Modification of a Vortex-Panel Method to Include Surface Effects and allow Finite-Element Interface

by

Scott R. Simmons

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Engineering Mechanics

APPROVED:

Dr. Dean T. Mook, Chairman

Dr. Saad A. Ragab

Dr. Robert A. Heller

August, 1994
Blacksburg, Virginia

Modification of a Vortex-Panel Method to Include Surface Effects and allow Finite-Element Interface

by

Scott R. Simmons

Dean T. Mook, Chairman

Engineering Mechanics

(Abstract)

A vortex-panel method for potential flow is used as a basis for modeling surface effects and creating a finite-element interface so that an arbitrary body can be analyzed. The basic model consists of triangular panels of linearly varying vorticity which represent the body, vortex cores on the lifting edges of the body, and vortex filaments representing the wake. The interface modification is made by using a finite-element application's output as the basis for an input file for the model, executing the main program, and writing body and wake output readable by the finite-element application. The surface-effect modification is made by including an image of the body below the real body to create a surface boundary condition through symmetry.

Acknowledgments

This work would not have been possible without the guidance and direction of D. T. Mook. I also thank the members of my committee, Dr. Heller and Dr. Ragab for spending their valuable time on my committee. I would also like to thank my family and friends, whose support kept me going long enough to finish this work.

Table of Contents

List of Figures

Introduction

The numerical simulation of fluid flow is like many other engineering disciplines: the closer the model is to reality, the more difficult it is to understand and implement. The purpose of this research is to modify an existing numerical model by 1) making it easier to use and 2) by including a surface effect without significantly complicating the model. The first goal is accomplished by writing translation programs such that any body created in a finite element processor can be read by the model. After the calculations are done, the final position of the body and the resultant wake are written in a form the finite element processor can read, and the body and its wake are displayed in their final form. The second goal is achieved by using an image of the body directly below the real body such that a horizontal plane of symmetry is created where the surface condition is to be imposed. The specifics of this approach will be investigated later in the paper. Next, background will be given on the research done concerning both the general numerical simulation of fluid flow and the influence of surface effects, and the motivation for the present modifications will be presented.

1.1 Numerical Simulation Research

Several models for computationally predicting incompressible potential flow over closed bodies have been implemented over the last thirty years. These models, for the most part, have used "a vortex lattice to represent the lifting surface and nonintersecting discrete vortex lines to represent the wakes" [1]. Initial linear models were accurate for small angles of attack only, and a nonlinear model was introduced by Ermolenko in 1966 [2]. Belotserkovskiy modified Ermolenko's model into a vortex lattice arrangement in 1969 [3], and later vortex-lattice models were developed to handle more general lifting surfaces. For example, Mook and Maddox [4] created a model capable of analyzing highly swept delta wings, and both Rehbach [5] and Kandil, Mook, and Nayfeh [1] developed models able to accommodate lifting bodies with large camber. Kim [6], and Kim and Mook [7], and Mook and Dong [8] used a two-dimensional panel method with linearly varying vorticity to predict incompressible, potential flow over closed bodies. This model was extended to three dimensions by Mracek, Mook, and Nayfeh [9], and it predicts the flow over a three-dimensional body using a vortex-panel method: planar triangular elements represent the vortex sheet of the body, vorticity is contained at any lifting edges by vortex cores, and the vorticity at the edge cores is convected away in the wake. This model is the basis for the current research.

1.2 Surface Effects Research

The increase in lift an airplane experiences as it flies close to a surface such as the ground or the sea has been noted by pilots ever since planes have been taking off and landing. The first recorded instance of pilots taking advantage of this effect was in the 1920's when Germany's Dornier flying boats flew very low over the sea to conserve fuel [10]. This effect was first modeled in 1922, when Wieselsberger placed an image of the lifting body beneath the real body such that a plane of symmetry was made where the ground condition was to be enforced. Since then, several countries have designed surface effect airplanes, including the former Soviet Union, Germany, China, Japan, Korea, and the United States. The Russian engineers Rostislav Alexeyev and Robert Bartini did the initial research into the surface-effect phenomenon in the 1940's. German scientists Lippisch and Jorg designed surface effect wings; Lippisch based his on a reversed delta planform, and Jorg designed stubby wings that were arranged together to increase the surface effect [10].

The numerical simulation of the surface effect is based on this concept of using a mirror image to create a ground boundary condition using symmetry. Initial numerical models considered a wing fixed in time, and the surface effect was observed as the altitude above ground was changed [10]. The model developed by Chen and Schweikhard in 1985, however, considered unsteady flow as the wing descends toward the ground [10]. According to their research, the unsteady effect is greater

than the steady effect for intermediate heights and less than the steady effect for very low heights. In 1988, Nuhait used a vortex-lattice method to predict unsteady ground effects, including varying sink rates, cross-winds, and the use of control surfaces [11]. In addition to supporting the conclusions of Chen and Schweikhard, Nuhait's model predicted an increased pitch angle response and a decrease in the effectiveness of the tail in controlling pitch near the ground.

1.3 The Current Model

The model developed by this research predicts incompressible, potential flow for closed bodies undergoing prescribed motion. The method accepts finite element representations of the closed bodies (as long as the surface has triangular elements), allows viewing of the closed body and its wake in the finite element application, and incorporates the surface-effect condition. The generalization of the basic model should provide easy access for planforms designed using a finite-element package, and the inclusion of surface effects should prove valuable to companies researching airplanes designed to take advantage of such effects. An example is Aerocon, Inc.; it is part of the United States effort in surface-effect airplane research, and it is currently designing a very large surface-effect airplane called the **wingship.**

The Aerodynamic Model

In this section, the mathematical basis for the numerical simulation is presented for both the basic model and the surface effect. Then, the numerical model is discussed: the details of the basic method are summarized (for more information, refer to [9]), and the modifications made to allow the finite element interface and the surface effects are discussed.

2.1 The Analytical Model

The basic analytical model is the same as that set forth in [9]. The velocity is considered in two parts: the total velocity ($\bar{V}$) is equal to the freestream velocity ($\bar{V}_{fs}$) plus the disturbance velocity ($\bar{V}_d$). The basic law applied to the flow is the continuity equation, expressed as

$$div\ \bar{V} = 0, \tag{1}$$

where $\bar{V} = \bar{V}_{fs} + \bar{V}_d$.

The boundary conditions are the no-penetration condition

$$\bar{V} \bullet \bar{n} = 0 \text{ on S,} \tag{2}$$

where $\bar{n}$ is the unit normal on the surface S, the freestream condition

$$\lim_{d \to \infty} \bar{V} = \bar{V}_{fs} \ , \tag{3}$$

where d is the distance from the lifting body and its wake, the Kutta condition

$$\Delta c_p = 0 \ \text{ on L,} \tag{4}$$

where L is the edge where the wake joins the surface of the body, and the conservation of circulation in the wake (Kelvin and Helmholtz theorem)

$$\frac{D\Gamma}{Dt} = 0 \text{ on W,} \tag{5}$$

where W is the wake. This condition ensures that the wake will be force-free.

The disturbance velocity is induced by three sources of vorticity: the surface vorticity from the vortex sheet representing the body, the vorticity from the vortex cores representing the edges of the lifting surface, and the vorticity produced in the wake. The surface vorticity ($\bar{\upsilon}$) and the vorticity $\overline{\Omega}$ are related in the equation

$$\bar{\upsilon}(\bar{s},t) = \lim_{|\bar{\Omega}|\to\infty, h\to 0} \bar{\Omega} h, \qquad (6)$$

where h is the thickness of the region containing vorticity above the body. The equation defining the disturbance velocity due to the surface vorticity is given in

$$\bar{V}_{d1}\,(\bar{r},t) = \frac{1}{4\pi} curl_{\bar{r}} \iint_S \frac{\bar{\upsilon}\,(\bar{s},t)}{|\bar{r}-\bar{s}|} d\sigma, \qquad (7)$$

where $\bar{\upsilon}\,(\bar{s})$ is the surface vorticity at point $\bar{s}$, and $\bar{V}_{d1}\,(\bar{r})$ is the disturbance velocity at point $\bar{r}$ due to the surface vorticity of the body. A mathematical constraint applies to the surface vorticity: since it is the curl of a vector, its divergence must be zero. Any velocity field calculated from the surface vorticity will also have a zero divergence; this satisfies equation 1.

The vortex cores are a linear representation of the strength of the surface vorticity from the triangular element, and they lie along the edges of the elements that are coincident with the edge of the lifting surface. The relationship between the circulation around the vortex core and the vortex strength of the sheet is given by the equation

$$\frac{d\Gamma}{dx} = -\upsilon, \qquad (8)$$

where x is the position along the length of the vortex core and Γ is the circulation at position x along the core. The disturbance velocity due to the circulation in these vortex cores is given in the equation

$$\bar{V}_{d2}(\bar{r},t) = \frac{1}{4\pi}\int_L \Gamma(x,t)\frac{\overline{dl}\times(\bar{r}-\bar{s})}{|\bar{r}-\bar{s}|^3}, \qquad (9)$$

where L is the length of the vortex core, x is distance along that length, $\bar{r}$ is the point of interest, $\bar{s}$ is the point where the vorticity is located, and $\overline{dl}$ is the vector element of length along the core.

The remaining vorticity is the free vorticity in the wake, and it is computed according to the general equation

$$\bar{V}_{d3}(\bar{r},t) = \frac{1}{4\pi} curl \iiint_W \frac{\overline{\Omega}(\bar{s},t)}{|\bar{r}-\bar{s}|} d\tau, \qquad (10)$$

where $\overline{\Omega}$ is the vorticity at point $\bar{s}$, and W is the wake.

All three of these disturbance velocities combine to give the total disturbance velocity discussed earlier. With the freestream velocity known, equation 2 can be numerically represented. Also, the sum of the disturbance velocities tends toward zero as the distance from the body increases since the denominator goes to infinity in all three cases. This satisfies the freestream condition in equation 3.

Two more quantities must be explained for the remaining equations to have all their variables defined: the vector normal to the surface, and the coefficient of pressure. The normal vector, $\bar{n}$, is defined as the unit vector parallel to the cross-product of the first and second sides of the triangular element. (The sides are numbered such that this product will produce a vector pointing out of the body). The difference in the coefficients of pressure, Δc_p, is obtained from Bernouilli's equation

for unsteady flow:

$$\Delta c_p = 2[(\overline{V}_m - \overline{V}_p) \bullet \Delta\overline{V} + \frac{\partial}{\partial t}(\Delta\Phi)], \qquad (11)$$

where $\overline{V}_m$ is the mean velocity between the upper and lower surfaces, $\overline{V}_p$ is the velocity of the control point, and $\Delta\Phi$ is the jump in the velocity potential across the surface.

These equations represent the basic aerodynamic model presented by Mracek, Mook, and Nayfeh [9]. The surface effect is included in these equations by including the images of the body and wake as well as the body and wake themselves. In this way, the surface vorticity at the image point $\bar{s}$ affects the disturbance velocity at $\bar{r}$, creating the surface effect by symmetry as earlier discussed.

2.2 The Basic Computational Model

This section describes the computational model as presented by Mracek, Mook, and Nayfeh [9]. This model consists of two basic parts: the first part is the program INPFOR.F, which creates the body in the form of an input file, and the second part is the main program 3D.F, which reads the input body, does the main computations, and writes the output.

2.21 INPFOR.F

This program generates the input data for three types of shapes: spheroids, rectangular wings, and delta wings. A spheroid body is created using the following parameters: number of latitudinal segments, number of longitudinal segments, length of the major axis, and the finess ratio. A rectangular wing is created by inputting the number of rows, the number of columns, and the aspect ratio. A delta wing is formed by specifying the number of rows and the aspect ratio. From this input information, the coordinate and connectivity matrices are generated. This basic input is enough to define the structure of the body, but additional information is needed to perform a numerical flow analysis on the body.

In addition to providing the basic structure of a body, INPFOR.F also provides the following information about the body: the number and positions of the edge cores, the number and positions of the convecting edge cores, the number and locations of the starting wake positions, the number and locations of the nodes where the Kutta condition is imposed, the number and locations of the corner nodes, the number and identifications of the starting nodes for the potential, the path for the potential, the reference length and velocity, the chord, the span, and the centroid. Once all the pertinent details of the body are defined, the final information about the specific test case is input: the initial conditions, the length of the wake to be convected, the number of time steps to run, and the time increment.

2.22 3D.F

The main program for the numerical model, **3D.F** accepts the input data created in the INPFOR.F program and outputs data about the resulting velocity, vorticity, and pressure distributions, the convected wake position, and the net lift and drag forces. The program has three main sections: the input and initialization section, the calculation section repeated every time step, and the postprocessing and output section.

2.221 Input

The first step in setting up the body to be analyzed is reading in the input data. This is done in the subroutine **READER**, which reads the input file and stores the data mentioned above in the appropriate variables. Next, the subroutine **REDUCK** is called to check the nodes and elements for redundancies. This is done by testing the nodes to be sure they all have unique coordinates and are included in at least one element, by testing the triangular elements to determine if any contain repeated nodes or are repeated, and by testing the edge segments and starting wake positions for repeated nodes. The subroutine **SETEDG** is called next; it takes the edge data and determines which edges form a complete circuit (called closed circuits) and which do not (called open circuits), and it also reorders the edge data such that the edge segments are ordered according to their connectivity. The next subroutine, **SETCON**, also manipulates the edge data. It establishes a connection between the circulation on

the convecting edges of the body and the circulation around the closed loops of the wake. **NEWELE** is then called to renumber the nodes of the elements so that a unique vorticity can be found on each side of the edge. The following subroutine is called **ARRCON**, and it finds the triangular element constants. The constants determined for each element are: the control point, the unit normals, the local vertices, the basis functions, and the edge constants. The control point is found by averaging the element coordinates, the unit normal is found by doing a cross-product of two of the vectors forming the triangular element, the local vertices are found by determining the length of the sides of the triangular element, the basis functions are defined based on the local vertices, and the edge constants are set based on the edge circuits determined in the previous three subroutines. **NORMCK** is the next subroutine called; it determines if the adjacent triangular element's normals are compatible. It does this by computing the dot-product of the normals of adjacent elements and reversing the direction of the normal of one of the elements if the angle between the two normals is greater than or equal to equal to ninety degrees. The last subroutine called in this section is **PHIPAT**. It processes the input information about the path of the potential and creates the array that refers to this path.

