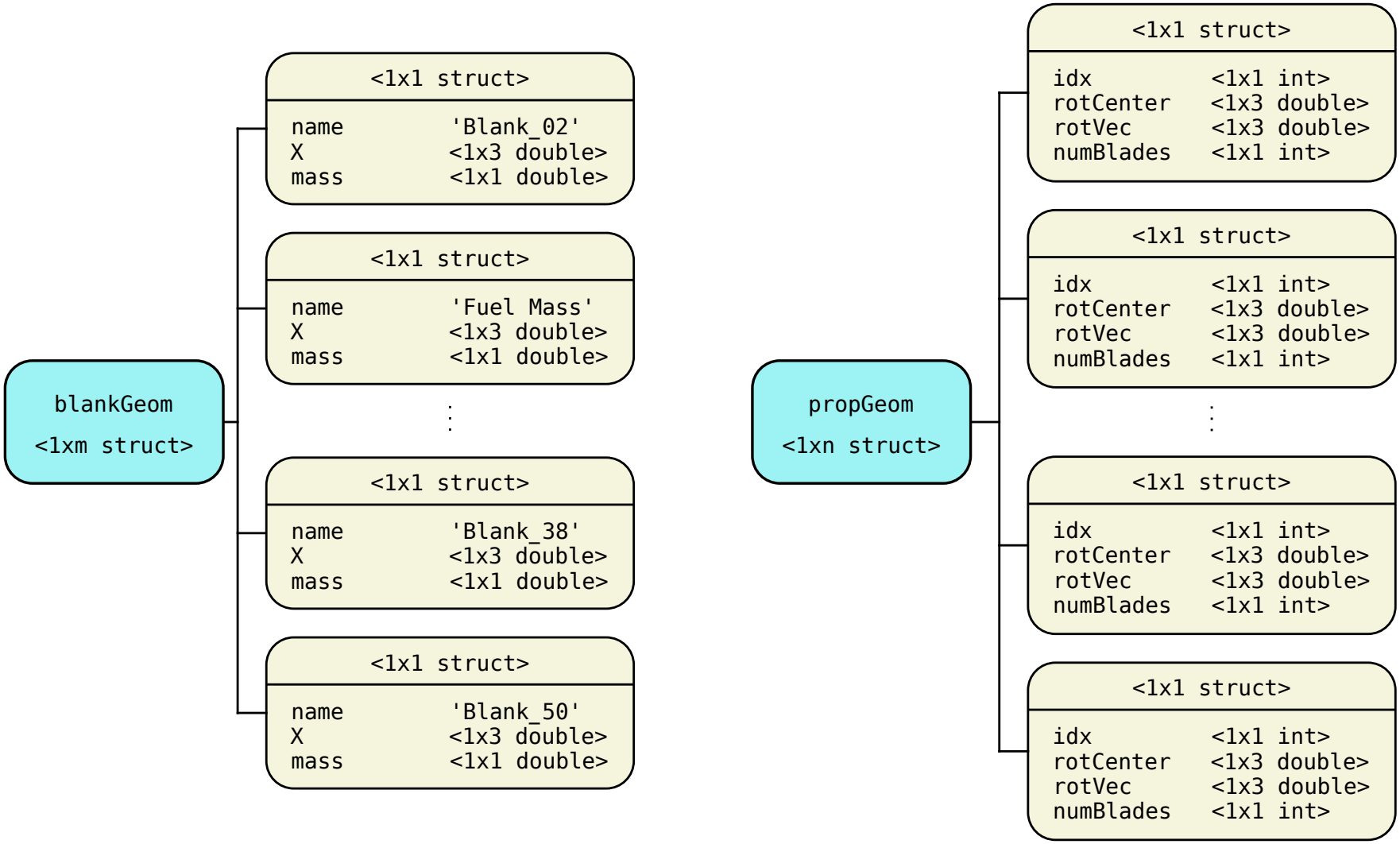

Figure 3.19: Format of additional Matlab structs describing propeller and point mass properties.

Chapter 4

Demonstration Cases, Results, and Conclusions

Having defined four types of degenerate geometry, the focus must now shift to demonstration of their usefulness. This final chapter will show that the information contained in the four degenerate types is sufficient to interface with the four target analysis techniques. Since these techniques are representative of the types of analysis in conceptual design, the demonstration cases should also inductively show that degenerate geometry is suitable for a wide array of reduced-order techniques.

The chapter will conclude with a summary of the work completed in defining degenerate geometry and implementing it in VSP. Lessons learned from the demonstration cases will also be presented, with the hope that future work can build in a meaningful way on what has been presented.

4.1 Demonstration Cases

In order to demonstrate the utility and suitability of degenerate geometry for multi-fidelity analysis, four demonstration cases were performed using a simplified Cessna 182 wing. Each corresponds to one of the target analysis techniques described in section 1.3. The purpose is to verify the suitability of degenerate geometry to interface with these four, and by extension with a wide array of lower-order techniques.

It should be noted that while the four degenerate geometry types were created with these

four analysis techniques in mind, none are intended as direct input to any particular program or formula. In fact, none were defined to necessarily provide all the information needed for their respective target analysis, but rather as a geometric representation on the same order. It is hoped that together these degenerate geometries provide a general set of information that allows interface with any number of techniques and analysis tools, providing maximum flexibility to aircraft designers.

Also of importance is the fact that no attempt was made to validate the results of these test cases or the techniques themselves. Ample literature exists for the latter and the former is beyond the point of the current investigation. In short, these test cases seek to demonstrate the usefulness of degenerate geometry in interfacing with analysis tools, not the usefulness of analysis tools in predicting design characteristics.

The previous statements notwithstanding, the analysis techniques used here have long track records of providing good results and their shortcomings are well understood. All test cases performed should yield reasonable, if not entirely accurate predictions, and comparison of outcomes between analysis techniques that utilize different sets of degenerate geometric information can therefore be used to show veracity among these different geometries. For this purpose, the aerodynamic test cases begin with AVL and structures with ELAPS, two commercial-grade analysis programs whose accuracy has been vetted, and conclude with lifting line and equivalent beam theory techniques for which there are no standard, off-the-shelf codes. Similar results between AVL and lifting line theory, and ELAPS and equivalent beam analysis will provide further support for the degenerate geometry creation process and the information contained in the various degenerate types.

4.1.1 Vortex Lattice: AVL

For the first test case, Athena Vortex Lattice was used to analyze a simplified version of the wing from the Cessna 182 example model that VSP ships with. The number of cross-sections was reduced to simplify the model and to enable global discretization options in AVL. Specifically, the model was reduced to the two center sections, eliminating the "bunched-up" cross-sections at the wing root and tip.

Translation from degenerate geometry to AVL input file was accomplished using a very brief Matlab script, which is provided in the appendix. AVL collapses geometry to a plane internally, so degenerate plate was not used in creating this file. If a fuselage is omitted (as

it is in this test case), the AVL user guide recommends connecting the two wings together instead of leaving a gap in the center. Treating both wings as a single component has this effect since AVL assumes all cross-sections supplied within any given component are connected.

Degenerate stick supplied a reference span, cross-section leading edge locations, and chord lengths. Degenerate stick was also used to define a reference chord, which was a simple mean of all sectional chord lengths. Degenerate point provided a global center of gravity, and the airfoil definitions were created using the nodes in degenerate surface.

Figure 4.1 shows the AVL geometry plot (magenta) with trailing vortex filaments and force plots along with the original geometry from VSP. The grey portions are those defined in VSP, while the center panel is "created" in AVL by simply treating the two wings as one component (supplying all cross-sections in order and letting AVL connect them together). It should be pointed out that the "plate" from AVL is, at the very least, qualitatively similar to the original component, and since the airfoil sections were defined by degenerate surface nodes, their representation in AVL carries the same degree of fidelity as the original VSP model.

The single test case performed was rather bland compared to AVL's full capabilities, but sufficient nonetheless. More importantly, it provided a set of conditions amenable to lifting line theory's capabilities. Analysis was performed at zero degrees angle of attack, no sideslip, and zero Mach (incompressible flow) with no angular rates. Figure 4.2 shows the Trefftz plane plot from this test. Without attempting to validate the results, the remark can be made that the lift and induced angle of attack distributions are reasonable across the span. The maximum sectional c_l value is at the center of the wing and is about 0.65. Lift drops off toward the wing tips, which is partially a function of larger negative induced angle of attack and partially because the original geometry has about a 4° washout.

Other quantities of interest (for comparison with lifting line theory) are C_L and $C_{D,i}$ which are 0.5128 and 0.0117, respectively. Finally, it is a safe observation at this point that all information necessary to evaluate a wing in AVL is readily obtained from degenerate geometry.