2.222 Calculation

After all the input information is checked for errors and organized, the main calculation is begun. The subroutine **SOLVER** runs the sequence of subroutines that

does the calculations at each time step. The first subroutine called is **OMEGAS**; it determines the angular rates of rotation in the body-fixed coordinate system using the Euler angular rates. The computation is then formulated by placing all the constraint equations into the general form

$$[\text{Left-Hand Side}]\ [\overline{\Omega}] = \text{Right-Hand Side.}$$

SETRHS is the subroutine that determines the right-hand side of the matrix equation. The top half of the matrix is the right-hand side of the no-penetration condition, expressed as the equation

$$RHS = -(\overline{V}_{fs} + \overline{V}_{d3}) \bullet \overline{n} \quad ,$$

where V_{fs} is the freestream velocity defined in the subroutine **VELLS**, and V_{d3} is the velocity at the element's control point induced by the wake, which is calculated in the **VELWK** and **VELCF** subroutines by using the general Biot-Savat law given by equation 10. The no-divergence condition right-hand side is stored in the array next; it is identically zero. After the two main conditions are met, the right-hand side of the circuit constraint equation (equation 5) is set equal to zero for both the edges and the corners of the vortex cores. The last right-hand side equation input is the Kutta condition: the change in potential with respect to time from the last time step $(\frac{\partial \Phi}{\partial t}_{\bar{r}})$ multiplied by a weighting factor.

After the right-hand side is defined, the left-hand side must be defined; this is the function of the **EINFLU** and **DIVERG** subroutines. The **EINFLU** subroutine calculates the influence matrix for the no-penetration condition by calling the **VELE** and **VELVF** subroutines. The **VELE** subroutine calculates the effect that a unit vorticity on the surface of the body has on the normal velocity of the point in question using equation 7. The **VELVF** subroutine calculates the influence that a unit vorticity generated in the vortex cores along the lifting edges of the body has on the normal velocity of the point using equation 9. These results are added together to form the no-penetration portion of the left-hand side. The **DIVERG** subroutine finishes the formulation of the left-hand side. The left-hand side of the no-divergence condition is generated from the basis functions used for the triangular elements to define the surface vorticity. Since it is necessary and sufficient for the divergence of the surface vorticity to be zero given the divergence of the global vorticity is zero, the no-divergence condition can be expressed in terms of the global vorticity and input into the left-hand side of the main matrix equation. The left-hand side of the equation for the circuit constraint equation is calculated next: coefficients are set for the edges and corners such that the core vorticity is continuous. Finally, the left-hand side of the Kutta condition is input: coefficients for defining the velocities in equation 11 due to unit vorticity are stored in the last section of the left-hand side matrix.

With the left and right sides of the equation defined, the matrix solution is almost ready to be solved. Unfortunately, too many equations exist to be identically

satisfied, so some method of minimizing the error must be used. The subroutine **REDOMA** left-multiplies the matrix equation by the transpose of the left-hand side, then the subroutine **SIMUL** solves the matrix for the unknown vorticities. This procedure has the effect of minimizing the square of the error. The solution of the matrix equation is followed by subroutines which use the computed solution: **FACTOR** unpacks the solution into the global vorticity strengths for each node and checks the continuity of the solution, **GAVERA** uses the circuit constraint solution to compute the average vorticity of each wake segment that will be convected away from the wing, and **FORMOM** finds the forces and moments on the body by summing the product of the pressure difference and the element area. This pressure difference is calculated from the velocity in the subroutine **LOPRES**. The next subroutine, **UPDAT1**, saves the potential for use in the next time step. One new subroutine is needed to calculate the total velocity: **VELBND** uses the vorticity solution to calculate the velocity at a point due to the bound vorticity. The remaining portions of the velocity can be calculated by the previously mentioned **VELWK** and **VELLS** subroutines. The last two subroutines in the calculation section are **CONVE** and **ISITOV**: **CONVE** convects the wake points at the calculated local velocity, and **ISITOV** enforces a computational no-penetration condition on the wake by moving a wake point if it is too close to the surface. After all the intermediate results are written and the wake is convected, the calculation repeats for the next time step until the last user-specified time is reached.

2.223 Output

The subroutine that handles the output at each time step is called **IMWRIT**. For each time step, it outputs the Euler angles and rates, the vorticity characteristics, the potential, the pressure, the force, and the lift. The final output is called **WRITER**, and it writes the final wake positions, the potentials, the vortex compatibility factors, the vortex strength, the edge circulation, the angle information, and the error vector. The output is written to several different files, and any error messages are printed to the screen during the execution of the program.

2.3 Modifications to Basic Model

Changes to the basic model have been made to facilitate an interface with a finite-element application and to include the ground effect. In addition, since the research has been restricted to steady-state problems, only these portions of the basic method were used. A basic modification has also been made to assist in accomplishing these objectives: the body was originally represented in local coordinates only; the subroutine **GEFFCT** uses the transformation matrix of the Euler angles to rewrite the body nodes into the global reference frame. The only calculation this effectively changes is the freestream condition; now it is input as simply a unit flow in the

negative x-direction. The changes specific to the generalization of the model and the inclusion of surface effects are discussed below.

2.31 Generalizing the Model

The first step in generalizing the model is to provide some interface between the finite-element application in which the body is created and the main program itself. In this research, the finite-element application used to model the body is **PATRAN**. **PATRAN** is capable of writing the connectivity and coordinate matrices for the body in a neutral file. An example of a body created in **PATRAN** is shown in Figure 1, and the exact commands needed to use this program are given in the appendix. The following information is also needed in order for the body to be analyzed: the node for the starting of the potential, the number and nodes of the leading edges as well as information indicating whether they convect, the number and nodes of the trailing edges, and the number and nodes of the corners. In general, this information can be acquired from a printout of the body that includes the node numbers. This information is requested by the EDGE.F program, which also formats the data for later reading into the main input program, P3G.F .

P3G.F replaces the program INPFOR.F in the input portion of the model. It has three subroutines: **READER**, **EDGE**, and **WRITER**. **READER** requests the name of the neutral file and reads the connectivity and coordinate matrices into arrays.

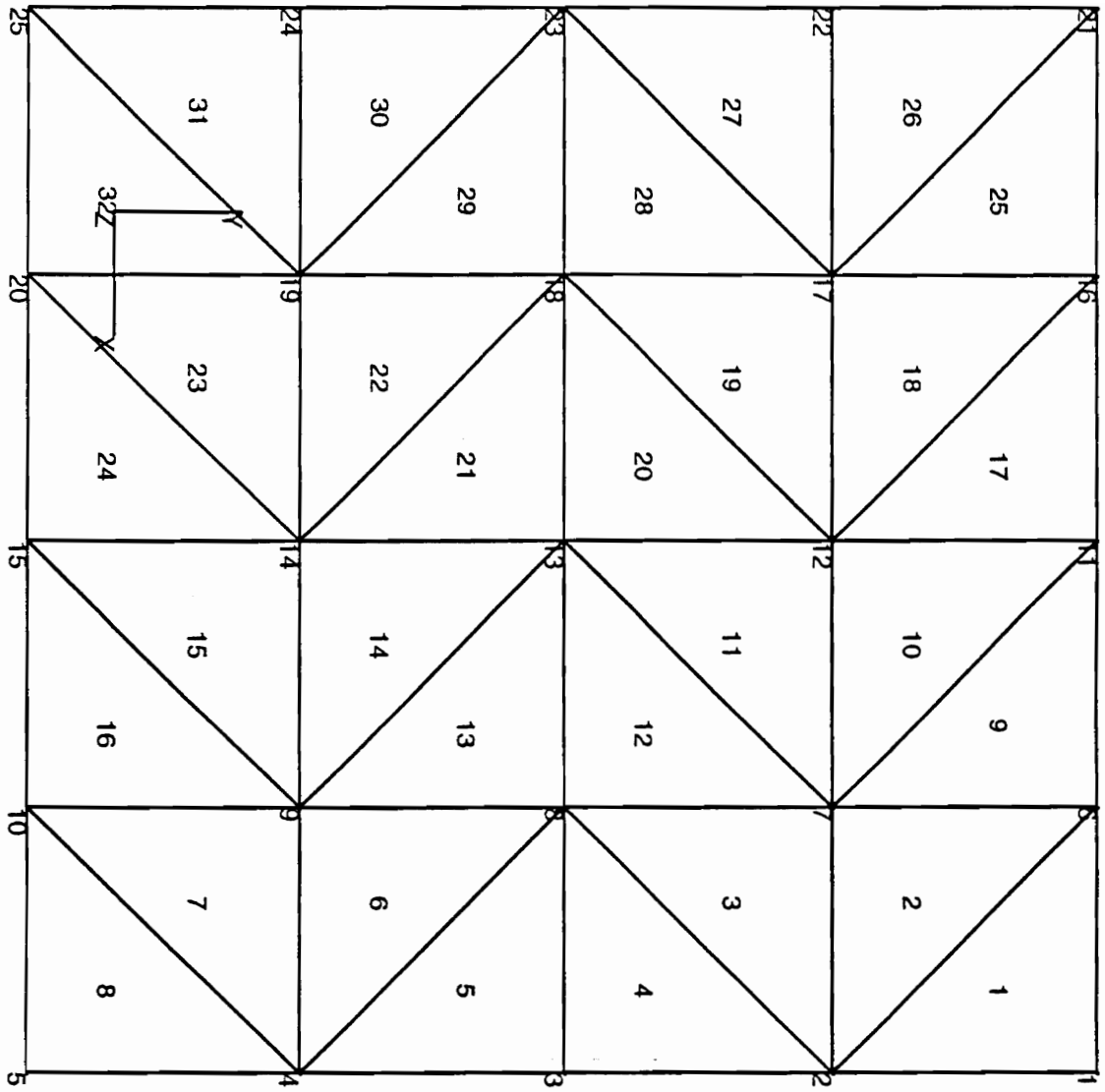

Figure 1: A rectangular wing created in Patran

The **EDGE** subroutine reads in the data mentioned above from the EDGE.F output file and stores it in the appropriate arrays. The main calculation and output is done in **WRITER**. This subroutine calculates and writes all the information the main program needs to the main input file; it also sorts the data in ascending order since some of the main program's subroutines function better with sequential data. After including the identification information to the input file, the connectivity and coordinate matrices are written. Then the edge information is calculated and written. First, the number of edge cores are calculated from the number of leading and trailing edge cores and written to the file. Then, the two nodes defining each edge are sorted and written. Next, the number of convecting edge cores are determined, and the edges that convect are sorted and written. Then, the starting wake positions are calculated from the nodes contained in the convecting edge cores from both the leading and trailing edges. The convecting nodes from the trailing edge are written as nodes where the Kutta condition is imposed. Next, the corner nodes are sorted and written, and the starting location for the potential is defined. The algorithm for finding the path of the potential is based on a generalized version of the one used in INPFOR.F. In this generalization, the path-finding portion of the program is repeated until all the nodes have a path to them. This allows a more complicated structure to be modeled in the program by guaranteeing the path will include all nodes. The rest of the input program is very similar to INPFOR.F with the exception of the length properties: the span, chord, and centroid are calculated based on the coordinates of the nodes.

Modifications have also been made to the main program in order for a body of arbitrary configuration to be analyzed. The first modification is in the **ARRCON** subroutine; the original subprogram assumes the longest side of the triangle is given first. This is not necessarily the case when an arbitrary body is input, so the subroutine **SETELE** is called to reorder the nodes of the triangular elements such that the longest side is listed first. A listing of this routine is given in the appendix. Another change to the basic program occurs in the **NORMCK** subroutine; this subprogram checks the compatibility of the element normals and is not needed in the basic method, since the input is ordered for the specified bodies. For an arbitrary ordering of the nodes, even given that the first two nodes define the longest side, it is possible for adjacent elements to have normals oriented incompatibly--more than ninety degrees apart . The original subroutine checks the compatibility of the normals by looping through the elements and computing the dot-product of the normal vectors of element i and element $i+1$; if the normals are incompatible, the orientation of the normal for element $i+1$ is inverted. For an arbitrary element configuration, however, this scheme will not work: if element i has an outward-pointing normal, element $i+1$ has an inward-pointing normal, and element i and $i+1$ are not adjacent, the algorithm fails. To overcome this problem, the rewritten subroutine checks the first element's orientation and compares it to all the adjacent elements' normals. If any adjacent element normal is incompatible, its orientation is changed. Then, all of the elements adjacent to the elements adjacent to the first element are checked, and this is repeated

until all of the elements have been checked. This method guarantees total compatibility. The last subroutine modified is the **ISITOV** subprogram which prevents the wake from penetrating the wing. This is another subroutine that was mostly unnecessary in the earlier model due to the symmetry and simplicity of the body, but is now very important in analyzing a more complex body. The new subprogram is more stringent in its condition on the wake points, and the algorithm is based on the mathematical definition of the distance from a point to a plane: the distance from the point (x_p, y_p, z_p) to the plane defined by the points $(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$ is

$$\text{Distance} = \frac{(d - ax_p - by_p - cz_p)}{\sqrt{a^2 + b^2 + c^2}},$$

where

$$a = (z_2 - z_1) - (y_2 - y_1)b,$$

$$b = \frac{[(x_2 - x_1)(z_3 - z_1) - (x_3 - x_1)(z_2 - z_1)]}{[(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)]},$$

$$c = -(x_2 - x_1),$$

and

$$d = x_1 z_2 - x_2 z_1 + b(x_2 y_1 - x_1 y_2).$$

This is the general form derived from the vector equations given in **Multivariable Mathematics** [12]; special cases where the denominators were zero are also included in the subroutine **ISITOV**, listed in the appendix. Using the distance of the wake

point from the plane, coarse and fine adjustments can be made. The coarse adjustment is this: if the wake point is below the surface of the wing element, it is raised to the minimum z value on that element. The fine adjustment moves the wake point in the direction normal to the element until the wake point is at least as far away from the wing as the cutoff distance. Then the process is repeated for all of the wing elements until no wake points are too close to the wing.

There are two modifications made to the main program for the output which make the program compatible with the finite-element package. The first is a modification in the subroutine **WRITER**: instead of the wake points being listed sequentially, they are organized into rectangular elements; this has the effect of creating a connectivity matrix for the wake. The other change is the addition of two subroutines, **PATPLT1** and **PATPLT2**. The first one creates a neutral file for the body in its final orientation, and the second one writes a neutral file for the wake. Using these files, the body and its wake can be viewed in a finite-element application, allowing many views, hidden line removal, and other operations according to the features of the particular application.

2.32 Including Surface Effects

The inclusion of surface effects in the model is accomplished by representing the body in global coordinates, as discussed at the beginning of this section, and

calculating the induced velocity as if an image were present below the actual body. A representation of this concept is given as Figure 2. The first step in this process is to calculate the global unit normals for the image elements; this is done in the subroutine **ARRCON** by recomputing the normals with the node's z-coordinate negative instead of positive. When the subroutine **VELE** calculates the velocity due to the wing elements, the calculation is redone with negative z-coordinates and image normals, and the image velocity is subtracted from the velocity calculated initially to produce the total velocity. The same procedure is followed in the subroutine **VELVF** for the velocity induced by the vortex cores and in the subprogram **VELCF** for the velocity induced by the wake. The last modification is in the CONVE subroutine, where the wake points are not allowed to penetrate the ground. This is the method for including the image of the body in the calculation of the velocity, and it has the desired effect of imposing a surface by symmetry.

RESULTS

Three lifting surfaces have been analyzed by the modified model: unit aspect ratio delta and rectangular wings have been used to verify the model, and a planform representation of the wingship has been analyzed in an attempt to predict its response at various altitudes. The results for the first two wings will be discussed in this section

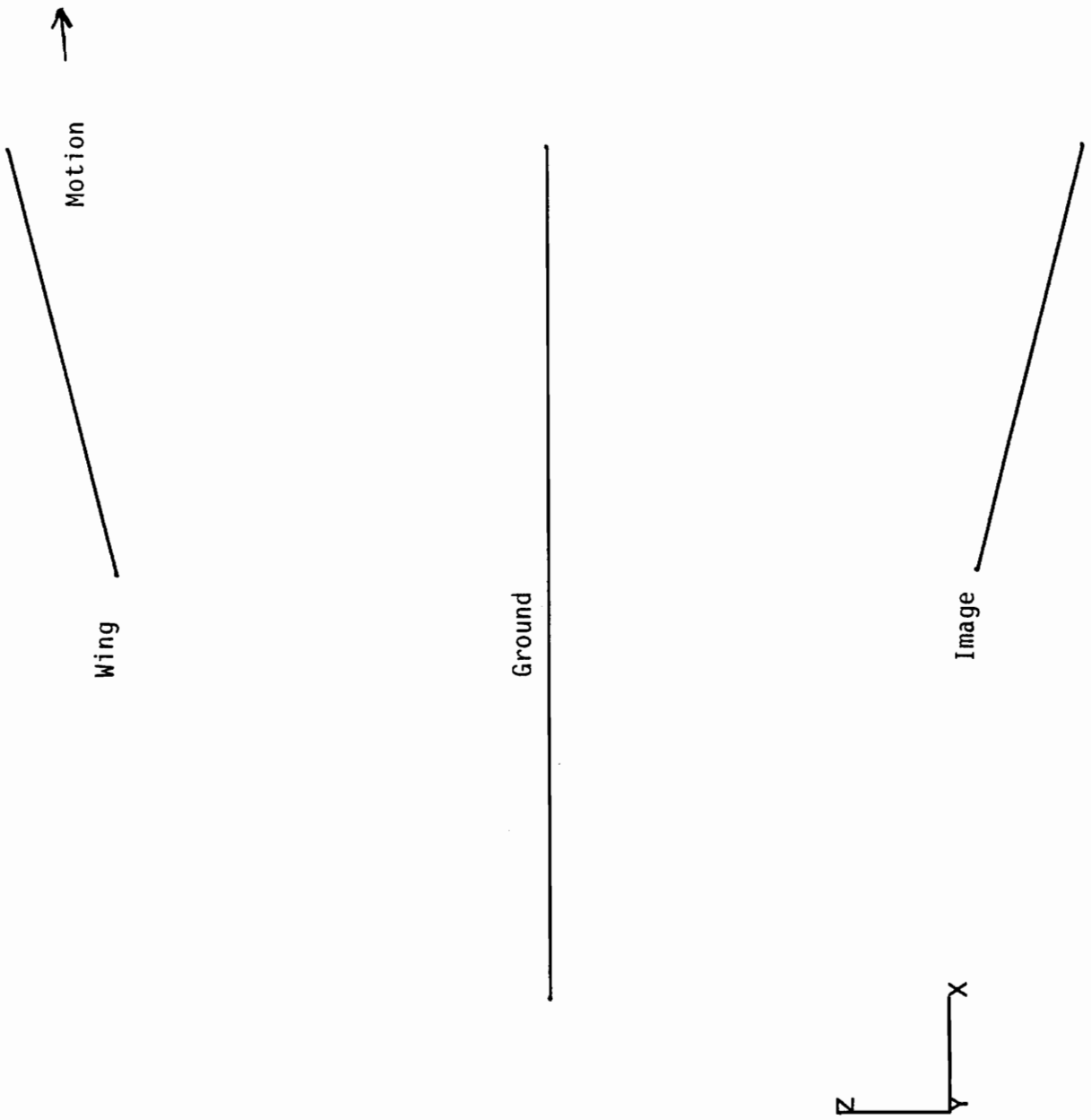

Figure 2: A representation of the wing and its image

and compared to other results for delta and rectangular wings both in and out of ground effects. The wingship planform results will also be discussed as an illustration of the versatility and limitations of this model.

3.1 Rectangular Wing

The rectangular wing used in the main calculations had an aspect ratio of one and a chord of three units, and it was analyzed at 5, 10, 15, and 20 angles of attack and altitudes ranging from 0.2 units to 100 units. A representation of a rectangular wing and its wake is given in Figure 3; this wing has a chord of five units, an altitude of one hundred units, and an angle of attack of ten degrees. As a comparison, Figure 4 shows the same wing experiencing ground effects at an altitude of .5 units; the wake is flatter and the coefficient of lift is higher. Figure 5 shows the effect of angle of attack on the coefficient of lift for the heights varying from 100 units to .2 units. The lowest line on the graph represents the lift coefficient for the rectangular wing out of ground effects, and Figure 6 shows the graph plotted by Kandil, Mook, and Nayfeh [1] for the normal force coefficients of unit aspect ratio rectangular wings. The lowest line on the graph in Figure 5, when divided by the cosine of the angle of attack to convert to the normal force coefficient, shows good agreement with the other numerical results shown in Figure 6. Figure 7 shows the effect of altitude on the coefficient of lift for different angles of attack; this graph show the increase in lift for

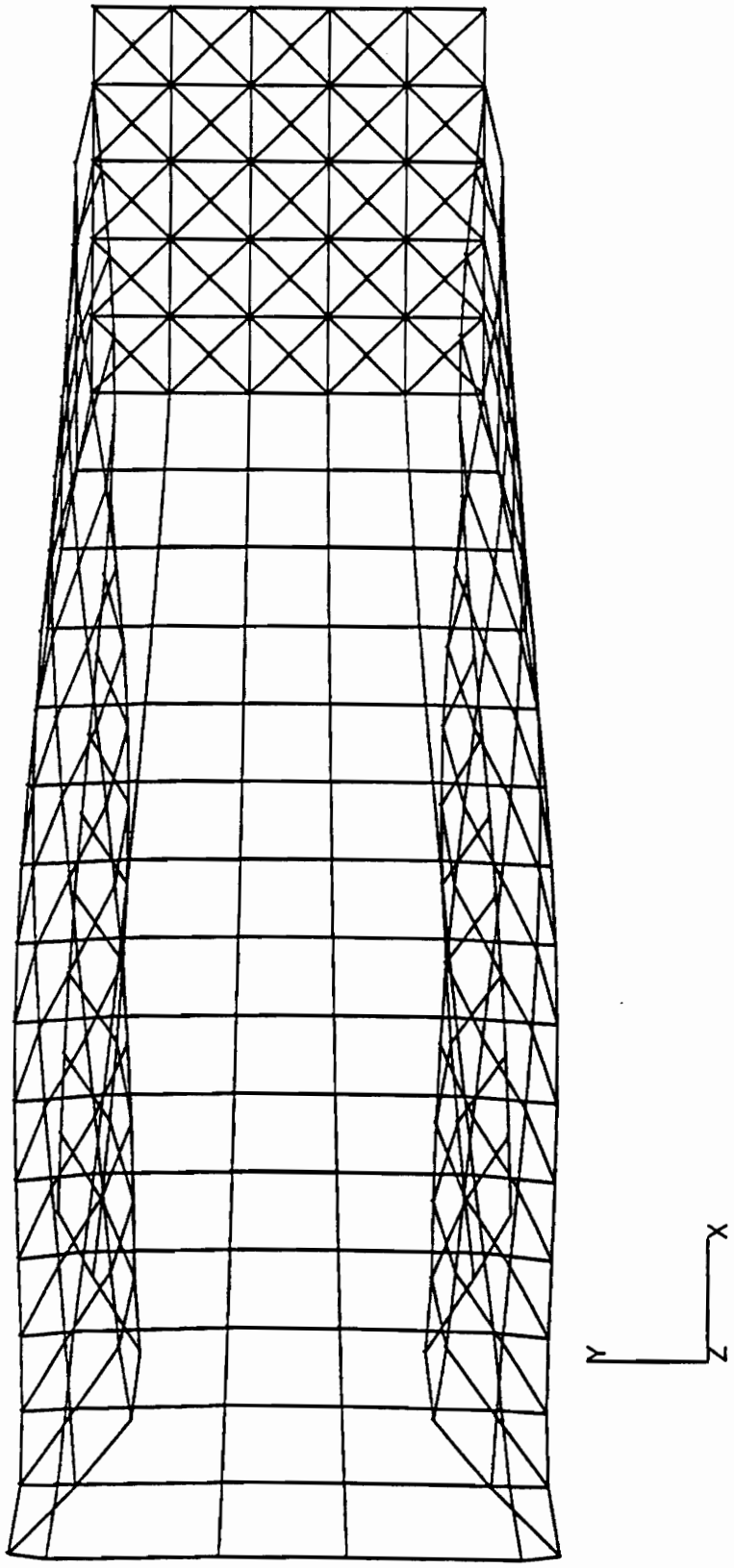

Figure 3a: Top view of a Rectangular Wing, Height=100 units, 10 degree pitch, CL=0.3405