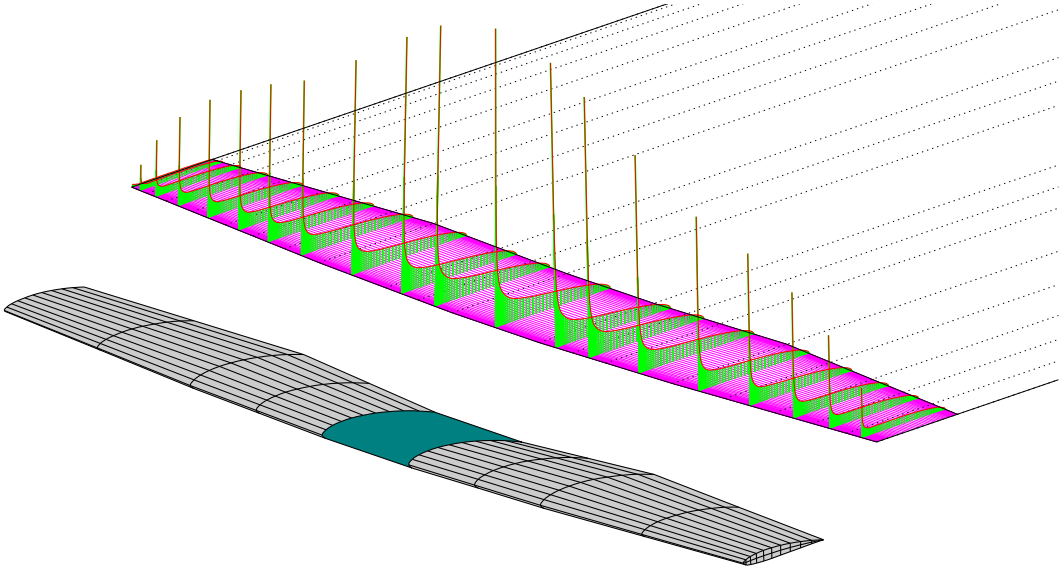

Figure 4.1: Geometry plots of Cessna wing from both VSP and AVL.

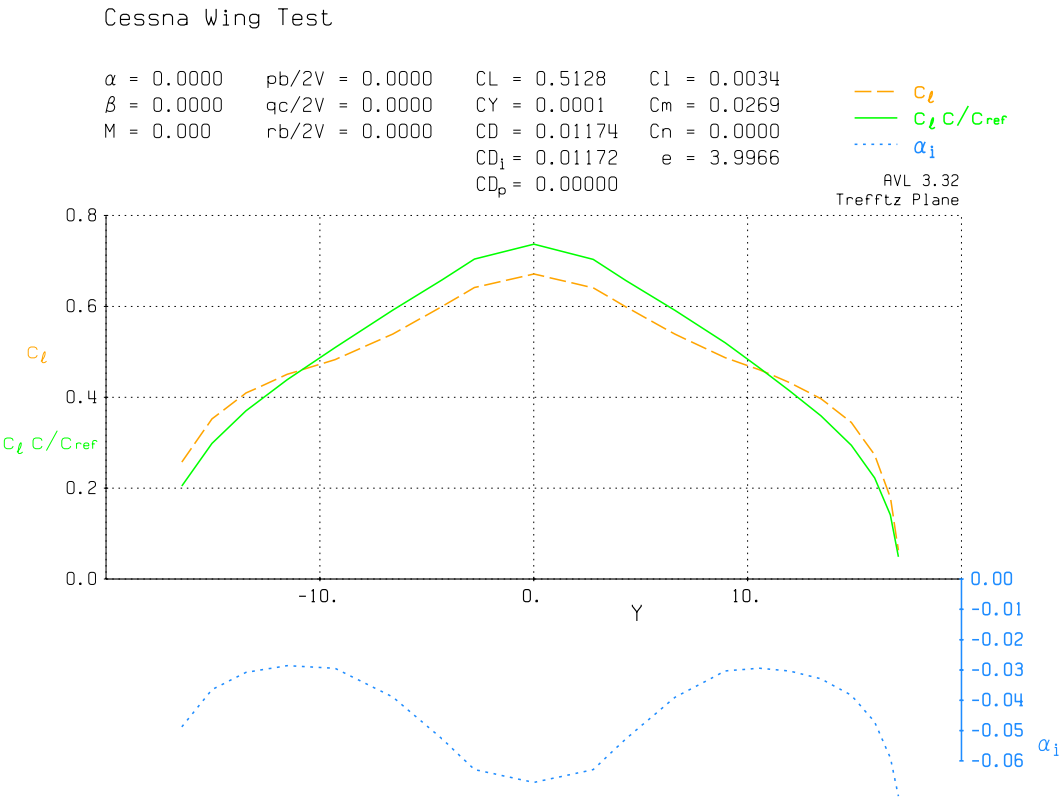

Figure 4.2: Trefftz Plane plot of Cessna wing force coefficients from AVL.

4.1.2 Lifting Line Theory

To solve for lift, induced drag, and induced angle of attack on a finite wing, classical lifting line theory needs total wing span, and geometric and zero-lift angles of attack along with chord at each station for which properties are to be computed. All of this information is contained in degenerate stick except the zero-lift angle of attack, which though a result of geometry is actually an aerodynamic property and is obtained through external analysis tools like XFoil [29].

In keeping with classical thin airfoil theory, a lift curve slope of 2π is assumed, though this can be modified if sectional properties are known. Perhaps future versions of VSP will have the capability to efficiently calculate these sectional airfoil properties, but for now zero-lift angle of attack is considered a quantity for input by the analyst, and sectional lift curve slopes are assumed to follow thin airfoil theory.

Figure 4.3 shows the modified Cessna wing used for this test case. Like the one used in AVL the grey portions are the original component from VSP. For lifting line theory the wing is assumed continuous, as shown by the center panel. Since there is no standard lifting line code in popular use, one was created in Matlab following the methods outlined in [6], with zero-lift angles of attack obtained from [27].

Figure 4.4 shows the results of analysis at zero degrees angle of attack, presented in a similar manner to AVL's Trefftz Plane plot. The sectional C_l distribution is qualitatively very similar. It has a maximum value of about 0.65 at mid-span, and falls off toward the wing tips with a flattening out near $y = \pm 10$ ft. The 3D lift coefficient is 0.5005 and the induced drag coefficient 0.0115, which differ from those predicted by AVL by 2.4% and 1.7%, respectively. Table 4.1 summarizes key results obtained by AVL and lifting line theory and notes their differences.

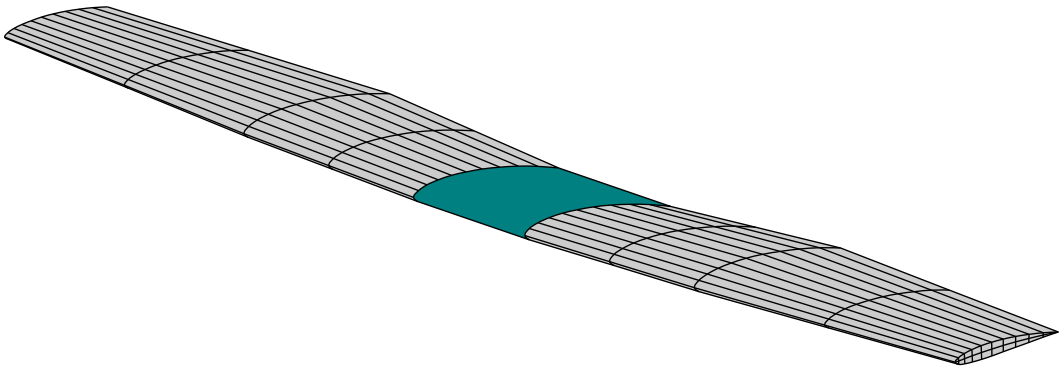

Figure 4.3: Modified Cessna wing used in lifting line theory analysis.

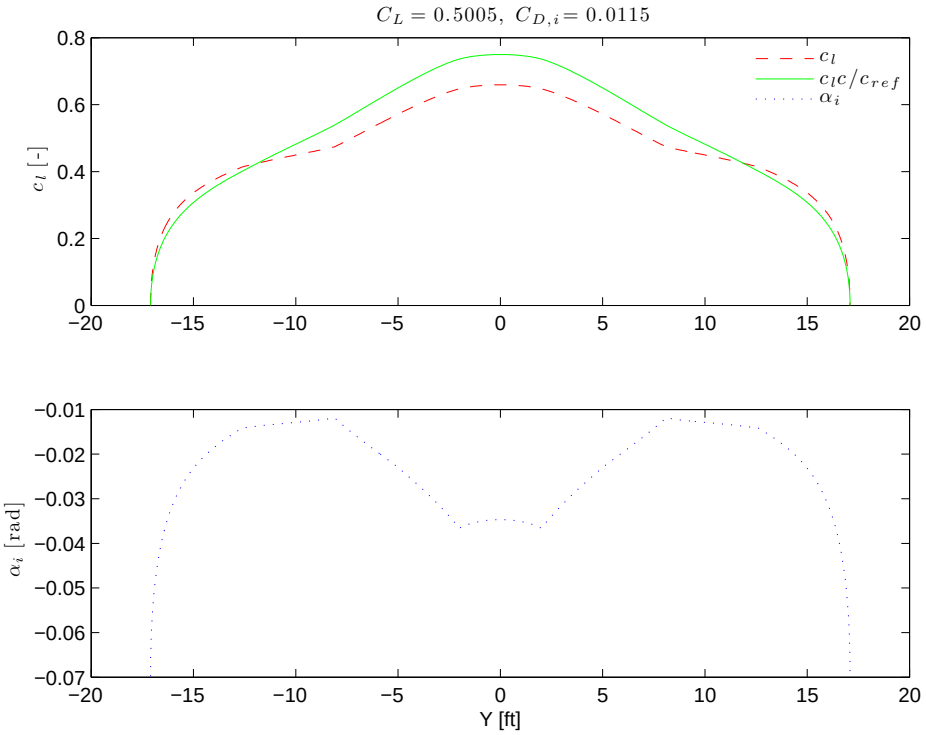

Figure 4.4: Aerodynamic properties of a modified Cessna wing computed with lifting line theory.

Table 4.1: Comparison of key quantities predicted by AVL and lifting line theory.

	C_L	$c_{l,center}$	$C_{D,i}$	$\alpha_{i,tip}$	$\alpha_{i,center}$
AVL	0.5128	0.65	0.0117	-0.08	-0.07
Lifting Line Theory	0.5005	0.65	0.0115	-0.08	-0.035
Percent Difference	2.4%	0%	1.7%	0%	50%

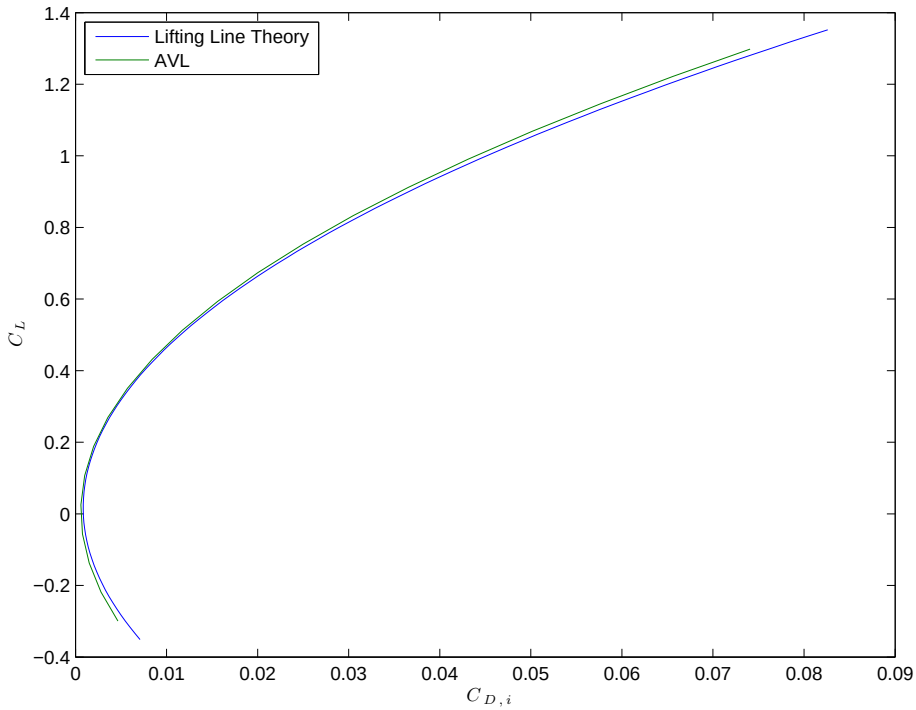

Figure 4.5: Comparison AVL and lifting line theory drag polars.

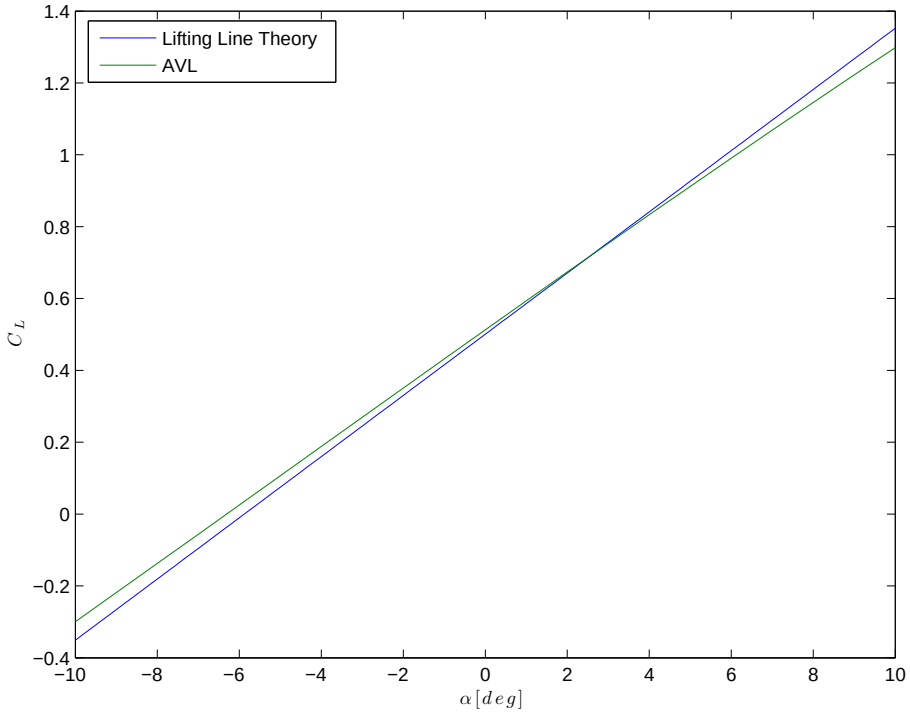

Figure 4.6: Comparison AVL and lifting line theory lift curves.

Another interesting comparison between AVL and lifting line theory is their respective drag polars and lift curves. These are shown in figures 4.5 and 4.6. As with the first case, the two methods are in excellent agreement considering that classical lifting line theory does not take the wing's dihedral into account. The only discrepancies seem to be induced angle of attack near the centerline and the difference in α_i around $y = \pm 10$ feet for the $\alpha = 0$ case. Also the AVL lift curve slope is slightly less than lifting line theory's 2π.

In either case this still provides a nice demonstration of the accuracy between different degenerate types, and verifies that all necessary information for lifting line theory analysis is readily obtained from degenerate stick.

4.1.3 Equivalent Plate: ELAPS

Using the same simplified Cessna wing geometry (right wing only), a test case was performed in ELAPS. Once again a simple Matlab script, also provided in the appendix, was written to translate degenerate geometry into a suitable input file. ELAPS requires node locations defining a plate in global coordinates, with a depth (thickness) and distance to camber line specified at each node. It also requires Young's modulus (E) in both the spanwise and chordwise directions, Poisson's ratio (ν), shear modulus (G), and material density (ρ).

All geometric information was available via degenerate plate. The material properties, which are design variables that should be input by the analyst, were taken from the first calibration region in [22]. These values correspond to a Vascomax T200© (steel) wind tunnel model wing and so are deemed reasonable, albeit somewhat arbitrary, choices. A point load (P) of 1550 pounds was applied at the wingtip mid-chord.

Figure 4.7 shows the original VSP geometry and the degenerate plate output along with the ELAPS nodes at which displacements were reported. ELAPS defines the node locations at which displacements are reported based on user-supplied options. For this test case, 5 grid locations were chosen for each plate in the spanwise and chordwise directions. The locations are interpolated using a polynomial fit of user-defined order between supplied plate nodes.

There were 5 defined cross-sections from VSP, corresponding to 4 plate regions in ELAPS. The same material properties were used for each plate region and are summarized in table 4.2. Note that both the grid discretization and material properties can vary among plate regions, but were chosen to be identical for simplicity.

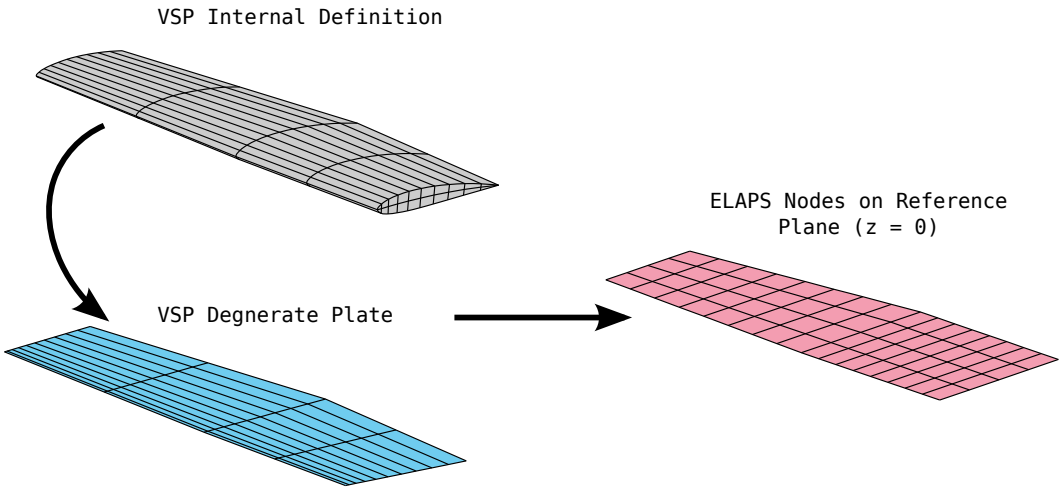

Figure 4.7: Geometry plots of Cessna wing and plate representation from VSP and plate in ELAPS.

Table 4.2: Material properties used for ELAPS test case.