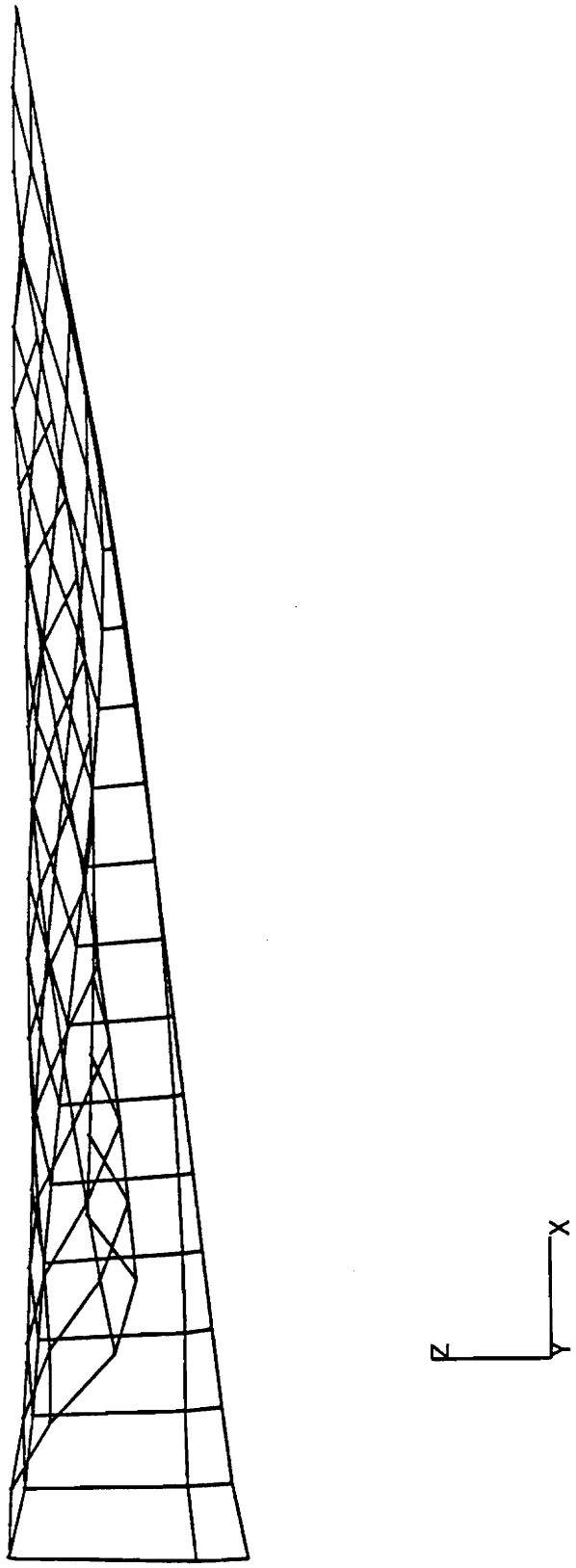

Figure 3b: Side View

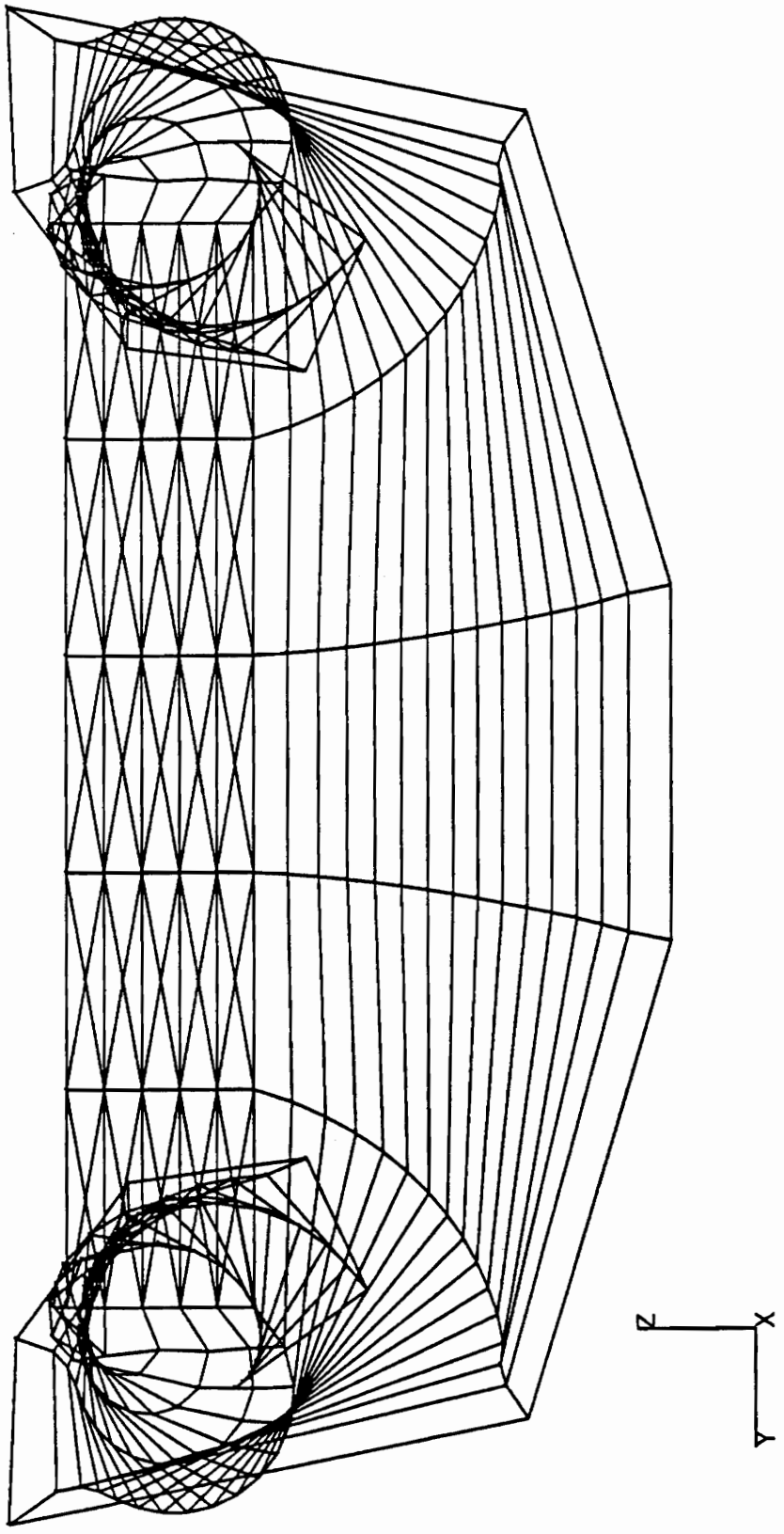

Figure 3c: Back View

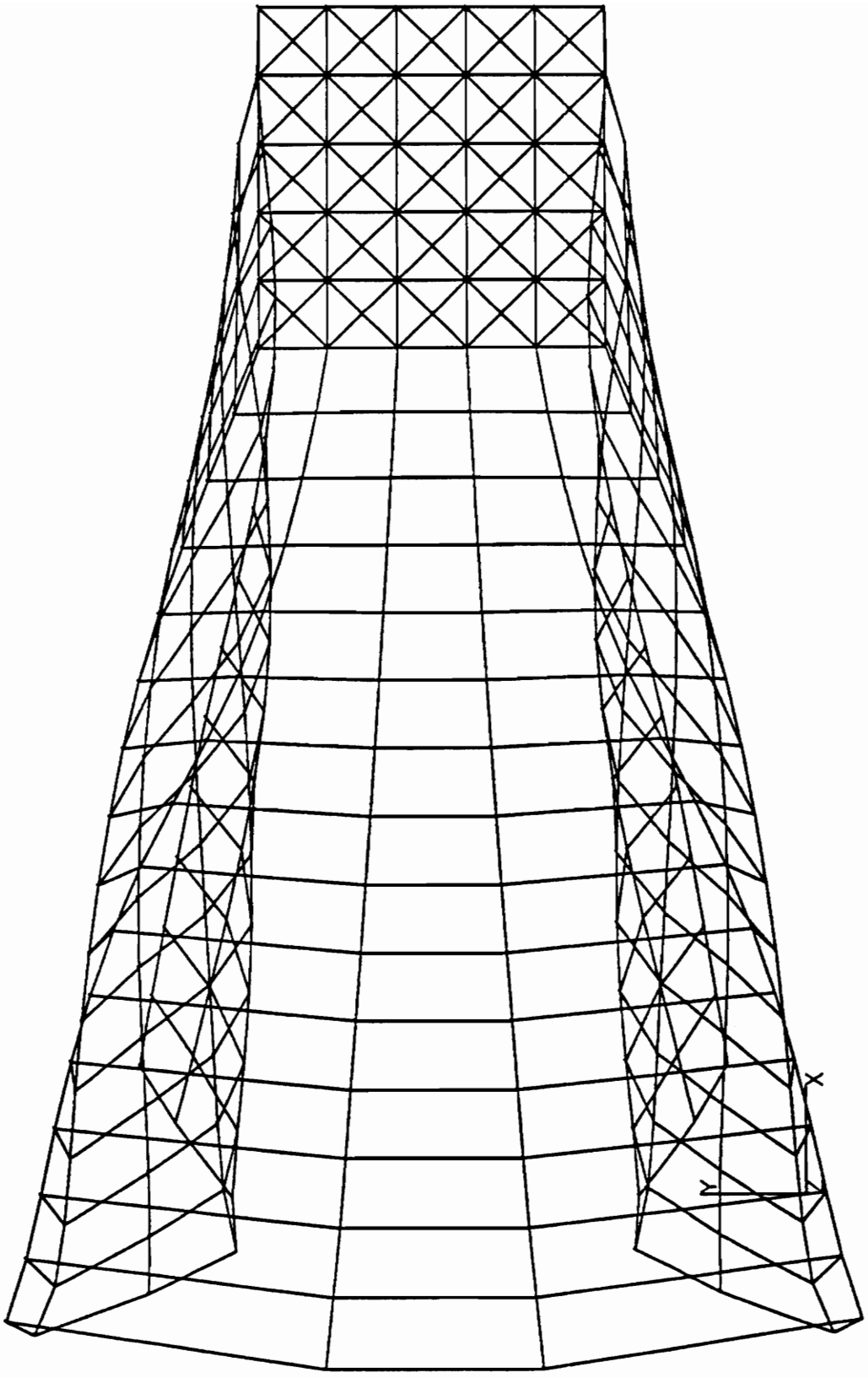

Figure 4a: Top view of a rectangular wing, height=.5 units
angle of attack=10 degrees, CL=.5342