$E_{chord}\ [ksi]$	$E_{span}\ [ksi]$	$\nu\ [-]$	$G\ [ksi]$	$\rho\ \left[\frac{lb_m}{in^3}\right]$
3169	3169	0.32	1200	0.288

ELAPS computes component properties like volume and center gravity location and reports them back to the user along with any deflections. Since these properties are also computed in degenerate point, an interesting comparison can be made between those properties computed solely from degenerate plate, and those that were computed from the full surface representation. Table 4.3 gives a comparison of select component properties as computed by ELAPS and contained in degenerate point. Given the plate representation, ELAPS comes up with almost identical values for volume and center of gravity location. Only the center of gravity z-location deviates appreciably when compared to other properties.

Figure 4.8 shows the ELAPS-computed wing deflection, exaggerated to show detail, with the undeflected wing shown in grey for reference. The maximum deflection was at the wing tip, as expected, and was about 1.55 inches. The reader should be reminded that no

Table 4.3: Comparison of ELAPS-calculated component properties with degenerate point.

	ELAPS	DegenPoint	% Difference
Volume	27.841	27.370	1.723
x_{cg}	4.167	4.153	0.334
y_{cg}	8.529	8.532	0.032
z_{cg}	1.956	2.106	7.109

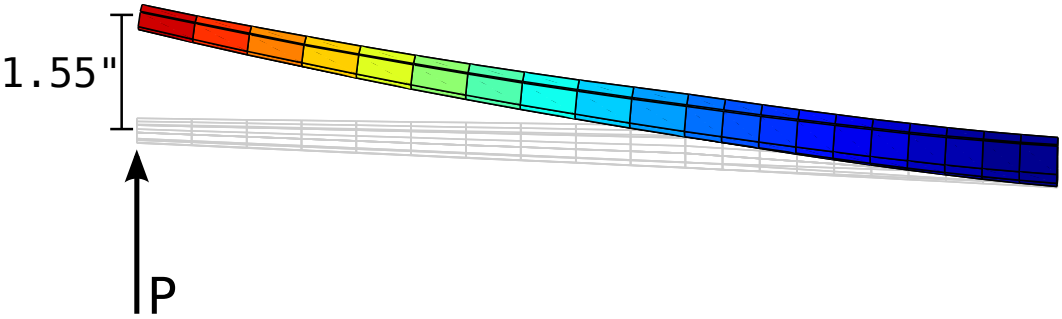

Figure 4.8: Wing deflection computed by ELAPS for end-loaded wing

attempt was made to validate either ELAPS or the given solution. The conclusion can still be drawn, however that degenerate geometry has sufficient information to interface with a respected equivalent plate structural code. In this case degenerate plate supplied the entirety of needed information, save material properties, which are of course design parameters and not intrinsic to the geometry.

4.1.4 Equivalent Beam Theory

The final demonstration case was an equivalent beam analysis. In the absence of a *de facto* equivalent beam theory standard, a simple, static cantilevered beam bending analysis *a la* mechanics of materials was performed.

Figure 4.9 shows the original wing from VSP along with its stick representation. The same modified Cessna wing was used, but for the sake of simplicity any dihedral was removed. The leading edge node locations reported by degenerate stick were used to construct the beam. Though sectional centers of gravity could have been used (and may have provided a more accurate representation), leading edge nodes were chosen since without dihedral or sweep, they define a truly one-dimensional line for this wing.

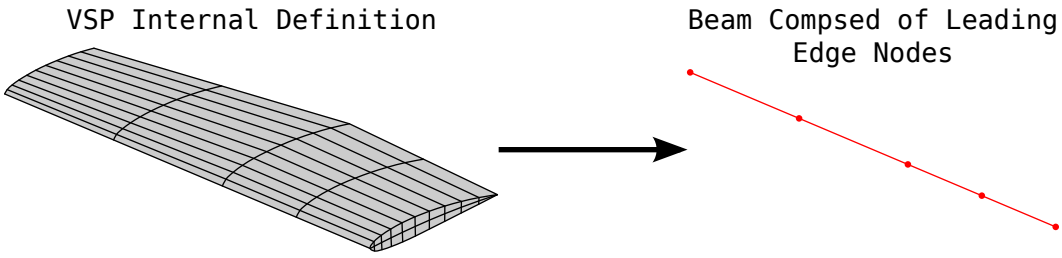

Figure 4.9: Plot of wing geometry in VSP and the beam representation composed of leading edge nodes.

In addition to the leading edge nodes, sectional area moments of inertia were also obtained from degenerate stick. The node locations and moments of inertia were linearly interpolated, and the resulting beam was subjected to a point load, P on the free end (wing tip). Analysis followed Euler-Bernoulli beam theory, which if the beam lies along the y-axis is given by

$$\frac{d^2}{dy^2}\left(EI\frac{d^2w}{dy^2}\right) = 0 \tag{4.1}$$

where w is the deflection, E is Young's modulus, and I is the area moment of inertia (in this case a function of y). The boundary conditions for this type of problem are

$$w|_{y=0} = 0 \qquad \left.\frac{dw}{dy}\right|_{y=0} = 0$$

$$\left.\frac{d^2w}{dy^2}\right|_{y=L} = 0 \qquad \left.-EI\frac{d^3w}{dy^3}\right|_{y=L} = P$$

where L is the length of the beam. It can be shown that the deflection at any point y_0 on the beam is given by

$$\delta(y_0) = \int_0^{y_0} \frac{Mm}{EI}dy \tag{4.2}$$

The elastic modulus, E is supplied by the analyst along with the moment M via the end load. The moment is given as either $M = Py$ or $M = P(L-y)$ depending on the direction of integration (free end to fixed or vice versa) and the unit load m is just M/P. The only remaining items are the physical dimensions of the beam and the area moment of inertia, both of which may be obtained from degenerate stick.

For analysis, the same value of Young's modulus given in table 4.2 was used, and the same point load of 1550 lb. was applied at the tip. Figure 4.10 shows the deflected beam, again exaggerated for detail, as well as its undeflected state in grey. The maximum deflection was at the free end (wing tip) and was about 2.62 inches. This represents roughly a 41% difference from the ELAPS solution, but is on the same order. The difference is almost certainly due to the simplicity of the 1D beam bending model compared to ELAPS' sophistication. Note also that the geometry between the two cases was slightly different.

Regardless of differences between the two methods, degenerate geometry has been shown to supply information sufficient to this type of analysis, and indeed with the preceding three methods. Together, these four demonstration cases show that degenerate geometry is a powerful tool for translating conceptual design geometry to analysis methods.

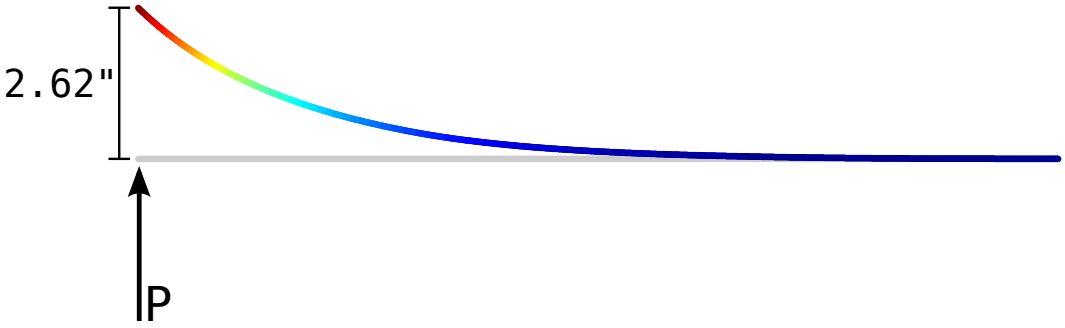

Figure 4.10: Deflection on end-loaded wing treated as simple beam.

4.2 Conclusions

The preceding work set out with the goal of helping to bridge the gap between conceptual design and analysis, to define and implement a method of distilling arbitrary, three-dimensional geometry into its essential characteristics. The low-level aim was simply enabling design analysis without the arduous, manual tinkering that has heretofore been necessary. The broader cause was to begin a process of breaking down the barriers to entry in conceptual design and analysis, to partially mitigate the risk of attempting novel ideas by coupling geometry modeling with inexpensive analysis techniques.

Four different levels of geometry distillation were defined, and four degenerate geometries created from them. They span from three- to zero-dimensional representations of underlying geometry. Two—degenerate plate and degenerate stick—were created with vortex lattice and equivalent plate, and lifting line and equivalent beam theories in mind. The other two—degenerate surface and degenerate point— were added to complete the dimensional spectrum and ensure all needed information was captured.

The ability to create and write out all four degenerate geometry types was implemented in VSP. Write out types include a comma separated value text file, and a Matlab script. Using the Matlab script, four analysis cases, corresponding to the four target analysis types, were run to ensure that degenerate geometry truly captured the underlying geometric characteristics needed for inexpensive analysis. The results were largely positive. Different methods even came up with similar (albeit not identical) results using different types of degenerate geometry. This showed that not only are the degenerate geometry types suitable to interface with external analysis techniques, but their definitions and implementation methods produce models that are true to the underlying geometry and consistent with one another.

The one area identified for improvement was the ability to obtain aerodynamic characteristics (sectional lift curves, for example) from the VSP geometry. This would remove any manual work remaining from interfacing with lifting line theory. Even so, the methods needed to obtain this information are currently within the purview of external analysis tools, which are the targets of degenerate geometry. The work is therefore complete in its current stage and to be considered a success, from the distillation and interface standpoint.

The true test of success is not readily available though, as it may only be measured by how the use of degenerate geometry (and its inevitable successors) helps spur innovation and the participation in conceptual design by a wider array of individuals and organizations, especially design students.

It is the author's sincere desire that time will show the work presented here to be a true success, that others may build on it in a meaningful fashion, and that it will in some small way help the field to progress into the future.

Bibliography

[1] L. M. Nicolai, G. Carichner, and American Institute of Aeronautics and Astronautics., *Fundamentals of Aircraft and Airship Design. Volume I-Aircraft Design.* Reston, Va: AIAA, 2010.

[2] A. Hahn, "Vehicle Sketch Pad: A Parametric Geometry Modeler for Conceptual Aircraft Design," in *Proceedings of the 48th AIAA Aerospace Sciences Meeting*, no. January, (Orlando, FL), 2010.

[3] "OpenVSP." http://www.openvsp.org/, 2012.

[4] J. Gloudemans, P. Davis, and P. Gelhausen, "A Rapid Geometry Modeler for Conceptual Aircraft," in *Proceedings of the 34th Aerospace Sciences Meeting*, no. January, (Reno, NV), 1996.

[5] A. Ko, "The OpenVSP ModelCenter Plug-In," Presented at the First OpenVSP Workshop, California Polytechnic State University, San Luis Obispo, CA, 2012.

[6] J. D. Anderson, *Fundamentals of Aerodynamics.* New York: McGraw-Hill, 2011.

[7] W. Phillips and D. Snyder, "Modern Adaptation of Prandtl's Classic Lifting-Line Theory," *Journal of Aircraft*, vol. 37, pp. 662–670, July 2000.

[8] M. Drela and H. Youngren, "AVL 3.30 User Primer," Aug. 2010.

[9] U. Lee, "Equivalent Continuum Beam-Rod Models of Aircraft Wing Structures for Aeroelastic Analysis," in *Archive Set 536*, Meeting Paper Archive, American Institute of Aeronautics and Astronautics, Jan. 1963.

[10] G. L. Giles, *Equivalent Plate Analysis of Aircraft Wing Box Structures with General Planform Geometry.* Hampton, Va.: National Aeronautics and Space Administration, Langley Research Center, 1986.

[11] G. L. Giles, "Further Generalization of an Equivalent Plate Representation for Aircraft Structural Analysis," *Journal of Aircraft*, no. February, 1989.

[12] G. L. Giles, "Equivalent Plate Modeling for Conceptual Design of Aircraft Wing Structures," (Los Angeles, CA), 1995.

[13] G. L. Giles, "Design Oriented Structural Analysis," *NASA STI/Recon Technical Report*, 1994.

[14] G. L. Giles, "Design-Oriented Analysis of Aircraft Fuselage Structures," *Journal of Aircraft*, vol. 36, no. 1, 1999.

[15] I. Jadic and V. Constantinescu, "Lifting Line Theory for Supersonic Flow Applications," *AIAA Journal*, vol. 31, no. 6, pp. 987–994, 1993.

[16] J. Chapman and F. Shaw, "Equivalent Beam Modeling Using Numerical Reduction Techniques," *Marshall Space Flight Center Structural Dynamics and Control Interaction of Flexible Structures*, pp. 567–594, 1987.

[17] N. Nguyen and I. Tuzcu, "Flight Dynamics of Flexible Aircraft with Aeroelastic and Inertial Force Interactions," in *AIAA Atmospheric Flight Mechanics Conference*, pp. 1–23, 2009.

[18] Y. Fung, *An Introduction to the Theory of Aeroelasticity.* Phoenix Edition Series, Dover, 2002.

[19] J. Kosmatka, "Flexure-Torsion Behavior of Prismatic Beams, Part I: Section Properties via Power Series," *AIAA Journal*, vol. 31, pp. 170–179, Jan. 1993.

[20] T. Megson, *Aircraft Structures for Engineering Students.* Oxford; Burlington, MA: Butterworth-Heinemann, 2007.

[21] W. Mason, "Aerodynamics of 3D Lifting Surfaces through Vortex Lattice Methods," in *Applied Computational Aerodynamics Text/Notes*, ch. 6, Blacksburg, VA: Virginia Polytechnic Institute and State University, 1998.

[22] D. Allison and P. Cavallo, *Static Aeroelastic Predictions for a Transonic Transport Model Using an Unstructured-grid Flow Slover Coupled with a Structural Plate Technique.* No. March, 2003.

[23] C. Steger, "On the Calculation of Arbitrary Moments of Polygons," *... Munchen, Germany, Tech. Rep. FGBV-96 ...*, no. October, 1996.

[24] A. Dobrovolskis, "Inertia of Any Polyhedron," *Icarus*, vol. 124, pp. 698–704, Dec. 1996.

[25] B. Eckel, *Thinking in C++*. Upper Saddle River, N.J.: Prentice Hall, 2000.

[26] E. Freeman, E. Freeman, K. Sierra, and B. Bates, *Head First Design Patterns*. Sebastopol [etc.]: O'Reilly, 2004.

[27] I. H. Abbott and A. E. Von Doenhoff, *Theory of Wing Sections : Including a Summary of Airfoil Data*. New York: Dover Publications, 1959.

[28] R. Denney, J. Gladin, J. Tai, and D. Mavris, "Propulsion Systems Modeling with Vehicle Sketch Pad," in *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan. 2013.

[29] M. Drela and H. Youngren, "XFOIL 6.9 User Primer," 2001.

[30] D. Flanagan, *Java in a nutshell*. Beijing; Sebastopol, CA: O'Reilly, 2005.

[31] S. Ziemer and G. Stenz, "The Case for Open Source Software in Aeronautics," *Aircraft Engineering and Aerospace Technology*, vol. 84, no. 3, pp. 133–139, 2012.

[32] Y.-H. Na and S. Shin, "Equivalent-Plate Analysis for a Composite Wing with a Control Surface," *Journal of Aircraft*, pp. 1–10, Feb. 2013.

[33] J. Belben, "Enabling Rapid Conceptual Design Using Geometry-Based Multi-Fidelity Models in VSP," in *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan. 2013.

Appendix A

Matlab Scripts

A.1 Cessna Wing to AVL File Conversion

```
% Convert degenGeom struct with one wing into AVL input file %

clear all; close all; clc;

% Load info
run cessnaWing

% Create AVL file for writing
fid = fopen('cessnaWing.avl','w+');

% Case name
fprintf(fid,'Cessna Wing Test\n');

% Description Comment
fprintf(fid,'# This is a test of DegenGeom interfacing with AVL\n');

% Mach, iYsym, iZsym, Zsym
fprintf(fid,'0.0\n0  0  0\n');

% Mean chord
Cref = mean(degenGeom(1).stick.chord);

% Wing semi-span
Bref = norm( degenGeom(1).stick.XcgSolid(end,:) ...
           - degenGeom(2).stick.XcgSolid(end,:) ) / 2;

% Wing reference area
Sref = 2*Bref*Cref;

% Global CG
cg = ( degenGeom(1).point.xcgSolid + degenGeom(2).point.xcgSolid) / 2;

% Reference quantities
fprintf(fid,'%f  %f  %f\n%f  %f  %f\n',Sref,Cref,Bref,cg(1),cg(2),cg(3));
fprintf(fid,'SURFACE\nWing\n300  1.0  20  -1.5\n');
for i = [2,1]
    if i == 1
        jVec = 1:numel(degenGeom(i).plate.sect);
    else
        jVec = fliplr(1:numel(degenGeom(i).plate.sect));
    end
    for j = jVec
        fprintf(fid,'SECTION\n');
        xle   = degenGeom(i).stick.Xle(j,:);
        chord = degenGeom(i).stick.chord(j);

        fprintf(fid,'%f  %f  %f  %f  %d\n', xle(1), xle(2), xle(3), ...
                                            chord , 0 );
        afFile = [pwd '/cwAF' num2str(i) num2str(j) '.dat'];
        afid   = fopen(afFile,'w');
        pnts = degenGeom(i).surf.sect(j).X(:,[1 3]);
        pnts(:,1) = pnts(:,1) - xle(1);
        pnts      = pnts / chord;
        for k = 1:size(pnts,1)
            fprintf(afid,'%f  %f\n',pnts(k,1),pnts(k,2));
        end
        fclose(afid);
        fprintf(fid,['AFILE\n' afFile '\n'] );
    end
end
fprintf(fid,'# This is the end of the file\n');
fclose(fid);
```