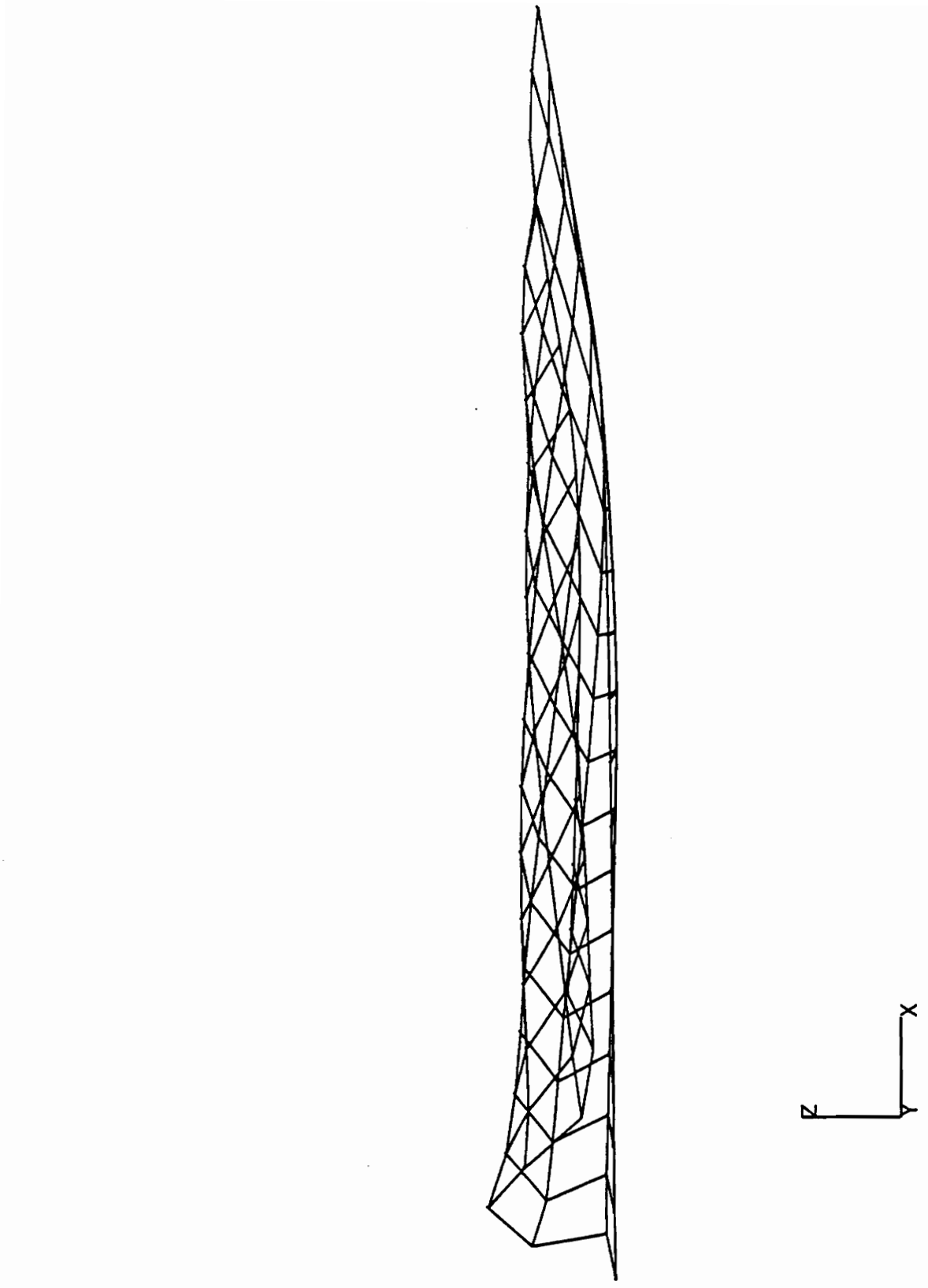

Figure 4b: Side view

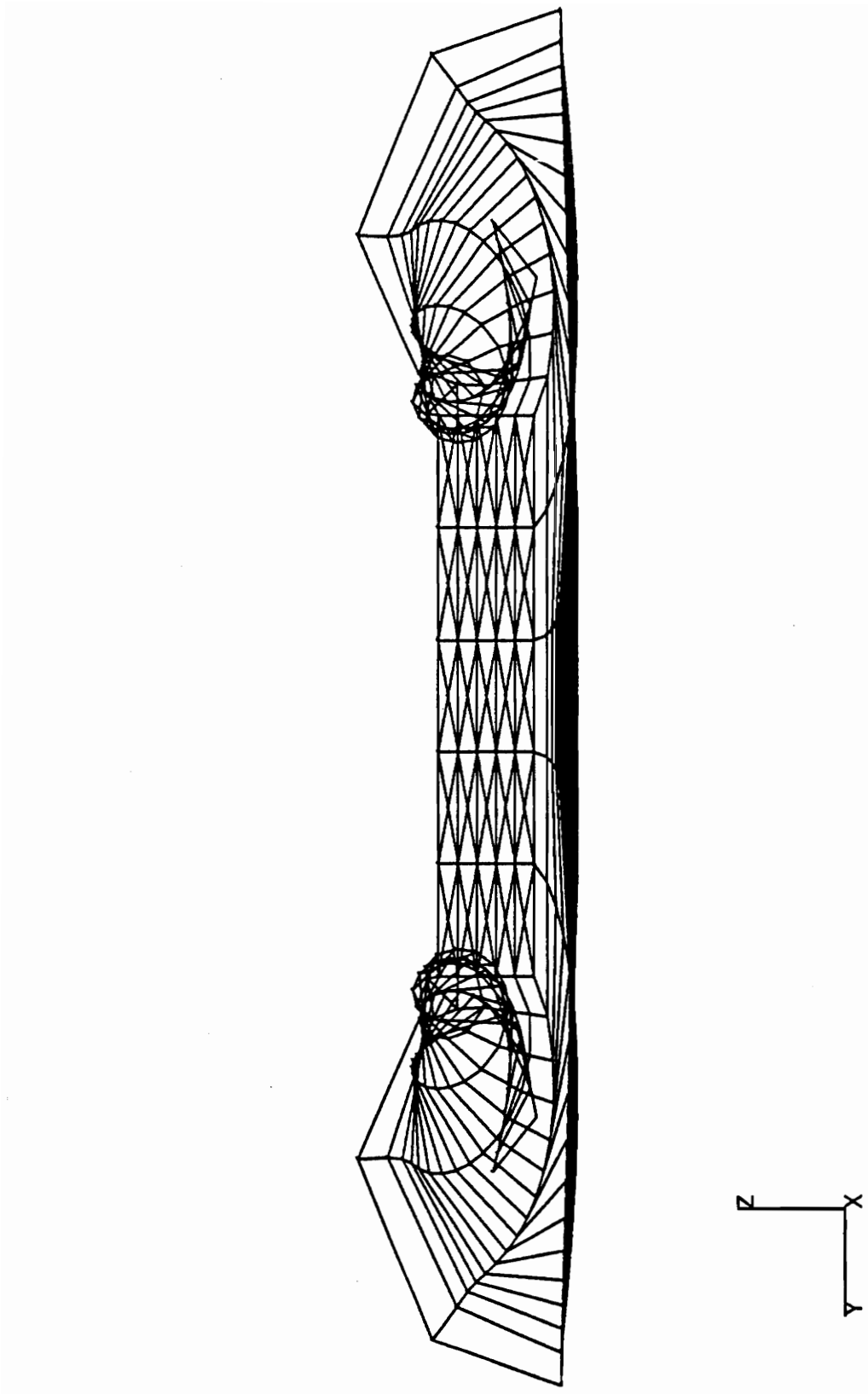

Figure 4b: Back view

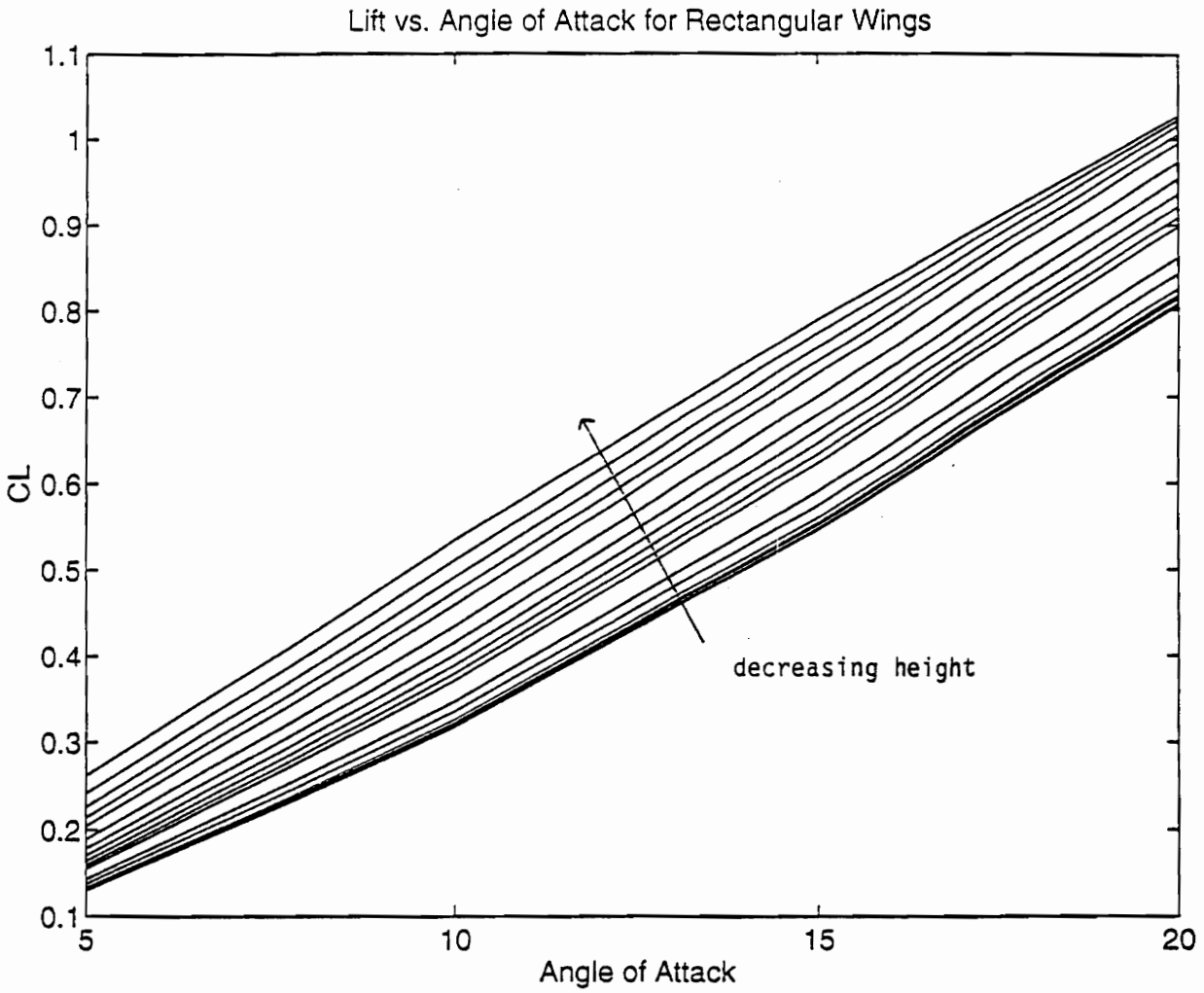

Figure 5: Lift vs. Angle of Attack for Rectangular Wings

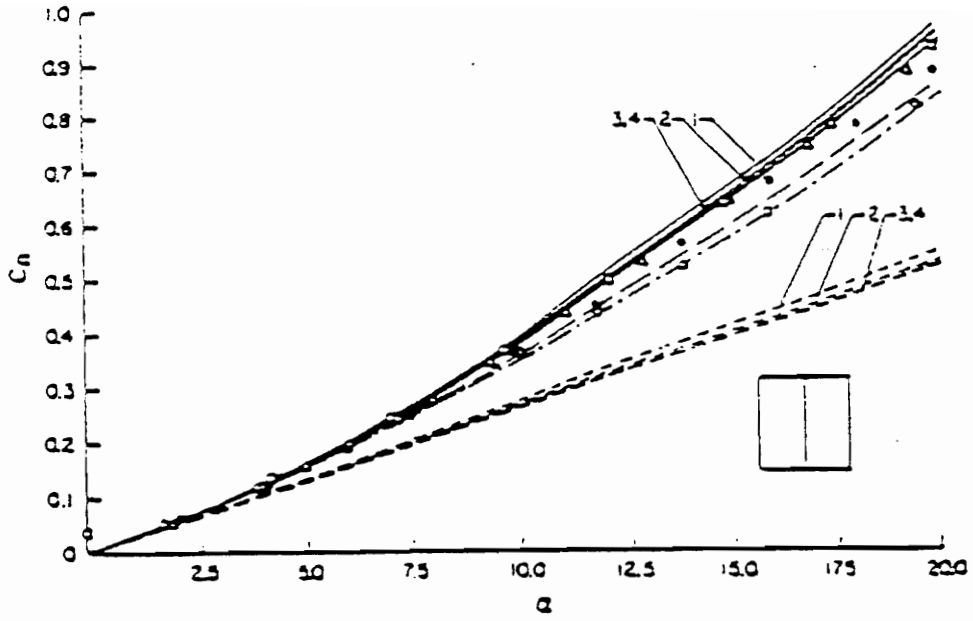

Fig. 4 Normal-force and pitching-moment coefficients vs angle of attack, $AR = 1$. Numerical results: —— Belotserkovskiy;[8] —— Ermolenko;[7] —— Bradley, et al.[23] ——— Linear, —— Nonlinear: Present method. Curve No. 1:4×4 lattice; curve No. 2:6×6 lattice; curve No. 3: 9×7 lattice; curve No. 4; 9×9 lattice. Experimental Results: • Belotserkovskiy;[8] ○ Ermolenko;[7] □ Winter;[6] △

Figure 6: Normal force coefficient vs. angle of attack plotted by Kandil, Mook, and Nayfeh [1]

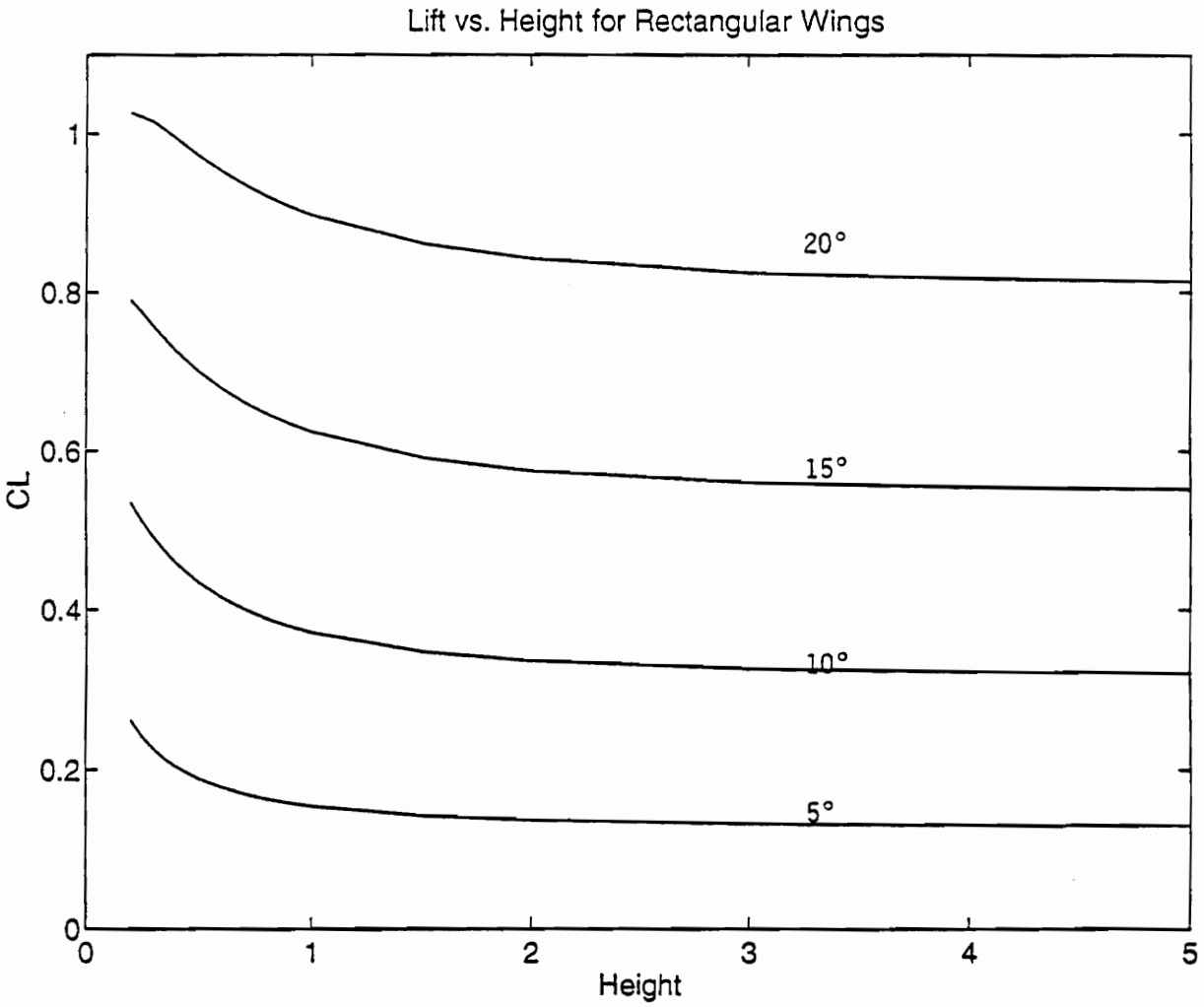

Figure 7: Lift vs. Height for Rectangular Wings

the rectangular wing as it nears the ground for all the angles of attack included. This information is also presented in Figure 8 as the percent increase in the lift coefficient due to the change in the altitude/chord ratio for different angles of attack. This information compares favorably with a similar plot (for only a ten degree angle of attack) plotted by Nuhait [11] and given in Figure 9. The model was stable for values of altitude as low as one tenth the chord of the rectangular wing, and good agreement with established high and low altitude lift coefficients indicates the model is predicting the flow correctly.