A.2 Lifting Line Analysis on Cessna Wing

```
% Use degenStick from Cessna wing for lifting line code %

%% Preliminaries
clear all; close all; clc

% Load info
run cessnaWing

% Wing span, assumed aligned with y-axis
b = abs(degenGeom(1).stick.Xle(end,2) - degenGeom(2).stick.Xle(end,2));

% Vector of chord lengths, assumed these align with x-axis
c = [fliplr(degenGeom(2).stick.chord), degenGeom(1).stick.chord];

% Reference area
S = mean(c)*b;

% Aspect ratio
AR = b^2/S;

% Angles of attack at each section
delX = [flipud(degenGeom(2).stick.Xte(:,1) - degenGeom(2).stick.Xle(:,1)); ...
                degenGeom(1).stick.Xte(:,1) - degenGeom(1).stick.Xle(:,1)];
delZ = [flipud(degenGeom(2).stick.Xte(:,3) - degenGeom(2).stick.Xle(:,3)); ...
                degenGeom(1).stick.Xte(:,3) - degenGeom(1).stick.Xle(:,3)];

a    = -atan2(delZ,delX);

% Location along wing
y = [flipud(degenGeom(2).stick.Xte(:,2)); degenGeom(1).stick.Xte(:,2)];
t = real(acos(-2*y/b));

% Zero lift angles of attack
a0 = pi/180*-2*ones(size(y));

%% Chop it up!

% Number of discretized points
N = 1000;

% Interpolated y locations
tt = linspace(0+eps,pi-eps,N);

% Chord lengths
cc = interp1(t,c,tt);

% Angles of attack
aa = interp1(t,a,tt);

% Zero-lift angles of attack
aa0 = interp1(t,a0,tt);


%% Solution

% Build right hand side
rhs = ((aa - aa0).*sin(tt))';

% Build coefficient matrix
M = zeros(N);
for i = 1:N
    for j = 1:N
        M(i,j) = (2*b)/(pi*cc(i))*sin(j*tt(i))*sin(tt(i)) + j*sin(j*tt(i));
    end
```

```
end

% Solve for Fourier coefficients
A = M\rhs;

% Globl lift coefficient
CL = A(1)*pi*AR;

% Efficiency factor
del = 0;
for i = 2:N
    del = del + i*(A(i)/A(1))^2;
end
e = 1/(1+del);

% Induced drag coefficient
CDi = CL^2/(pi*e*AR);

% Sectional lift coeffcient and induced alpha
Cl   = zeros(N,1);
ClcC = zeros(N,1);
ai   = zeros(N,1);
for i = 1:N
    Q1 = 0;
    Q2 = 0;
    for j = 1:N
        Q1 = Q1 + A(j)*sin(j*tt(i));
        Q2 = Q2 + j*A(j)*sin(j*tt(i))/sin(tt(i));
    end
    Cl(i)   = 4*b/cc(i)*Q1;
    ClcC(i) = Cl(i)*cc(i)/mean(cc);
    ai(i)   = -Q2;
end

figure
subplot(2,1,1)
h(1) = plot(-b/2*cos(tt),Cl,'r--');
hold all;
h(2) = plot(-b/2*cos(tt),ClcC,'g');
ylabel('$c_l$ [-]','Interpreter','Latex');
title(['$C_L = ' num2str(CL,'%0.4f') ',\;C_{D,i} = ' num2str(CDi,'%0.4f')...
    '$'], 'Interpreter','Latex')

subplot(2,1,2)
h(3) = plot(-b/2*cos(tt),ai,'b:');
xlabel('Y [ft]');
ylabel('$\alpha_i$ [rad]','Interpreter','Latex');

subplot(2,1,1)
lh = legend(h,'$c_l$','$c_lc/c_{ref}$','$\alpha_i$');
set(lh,'Box','off','Interpreter','Latex')
```

A.3 Cessna Wing to ELAPS File Conversion

```
% Convert degenGeom struct with one wing into ELAPS input file %

clear all; close all; clc;

% Load info
run cessnaWing

% Create ELAPS file for writing
fid = fopen('cessnaWing.elaps','w+');

fprintf(fid, '''TEXT''     1\n''This is a test file for degenGeom.''\n');
fprintf(fid, '''CONTRL''   10\n   2  0  2  0\n');
fprintf(fid, '''NDFUN''    3\n   11  4  0 1 2 3  4  2 3 4 5\n');
fprintf(fid,                 '   22  4  0 1 2 3  4  1 2 3 4\n');
fprintf(fid,                 '   33  4  0 1 2 3  4  2 3 4 5\n');
fprintf(fid, '''NDSYS''    1\n   1234  11  22  33  0  0\n');

% Init empty vars to hold plate points
xP = [];    yP = [];    zP = [];

for j = 1:numel(degenGeom(1).plate.sect)
    xP = [xP;degenGeom(1).plate.sect(j).X(:,1)];
    yP = [yP;degenGeom(1).plate.sect(j).X(:,2)];
end

% Bounds of reference surface
xPmin = min(xP)-1;    xPmax = max(xP)+1;
yPmin = min(yP)-1;    yPmax = max(yP)+1;
fprintf(fid, ['   ' num2str(xPmin) '  ' num2str(yPmin) '  0.0\n']);
fprintf(fid, ['   ' num2str(xPmax) '  ' num2str(yPmax) '  0.0\n']);

% Calc number of sections on wing
nSecs = numel(degenGeom(1).plate.sect) - 1;
fprintf(fid, ['''NMATL''  ' num2str(nSecs) '\n']);

Echord = num2str(3.196E+6*144);     Espan = Echord;
G      = num2str(1.200E+6*144);     nu    = '0.32';
rho    = num2str(0.0144*12^3);

for j = 1:nSecs
    fprintf(fid,[ '   ' Echord '  ' Espan '  ' nu '   ' G '  ' rho...
                  '  0.0  0.0\n']);
end

fprintf(fid, ['''NSEGMT''   ' num2str(nSecs) '\n']);

for j = 1:nSecs
    fprintf(fid, ['''PLATE''  ' num2str(j) '\n   1234\n']);
    fprintf(fid, ['   ''PLANF''  ' num2str(j) '\n']);

    % Bounds of inboard section
    xPmin = min(degenGeom(1).plate.sect(j).X(:,1));
    yPmin = min(degenGeom(1).plate.sect(j).X(:,2));
    xPmax = max(degenGeom(1).plate.sect(j).X(:,1));
    fprintf(fid, ['      ' num2str(xPmin, '%.6f') '  ' ...
        num2str(yPmin, '%.6f') '  ' num2str(xPmax, '%.6f') '\n']);

    % Bounds of outboard section
    xPmin = min(degenGeom(1).plate.sect(j+1).X(:,1));
    yPmin = min(degenGeom(1).plate.sect(j+1).X(:,2));
    xPmax = max(degenGeom(1).plate.sect(j+1).X(:,1));
    fprintf(fid, ['      ' num2str(xPmin, '%.6f') '  ' ...
        num2str(yPmin, '%.6f') '  ' num2str(xPmax, '%.6f') '\n']);

```

```
    fprintf(fid, ['   ''DEPTH''  ' num2str(j) '\n']);
    fprintf(fid,  '   ''TABPTS''   ''XIETA''\n');
    fprintf(fid,  '   6  1\n');

    numPnts = size(degenGeom(1).plate.sect(j).X,1);
    thckString = [];
    cambString = [];
    pnts = flipud(degenGeom(1).plate.sect(j).X(:,1));
    chrd = sqrt((pnts(end) - pnts(1))^2);
    pnts = ( pnts - pnts(1,:) ) / chrd;
    thck = flipud(degenGeom(1).plate.sect(j).t');
    camb = flipud(degenGeom(1).plate.sect(j).zCamber' ...
                + degenGeom(1).plate.sect(j).X(:,3)   );
    fprintf(fid, ['   ' num2str(2*numPnts) '  ''XIETA''\n']);
    for k = 1:numPnts
        if thck(k) == 0
            thck(k) = 0.0001;
        end
        if pnts(k,1) == 0
            pnts(k,1) = 0.0;
        end
        thckString = [thckString '   ' num2str(pnts(k,1), '%.6f') ...
                      '  0.000000  '   num2str(thck(k),   '%.6f') '\n'];
        cambString = [cambString '   ' num2str(pnts(k,1), '%.6f') ...
                      '  0.000000  '   num2str(camb(k),   '%.6f') '\n'];
    end
    pnts = flipud(degenGeom(1).plate.sect(j+1).X(:,1));
    chrd = sqrt((pnts(end) - pnts(1))^2);
    pnts = ( pnts - pnts(1,:) ) / chrd;
    thck = flipud(degenGeom(1).plate.sect(j+1).t');
    camb = flipud(degenGeom(1).plate.sect(j+1).zCamber' ...
                + degenGeom(1).plate.sect(j+1).X(:,3)   );
    for k = 1:numPnts
        if thck(k) == 0
            thck(k) = 0.0001;
        end
        if pnts(k,1) == 0
            pnts(k,1) = 0.0;
        end
        thckString = [thckString '   ' num2str(pnts(k,1), '%.6f') ...
                      '  1.000000  '   num2str(thck(k),   '%.6f') '\n'];
        cambString = [cambString '   ' num2str(pnts(k,1), '%.6f') ...
                      '  1.000000  '   num2str(camb(k),   '%.6f') '\n'];
    end
    fprintf(fid, thckString);
    fprintf(fid, ['   ''CAMBR''  ' num2str(j) '\n']);
    fprintf(fid,  '   ''TABPTS''   ''XIETA''\n');
    fprintf(fid,  '   6  1\n');
    fprintf(fid, ['   ' num2str(2*numPnts) '  ''XIETA''\n']);
    fprintf(fid, cambString);

    fprintf(fid, ['   ''SOLID''  ' num2str(j) '\n']);
    fprintf(fid, ['   ' num2str(j)  '  0.0\n']);
end
fprintf(fid, '''NFORC''  1\n   1  1234  1.0\n');
fprintf(fid, '   5  1\n   1\n');

xPmax = mean(degenGeom(1).plate.sect(end).X(:,1));
yPmax = max(degenGeom(1).plate.sect(end).X(:,2));

fprintf(fid, ['   3  ' num2str(xPmax) '  ' num2str(yPmax) '  0.0  1550\n']);
fprintf(fid, '''STAT''  1\n   1\n');
fprintf(fid, '''OSOLTNV''  21\n   1  1  7\n');
fprintf(fid, ['''PDISP''  1\n   1  1\n   1  ' num2str(nSecs) '\n']);
sSpan = abs(degenGeom(1).plate.sect(end).X(1,2) ...
          - degenGeom(1).plate.sect(1).X(1,2)   );
fprintf(fid, ['   6  6\n   2  ' num2str(sSpan) '\n''ENDDATA''  1\n']);
fclose(fid);
```