3.2 Delta Wing

The unit aspect ratio delta wing had a chord of five units, and it was also run at 5, 10, 15, and 20 angles of attack and altitudes ranging from 0.2 units to 100 units. The coefficient of lift converged in most cases very quickly, but in some cases the solution oscillated with a very small amplitude near the steady-state value. This numerical instability has not been corrected at the time of this paper, and its effect has been to make the solution less precise in some areas as well as corrupting some of the wake output. The output for the delta wing is shown in Figure 10 for an angle of attack of 15 degrees and an altitude of 100 units. The output for the delta wing experiencing ground effects is given in Figure 11 for an angle of attack of 10 degrees and a height of .4 units. Figure 12 shows the effect of angle of attack on the

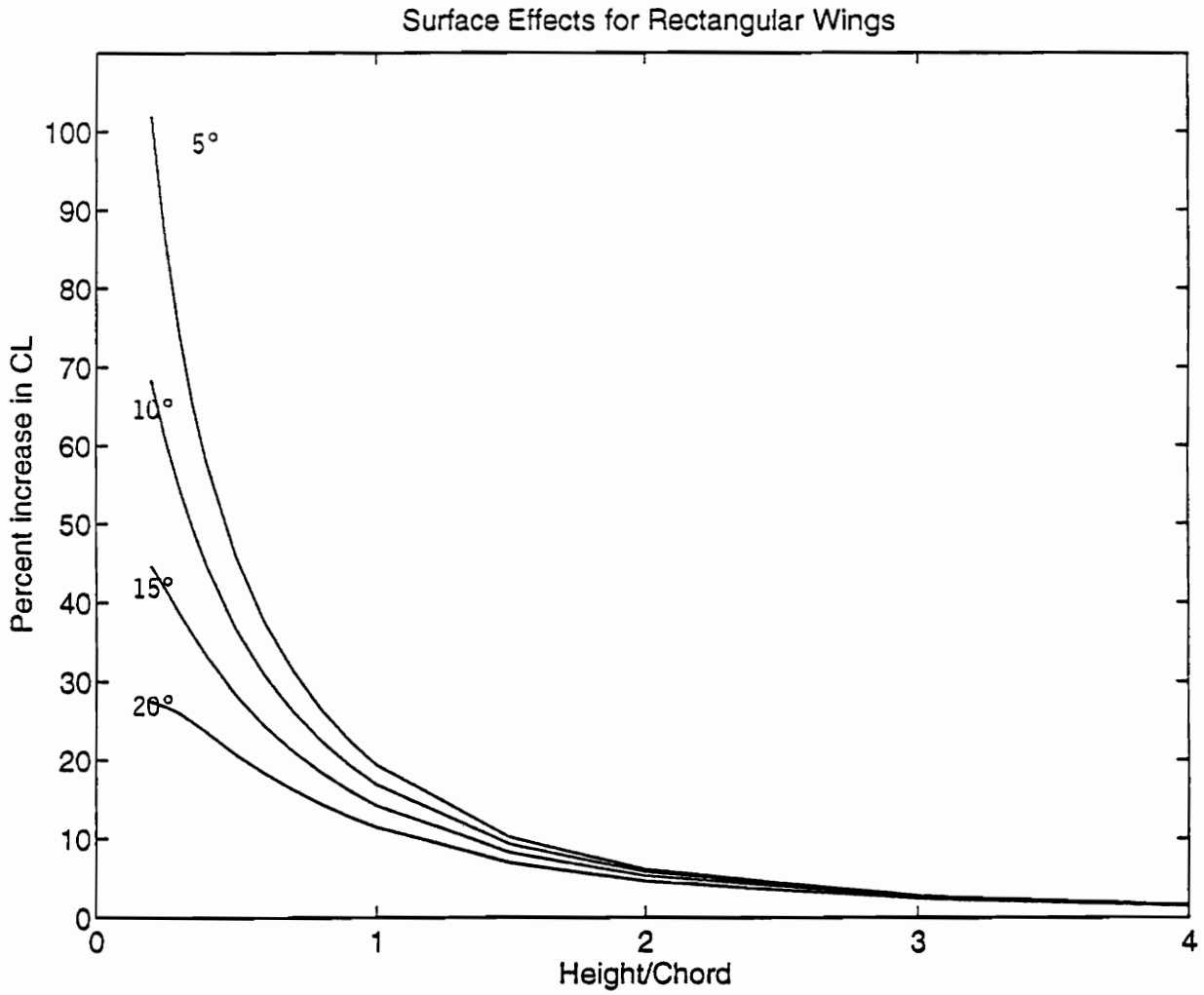

Figure 8: Percent increase in lift coefficient due to change in the altitude/chord ratio for different angles of attack

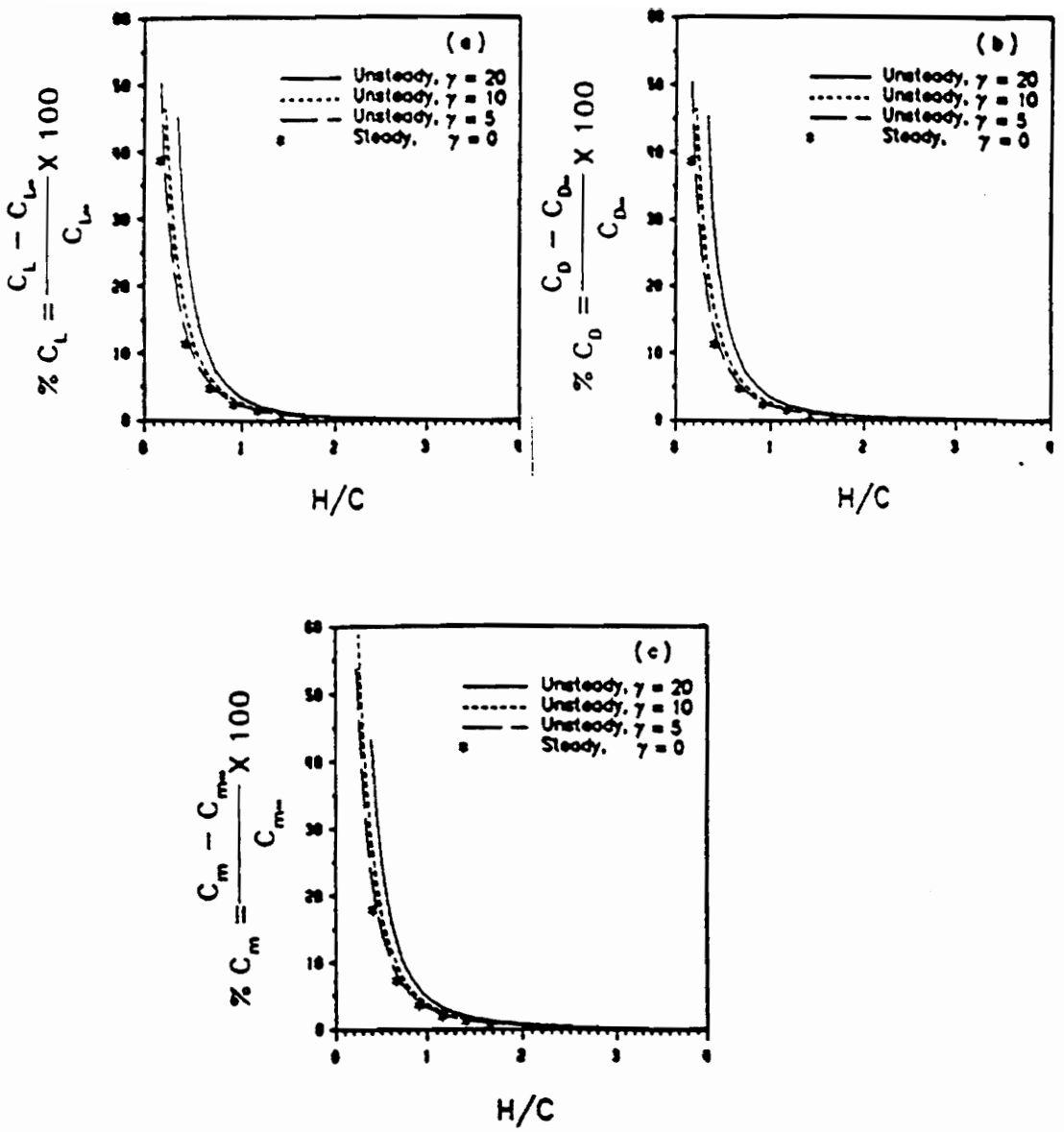

Figure 28. **Computed results for a rectangular wing of unit aspect ratio in ground effect: (a) lift, (b) drag and (c) pitching-moment increments as functions of the height of the trailing edge above the ground for different sink rate (see figure 15 for the definition of γ). $\alpha = 10°$.**

Figure 9: Ground effects plot for 10 degree angle of attack, percent increase in lift vs. height/chord

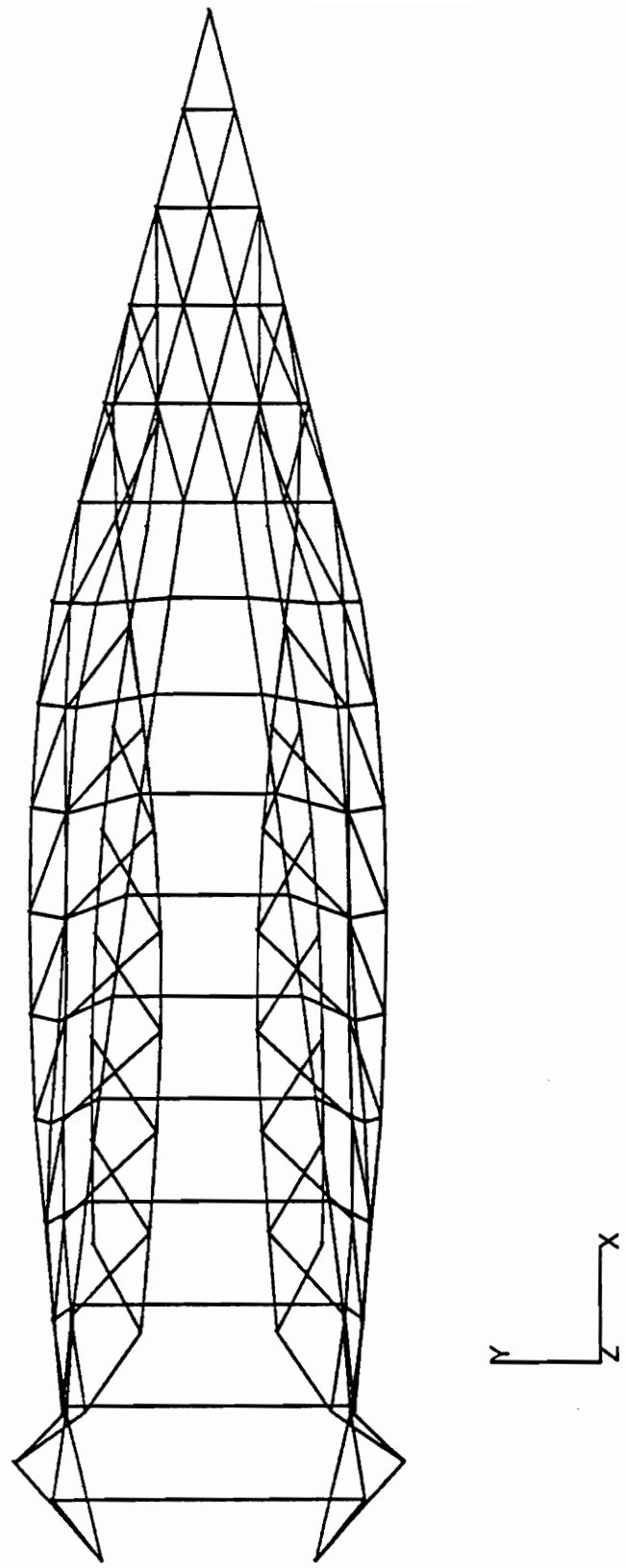

Figure 10a: Top view of a delta wing, height=100 units,
angle of attack=15°, CL= 0.4962