A.4 Beam Bending Analysis on Cessna Wing

```
% Use degenStick from Cessna wing for beam bending %

clear all; close all; clc
% Load info
run cessnaWing

% Number of dicretized points
N = 1000;

% End loading
P = 1550;

% Young's modulus
E = 3.169e6*144;

% Dimension along a 1D beam.
y = degenGeom(1).stick.Xle(:,2);

% Make root start from zero
y = y - y(1);

% Add y points for interpolation
yy = linspace(y(1),y(end),N);

% Length of beam
L = y(end);

% Inertia in lift (z) direction
I = degenGeom(1).stick.Isolid(:,1);

% Linearly interpolate inertia
II = flipud(interp1(y,I,yy));

del = cumtrapz(yy,P*yy.^2./(E*II));

figure
scatter3(zeros(N,1),yy,del,5,del,'filled');
caxis([min(min(del)) max(max(del))]);
axis equal;
ylim([-1 16]);
zlim([-1 4]);
view(-90,0);
axis off;

figure
plot3(zeros(length(y),1),y,zeros(length(y),1),'LineWidth',5);
axis equal;
ylim([-1 16]);
zlim([-1 4]);
view(-90,0);
axis off;
```

Appendix B

Target Analysis Input Files

B.1 AVL Input File

```
Cessna Wing Test
# This is a test of DegenGeom interfacing with AVL
0.0
0  0  0
163.741307  4.788814  17.096226
4.152763  0.000000  2.105484
SURFACE
Wing
300  1.0  20  -1.5
SECTION
2.390664  -17.096265  2.354235  3.700000  0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF25.dat
SECTION
2.195332  -12.598788  2.257002  4.477840  0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF24.dat
SECTION
2.000000  -8.101310  2.159768  5.256011  0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF23.dat
SECTION
2.000000  -5.050655  2.079884  5.254210  0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF22.dat
SECTION
2.000000  -2.000000  2.000000  5.256011  0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF21.dat
SECTION
2.000000  2.000000  2.000000  5.256011  0
```

```
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF11.dat
SECTION
2.000000   5.050655   2.079884   5.254210   0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF12.dat
SECTION
2.000000   8.101310   2.159768   5.256011   0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF13.dat
SECTION
2.195332   12.598788   2.257002   4.477840   0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF14.dat
SECTION
2.390664   17.096265   2.354235   3.700000   0
AFILE
/Users/joel/Dropbox/vsp/vspTest/cwAF15.dat
# This is the end of the file
```

B.2 Airfoil Files for AVL

cwAF11.dat

```
0.994522  0.276024
0.853407  0.321035
0.718201  0.356694
0.590007  0.385071
0.469738  0.406472
0.358350  0.420946
0.257050  0.428451
0.167361  0.428983
0.091391  0.422543
0.032576  0.408707
0.000000  0.380517
0.030835  0.350589
0.087877  0.332753
0.161769  0.319553
0.249304  0.309774
0.348706  0.302556
0.458847  0.297003
0.578959  0.292168
0.708447  0.287178
0.846956  0.281418
0.994522  0.276024
```

cwAF12.dat

```
0.996917  0.317419
0.854672  0.358722
0.718579  0.390831
0.589685  0.415843
0.468897  0.434090
0.357168  0.445643
0.255707  0.450496
0.166034  0.448680
0.090259  0.440255
0.031827  0.424884
0.000000  0.395851
0.031608  0.366740
0.089097  0.350403
0.163310  0.339142
0.251070  0.331656
0.350628  0.327042
0.460876  0.324373
0.581074  0.322683
0.710649  0.321083
0.849260  0.318949
0.996917  0.317419
```

cwAF13.dat

```
0.998630  0.358596
0.855352  0.396163
0.718464  0.424699
0.588960  0.446330
0.467735  0.461410
0.355742  0.470036
0.254188  0.472231
0.164594  0.468070
0.089066  0.457664
0.031055  0.440770
0.000000  0.410914
0.032360  0.382641
0.090257  0.367813
0.164739  0.358498
0.252665  0.353311
0.352309  0.351304
0.462590  0.351521
0.582791  0.352976
0.712363  0.354768
0.850983  0.356262
0.998630  0.358596
```

cwAF14.dat

```
0.998981  0.458919
0.855309  0.494076
0.718178  0.520591
0.588533  0.540481
0.467248  0.554118
0.355260  0.561633
0.253758  0.563091
0.164248  0.558607
0.088821  0.548337
0.030919  0.532117
0.000000  0.504038
0.032483  0.477999
0.090470  0.464681
0.165042  0.456582
0.253055  0.452352
0.352777  0.451058
0.463117  0.451767
0.583350  0.453520
0.712917  0.455439
0.851482  0.456897
0.998981  0.458919
```

cwAF15.dat

```
0.999391  0.601392
0.855173  0.633122
0.717707  0.656762
0.587874  0.674178
0.466514  0.685765
0.354543  0.691701
0.253125  0.692111
0.163742  0.687169
0.088466  0.677092
0.030723  0.661831
0.000000  0.636280
0.032655  0.613416
0.090765  0.602245
0.165458  0.595875
0.253586  0.593005
0.353410  0.592723
0.463825  0.594130
0.584092  0.596306
0.713641  0.598406
0.852114  0.599814
0.999391  0.601392
```

cwAF21.dat

```
0.994522  0.276024
0.853407  0.321035
0.718201  0.356694
0.590007  0.385071
0.469738  0.406472
0.358350  0.420946
0.257050  0.428451
0.167361  0.428983
0.091391  0.422543
0.032576  0.408707
0.000000  0.380517
0.030835  0.350589
0.087877  0.332753
0.161769  0.319553
0.249304  0.309774
0.348706  0.302556
0.458847  0.297003
0.578959  0.292168
0.708447  0.287178
0.846956  0.281418
0.994522  0.276024
```

cwAF22.dat

```
0.996917  0.317419
0.854672  0.358722
0.718579  0.390831
0.589685  0.415843
0.468897  0.434090
0.357168  0.445643
0.255707  0.450496
0.166034  0.448680
0.090259  0.440255
0.031827  0.424884
0.000000  0.395851
0.031608  0.366740
0.089097  0.350403
0.163310  0.339142
0.251070  0.331656
0.350628  0.327042
0.460876  0.324373
0.581074  0.322683
0.710649  0.321083
0.849260  0.318949
0.996917  0.317419
```

cwAF23.dat

```
0.998630  0.358596
0.855352  0.396163
0.718464  0.424699
0.588960  0.446330
0.467735  0.461410
0.355742  0.470036
0.254188  0.472231
0.164594  0.468070
0.089066  0.457664
0.031055  0.440770
0.000000  0.410914
0.032360  0.382641
0.090257  0.367813
0.164739  0.358498
0.252665  0.353311
0.352309  0.351304
0.462590  0.351521
0.582791  0.352976
0.712363  0.354768
0.850983  0.356262
0.998630  0.358596
```

cwAF24.dat

```
0.998981  0.458919
0.855309  0.494076
0.718178  0.520591
0.588533  0.540481
0.467248  0.554118
0.355260  0.561633
0.253758  0.563091
0.164248  0.558607
0.088821  0.548337
0.030919  0.532117
0.000000  0.504038
0.032483  0.477999
0.090470  0.464681
0.165042  0.456582
0.253055  0.452352
0.352777  0.451058
0.463117  0.451767
0.583350  0.453520
0.712917  0.455439
0.851482  0.456897
0.998981  0.458919
```

cwAF25.dat

```
0.999391  0.601392
0.855173  0.633122
0.717707  0.656762
0.587874  0.674178
0.466514  0.685765
0.354543  0.691701
0.253125  0.692111
0.163742  0.687169
0.088466  0.677092
0.030723  0.661831
0.000000  0.636280
0.032655  0.613416
0.090765  0.602245
0.165458  0.595875
0.253586  0.593005
0.353410  0.592723
0.463825  0.594130
0.584092  0.596306
0.713641  0.598406
0.852114  0.599814
0.999391  0.601392
```