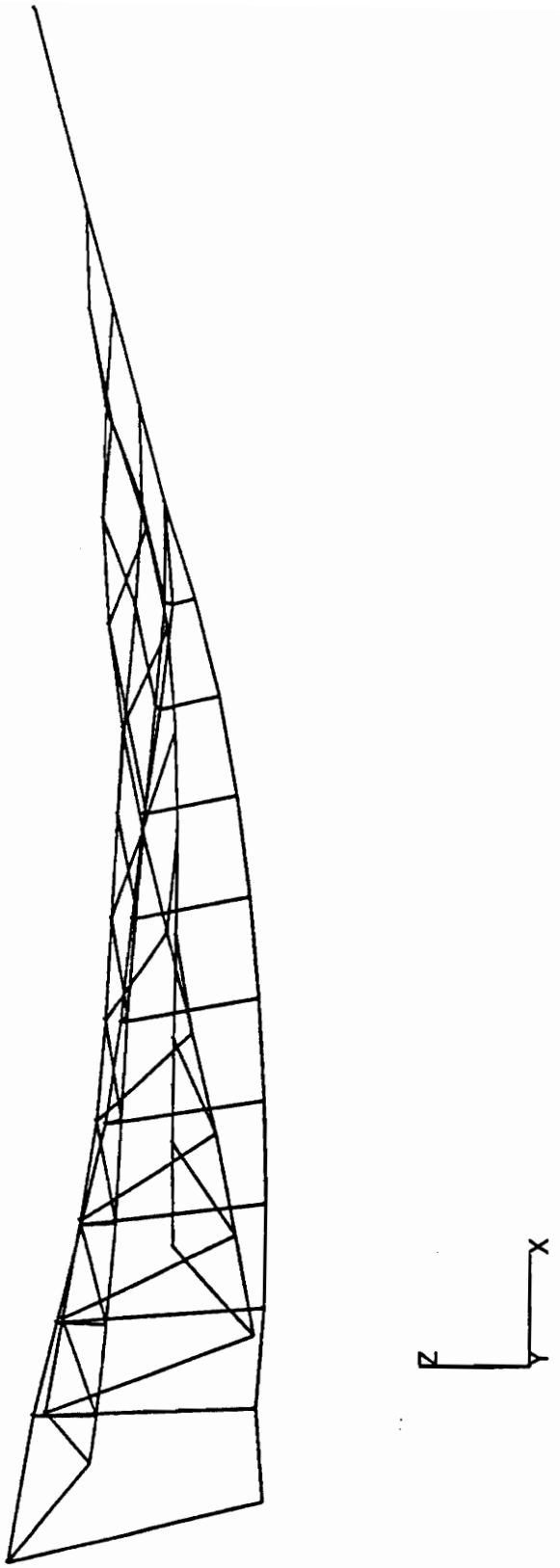

Figure 10b: Side view

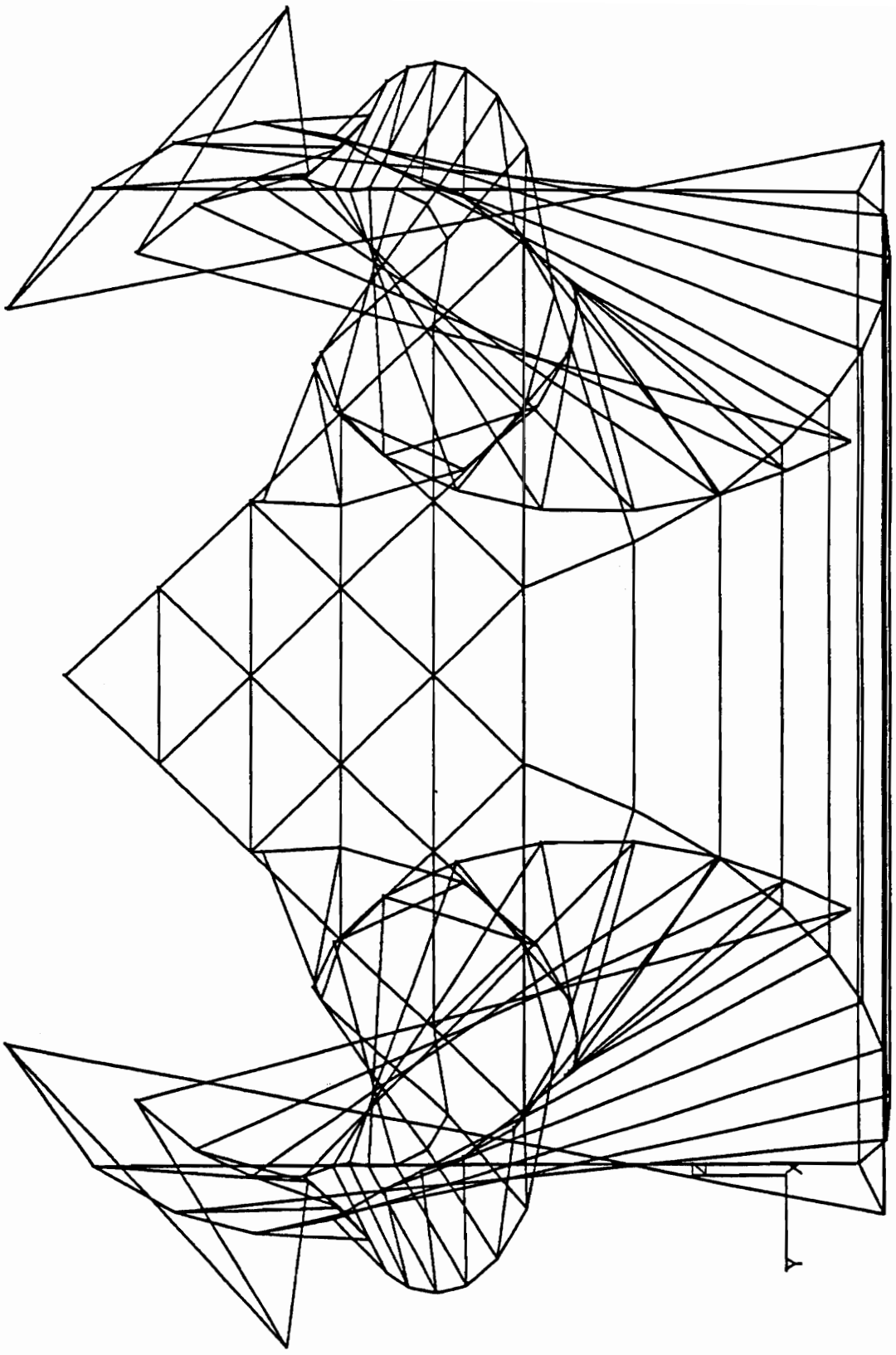

Figure 10c: Back view

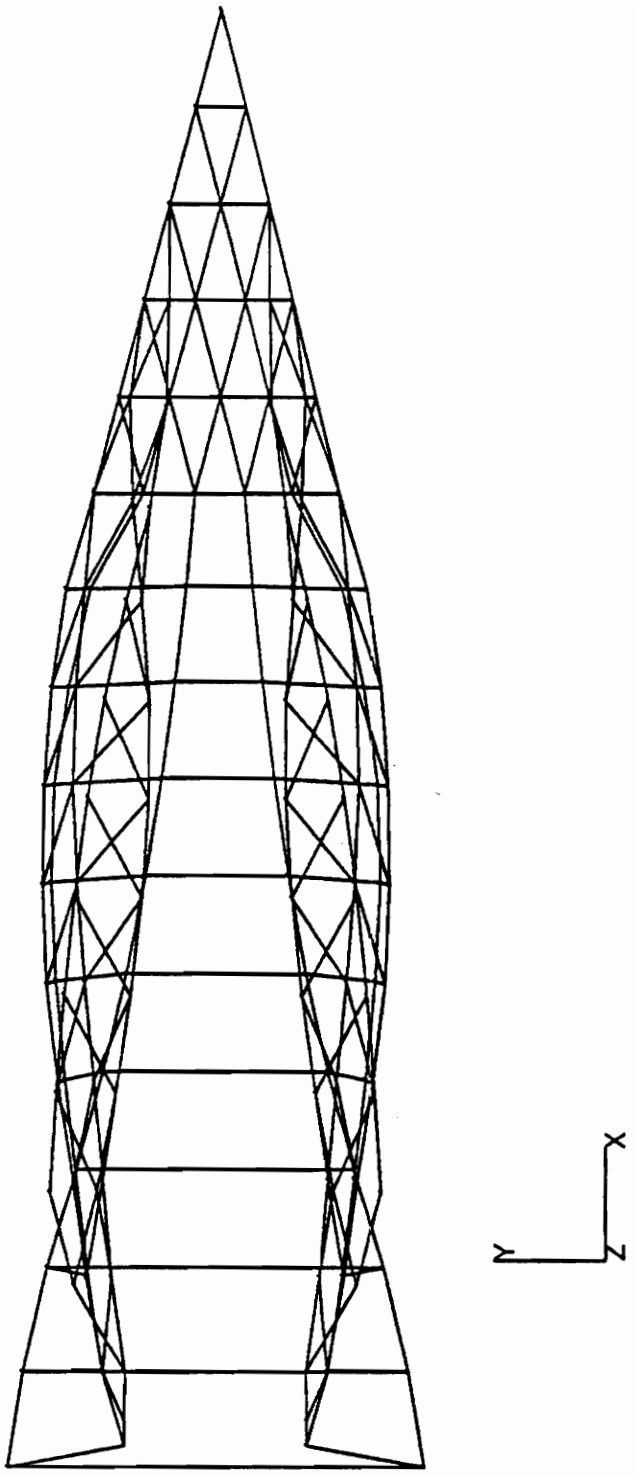

Figure 11a: Top view of a delta wing, height=.4 units, angle of attack= 10°, CL=.4062