B.3 ELAPS Input File

```
'TEXT'     1
'This is a test file for degenGeom.'
'CONTRL'   10
   2  0  2  0
'NDFUN'    3
   11  4  0 1 2 3  4  2 3 4 5
   22  4  0 1 2 3  4  1 2 3 4
   33  4  0 1 2 3  4  2 3 4 5
'NDSYS'    1
   1234  11  22  33  0  0
   1  1  0.0
   8.2488  18.0996  0.0
'NMATL'  4
   460224000  460224000  0.32   172800000  24.8832  0.0  0.0
   460224000  460224000  0.32   172800000  24.8832  0.0  0.0
   460224000  460224000  0.32   172800000  24.8832  0.0  0.0
   460224000  460224000  0.32   172800000  24.8832  0.0  0.0
'NSEGMT'   4
'PLATE'  1
   1234
   'PLANF'  1
      2.000000  2.000000  7.227218
      2.000000  5.050655  7.238013
   'DEPTH'  1
   'TABPTS'   'XIETA'
   6  1
   22  'XIETA'
   0.000000  0.000000  0.000100
   0.031623  0.000000  0.305710
   0.089443  0.000000  0.472462
   0.164317  0.000000  0.576111
```

```
0.252982  0.000000  0.625309
0.353553  0.000000  0.624531
0.464758  0.000000  0.578407
0.585662  0.000000  0.491903
0.715542  0.000000  0.369080
0.853815  0.000000  0.211044
1.000000  0.000000  0.000100
0.000000  1.000000  0.000100
0.031623  1.000000  0.305605
0.089443  1.000000  0.472300
0.164317  1.000000  0.575914
0.252982  1.000000  0.625095
0.353553  1.000000  0.624317
0.464758  1.000000  0.578209
0.585662  1.000000  0.491734
0.715542  1.000000  0.368954
0.853815  1.000000  0.210971
1.000000  1.000000  0.000100
'CAMBR'  1
'TABPTS'   'XIETA'
6  1
22  'XIETA'
0.000000  0.000000  2.000000
0.031623  0.000000  1.969756
0.089443  0.000000  1.916632
0.164317  0.000000  1.852016
0.252982  0.000000  1.781595
0.353553  0.000000  1.709721
0.464758  0.000000  1.640150
0.585662  0.000000  1.576311
0.715542  0.000000  1.521429
0.853815  0.000000  1.478590
```

```
   1.000000  0.000000  1.450785
   0.000000  1.000000  2.079884
   0.031623  1.000000  2.053980
   0.089443  1.000000  2.008792
   0.164317  1.000000  1.954450
   0.252982  1.000000  1.896194
   0.353553  1.000000  1.838116
   0.464758  1.000000  1.783796
   0.585662  1.000000  1.736533
   0.715542  1.000000  1.699454
   0.853815  1.000000  1.675564
   1.000000  1.000000  1.667785
   'SOLID'  1
   1  0.0
'PLATE'  2
   1234
   'PLANF'  2
      2.000000  5.050655  7.238013
      2.000000  8.101310  7.248808
   'DEPTH'  2
   'TABPTS'   'XIETA'
   6  1
   22  'XIETA'
   0.000000  0.000000  0.000100
   0.031623  0.000000  0.305605
   0.089443  0.000000  0.472300
   0.164317  0.000000  0.575914
   0.252982  0.000000  0.625095
   0.353553  0.000000  0.624317
   0.464758  0.000000  0.578209
   0.585662  0.000000  0.491734
   0.715542  0.000000  0.368954
```

```
0.853815  0.000000  0.210971
1.000000  0.000000  0.000100
0.000000  1.000000  0.000100
0.031623  1.000000  0.305710
0.089443  1.000000  0.472462
0.164317  1.000000  0.576111
0.252982  1.000000  0.625309
0.353553  1.000000  0.624531
0.464758  1.000000  0.578407
0.585662  1.000000  0.491903
0.715542  1.000000  0.369080
0.853815  1.000000  0.211044
1.000000  1.000000  0.000100
'CAMBR'  2
'TABPTS'   'XIETA'
6  1
22  'XIETA'
0.000000  0.000000  2.079884
0.031623  0.000000  2.053980
0.089443  0.000000  2.008792
0.164317  0.000000  1.954450
0.252982  0.000000  1.896194
0.353553  0.000000  1.838116
0.464758  0.000000  1.783796
0.585662  0.000000  1.736533
0.715542  0.000000  1.699454
0.853815  0.000000  1.675564
1.000000  0.000000  1.667785
0.000000  1.000000  2.159768
0.031623  1.000000  2.138197
0.089443  1.000000  2.100928
0.164317  1.000000  2.056845
```

```
   0.252982  1.000000  2.010739
   0.353553  1.000000  1.966445
   0.464758  1.000000  1.927369
   0.585662  1.000000  1.896686
   0.715542  1.000000  1.877421
   0.853815  1.000000  1.872501
   1.000000  1.000000  1.884784
   'SOLID'  2
   2  0.0
'PLATE'  3
   1234
   'PLANF'  3
      2.000000  8.101310  7.248808
      2.195332  12.598788  6.668609
   'DEPTH'  3
   'TABPTS'   'XIETA'
   6  1
   22  'XIETA'
   0.000000  0.000000  0.000100
   0.031623  0.000000  0.305710
   0.089443  0.000000  0.472462
   0.164317  0.000000  0.576111
   0.252982  0.000000  0.625309
   0.353553  0.000000  0.624531
   0.464758  0.000000  0.578407
   0.585662  0.000000  0.491903
   0.715542  0.000000  0.369080
   0.853815  0.000000  0.211044
   1.000000  0.000000  0.000100
   0.000000  1.000000  0.000100
   0.031623  1.000000  0.242516
   0.089443  1.000000  0.374798
```

```
0.164317  1.000000  0.457021
0.252982  1.000000  0.496050
0.353553  1.000000  0.495432
0.464758  1.000000  0.458843
0.585662  1.000000  0.390220
0.715542  1.000000  0.292786
0.853815  1.000000  0.167418
1.000000  1.000000  0.000100
'CAMBR'  3
'TABPTS'   'XIETA'
6  1
22  'XIETA'
0.000000  0.000000  2.159768
0.031623  0.000000  2.138197
0.089443  0.000000  2.100928
0.164317  0.000000  2.056845
0.252982  0.000000  2.010739
0.353553  0.000000  1.966445
0.464758  0.000000  1.927369
0.585662  0.000000  1.896686
0.715542  0.000000  1.877421
0.853815  0.000000  1.872501
1.000000  0.000000  1.884784
0.000000  1.000000  2.257002
0.031623  1.000000  2.239643
0.089443  1.000000  2.209756
0.164317  1.000000  2.174613
0.252982  1.000000  2.138192
0.353553  1.000000  2.103698
0.464758  1.000000  2.073993
0.585662  1.000000  2.051750
0.715542  1.000000  2.039524
```

```
   0.853815  1.000000  2.039790
   1.000000  1.000000  2.054968
   'SOLID'  3
   3  0.0
'PLATE'  4
   1234
   'PLANF'  4
      2.195332  12.598788  6.668609
      2.390664  17.096265  6.088410
   'DEPTH'  4
   'TABPTS'   'XIETA'
   6  1
   22  'XIETA'
   0.000000  0.000000  0.000100
   0.031623  0.000000  0.242516
   0.089443  0.000000  0.374798
   0.164317  0.000000  0.457021
   0.252982  0.000000  0.496050
   0.353553  0.000000  0.495432
   0.464758  0.000000  0.458843
   0.585662  0.000000  0.390220
   0.715542  0.000000  0.292786
   0.853815  0.000000  0.167418
   1.000000  0.000000  0.000100
   0.000000  1.000000  0.000100
   0.031623  1.000000  0.179339
   0.089443  1.000000  0.277160
   0.164317  1.000000  0.337964
   0.252982  1.000000  0.366825
   0.353553  1.000000  0.366369
   0.464758  1.000000  0.339311
   0.585662  1.000000  0.288565
```

```
   0.715542  1.000000  0.216513
   0.853815  1.000000  0.123805
   1.000000  1.000000  0.000100
   'CAMBR'  4
   'TABPTS'   'XIETA'
   6  1
   22  'XIETA'
   0.000000  0.000000  2.257002
   0.031623  0.000000  2.239643
   0.089443  0.000000  2.209756
   0.164317  0.000000  2.174613
   0.252982  0.000000  2.138192
   0.353553  0.000000  2.103698
   0.464758  0.000000  2.073993
   0.585662  0.000000  2.051750
   0.715542  0.000000  2.039524
   0.853815  0.000000  2.039790
   1.000000  0.000000  2.054968
   0.000000  1.000000  2.354235
   0.031623  1.000000  2.341089
   0.089443  1.000000  2.318583
   0.164317  1.000000  2.292378
   0.252982  1.000000  2.265640
   0.353553  1.000000  2.240945
   0.464758  1.000000  2.220610
   0.585662  1.000000  2.206808
   0.715542  1.000000  2.201622
   0.853815  1.000000  2.207076
   1.000000  1.000000  2.225151
   'SOLID'  4
   4  0.0
'NFORC'  1
```

```
   1  1234  1.0
   5  1
   1
   3  3.9073  17.0996  0.0  1550
'STAT'  1
   1
'OSOLTNV'  21
   1  1  7
'PDISP'  1
   1  1
   1  4
   6  6
   2  15.0853
'ENDDATA'  1
```