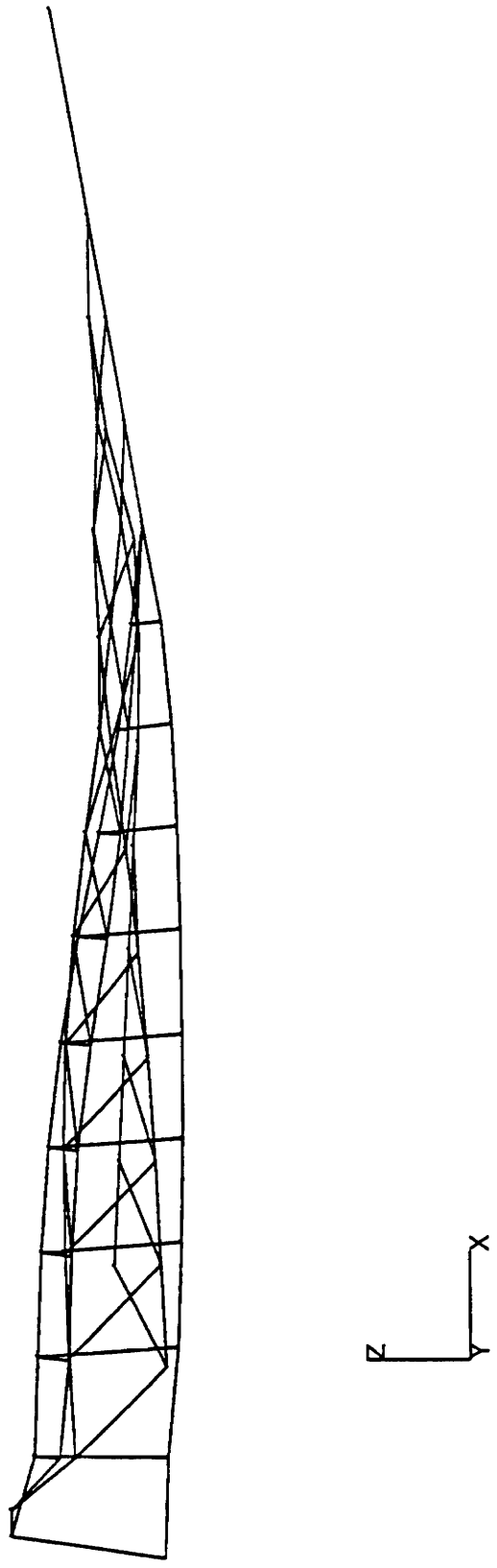

Figure 11b: Side view

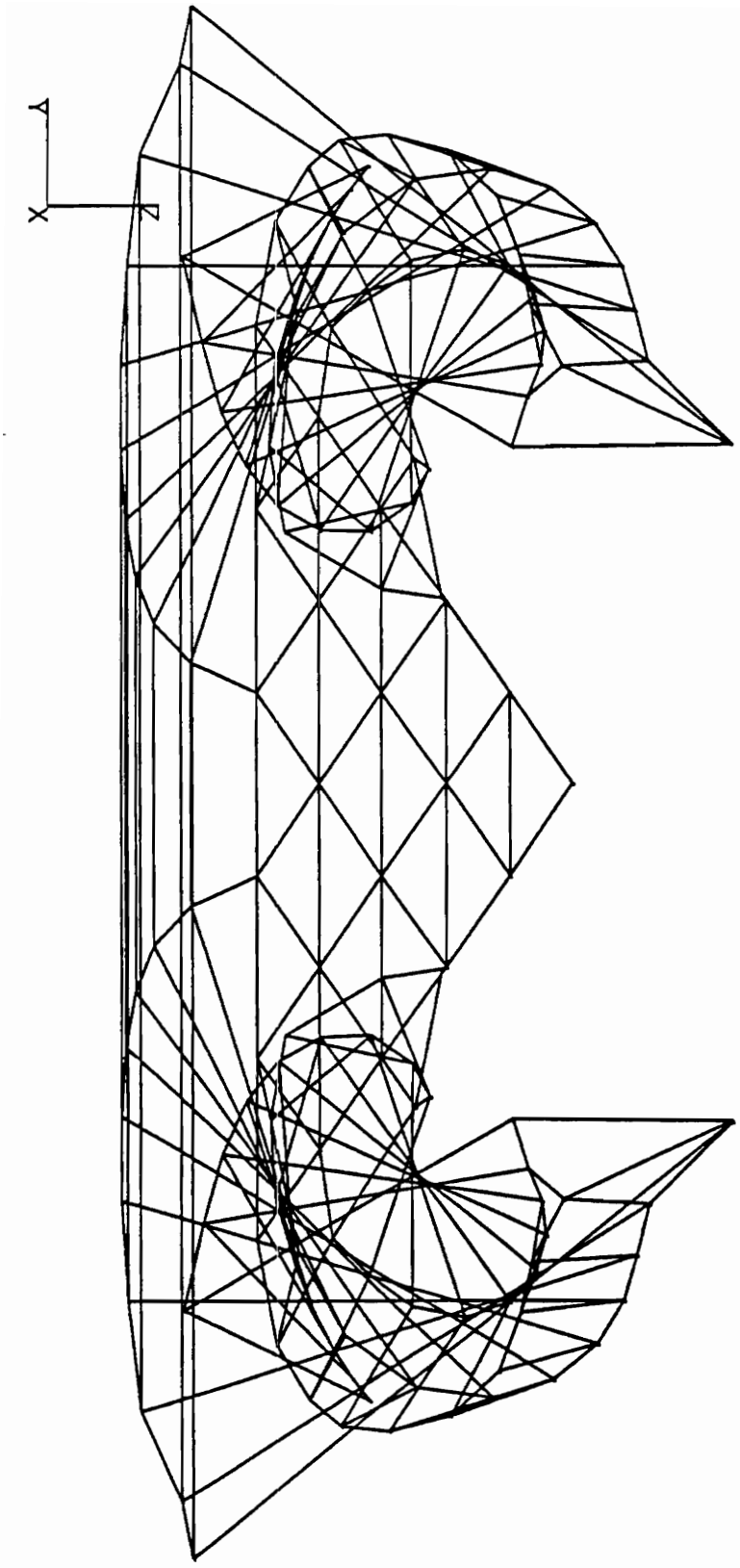

Figure 11c: Back view

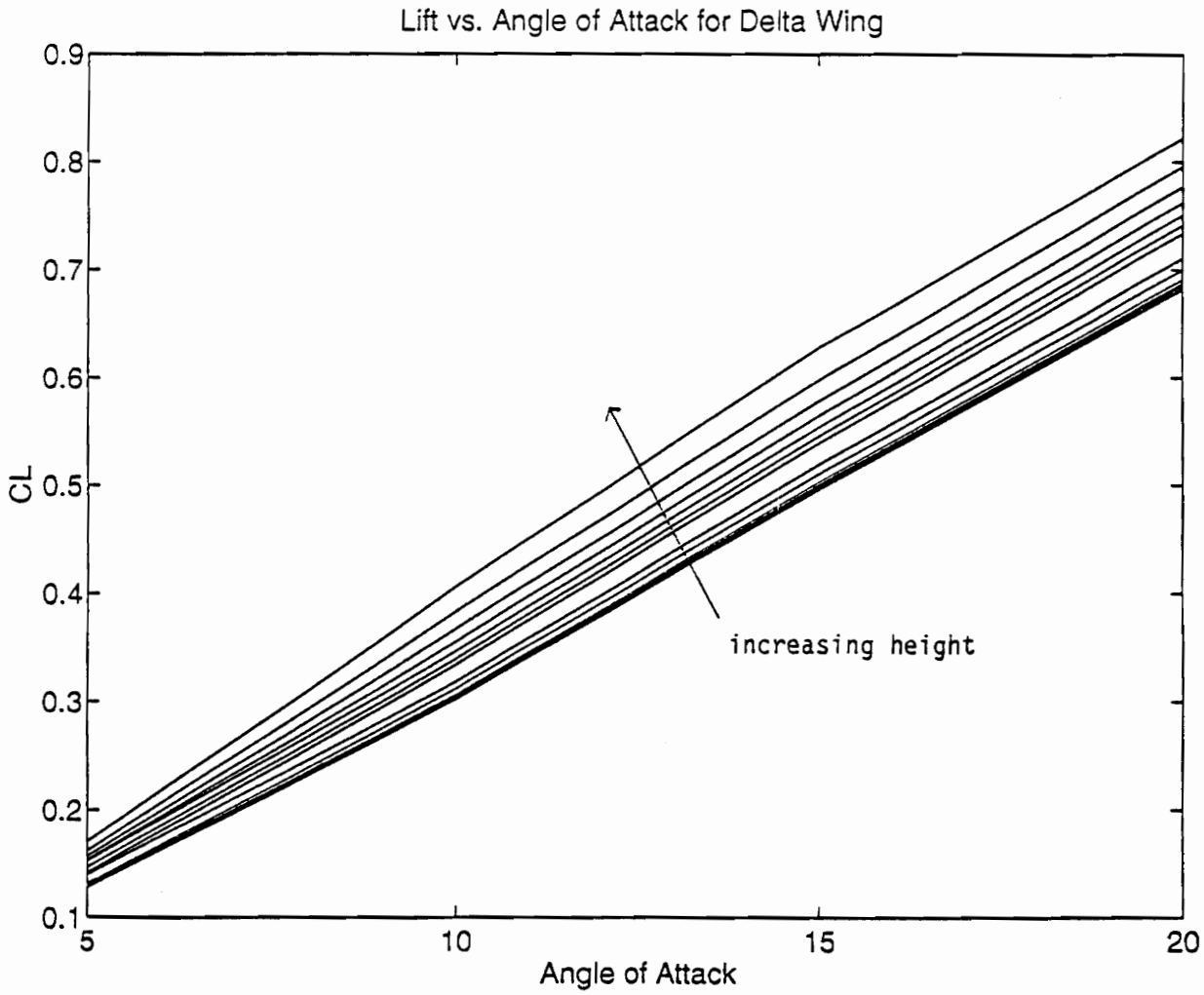

Figure 12: Lift vs. Angle of Attack for Delta Wing

coefficient of lift for the heights varying from 100 units to .2 units. As for the rectangular wing, the lowest line on the graph represents the lift coefficient for the delta wing out of ground effects, and Figure 13 shows the graph plotted by Kandil, Mook, and Nayfeh [1]. The solution for the delta wing converged to a steady value down to about one tenth the chord for the angles of attack tested, and Figure 14 shows the effect of altitude on the coefficient of lift for different angles of attack. A plot of the increase in the coefficient of lift versus the ratio of the height and the chord was also made for the delta wing in Figure 15, and it agrees with the results found by Nuhait [11] in Figure 16. The results for the delta wing are very close to the results predicted by other methods, and the numerical instability appears to have little or no effect on the solution.

3.3 Wingship Planform

The wingship planform was based on the dimensions of the proposed experimental plane called the wingship developed by Aerocon, Inc. Figure 17 is an illustration of the wing at an altitude of 280 units (10 chords) and an angle of attack of 10 degrees. The same wing is shown in Figure 18 at an altitude of 8 units; the wake is more flat and the coefficient of lift is greater at this height. The effect of angle of attack on the lift coefficient at different heights is shown in Figure 19, and the effect of height on the coefficient of lift at various angles of attack is shown as Figure 20.

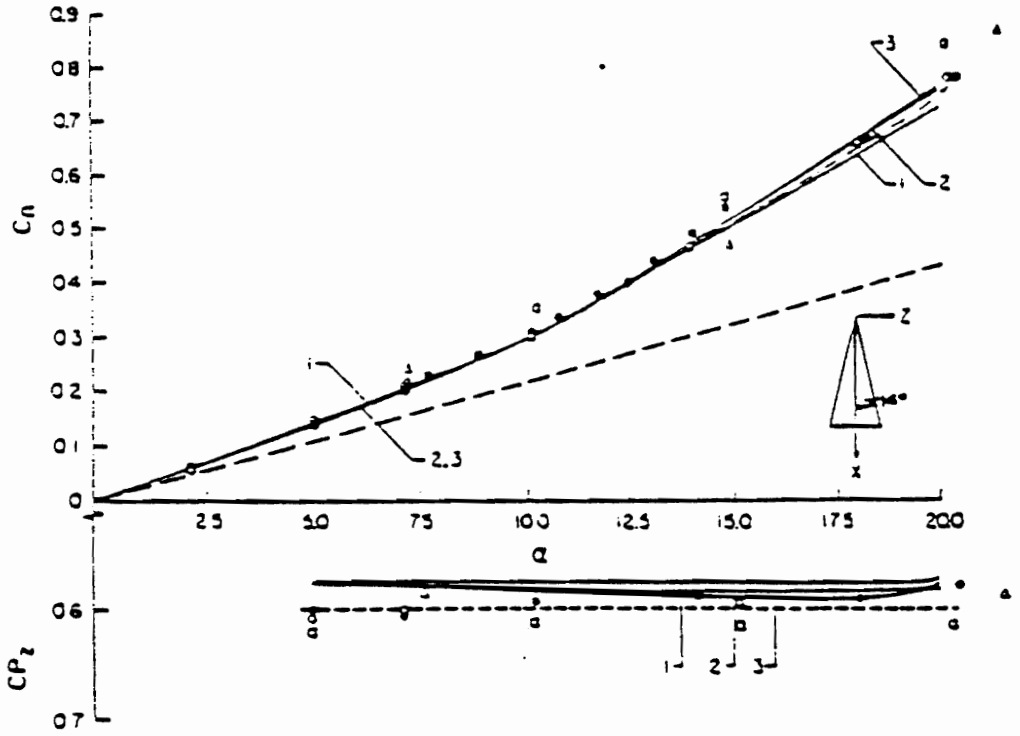

Fig. 11 Normal-force coefficient and position of center of pressure vs angle of attack, $\mathcal{R} = 1$. Numerical results: —— Lawrence;[42] □ Sacks, et al.[24] —— Polhamus;[27] △ Rehbach;[38] ■ Mook and Maddox;[36] —— Present Method. Curve No. 1: 4 × 4 lattice; curve No. 2: 5 × 5 lattice: curve No. 3: 6 × 6 lattice. Experimental results: • Bar-

Figure 13: Normal force coefficients for delta wing plotted by Kandil, Mook, and Nayfeh [1]

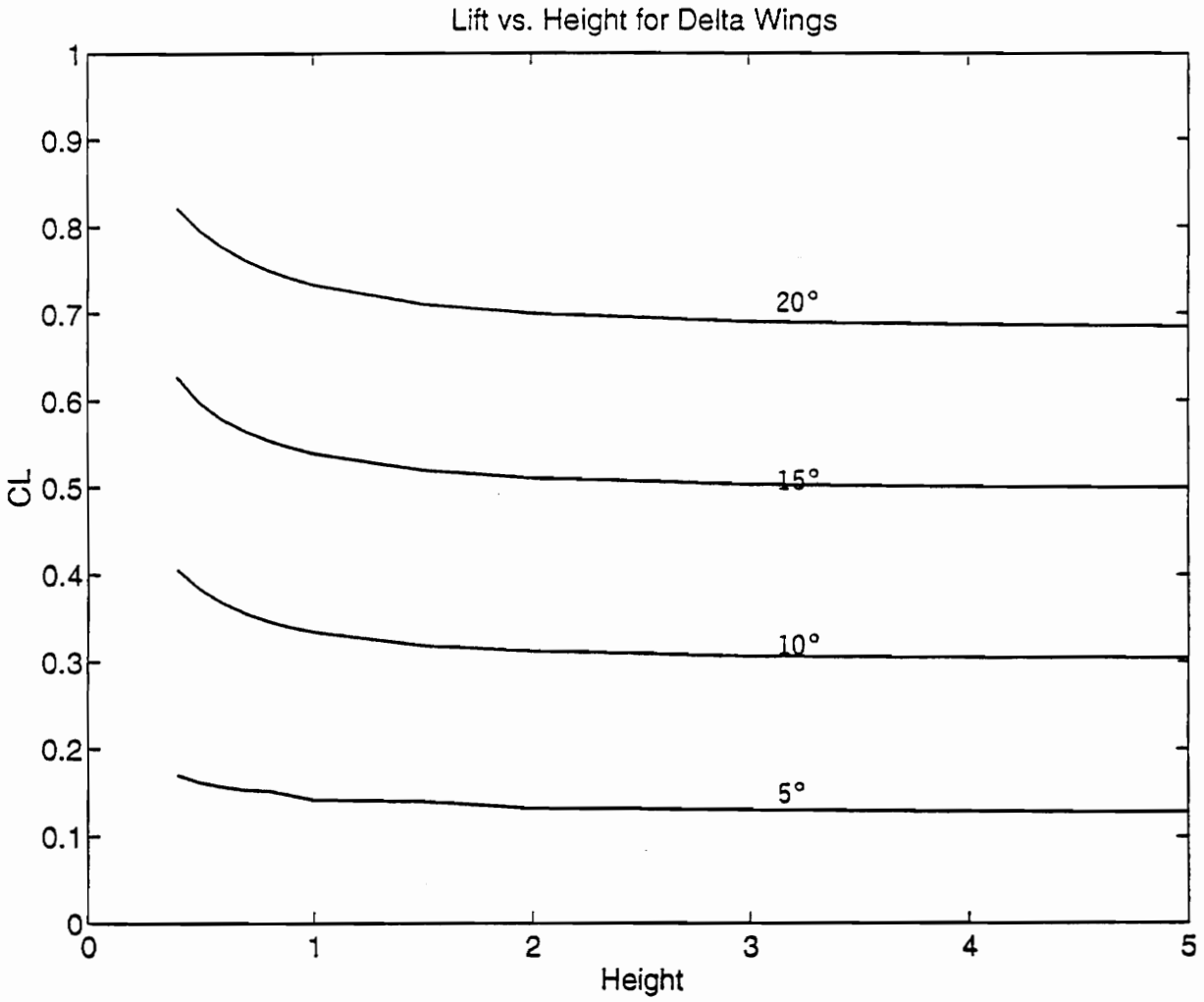

Figure 14: Lift vs. height for delta wings

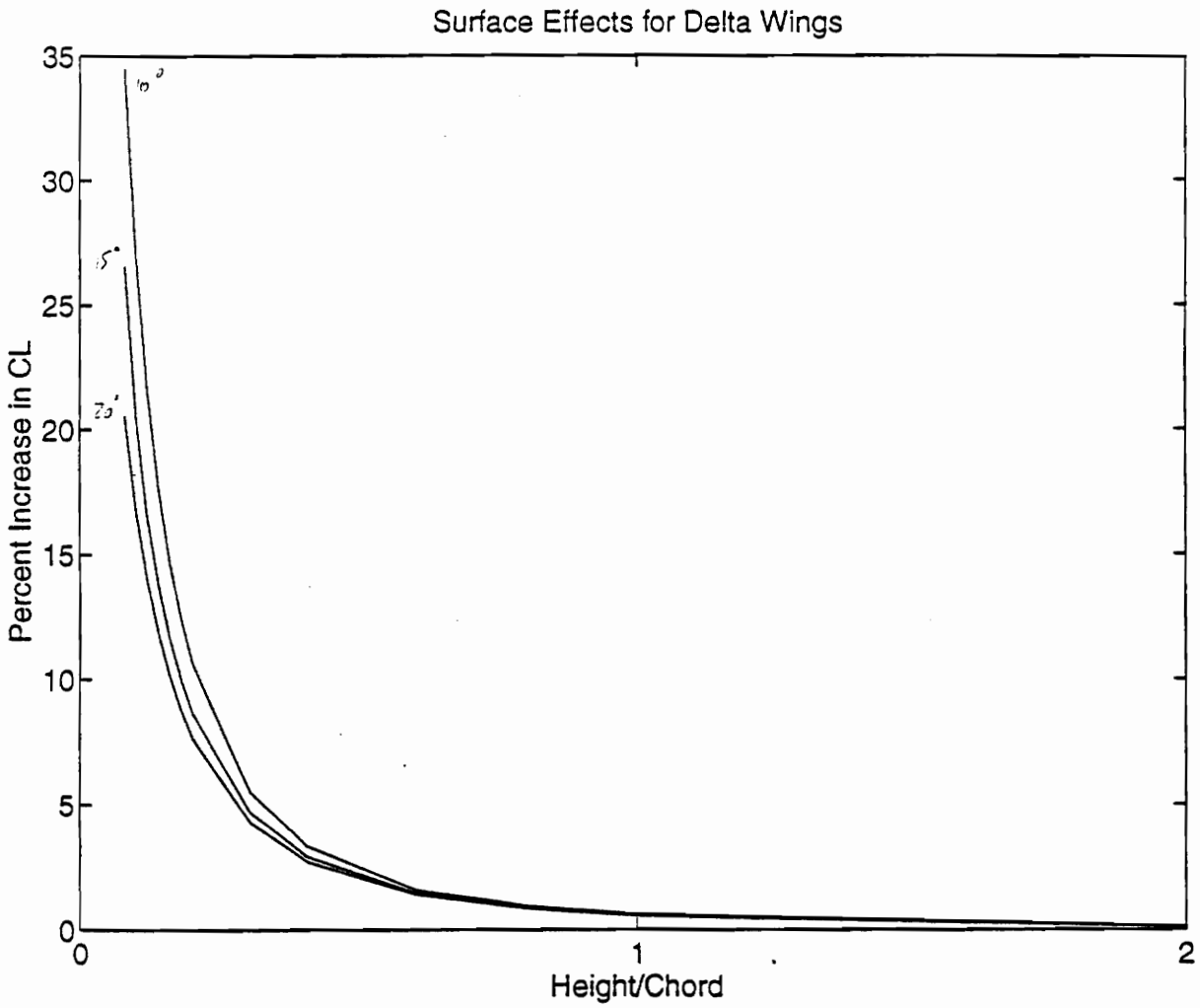

Figure 15: Surface effects for delta wings

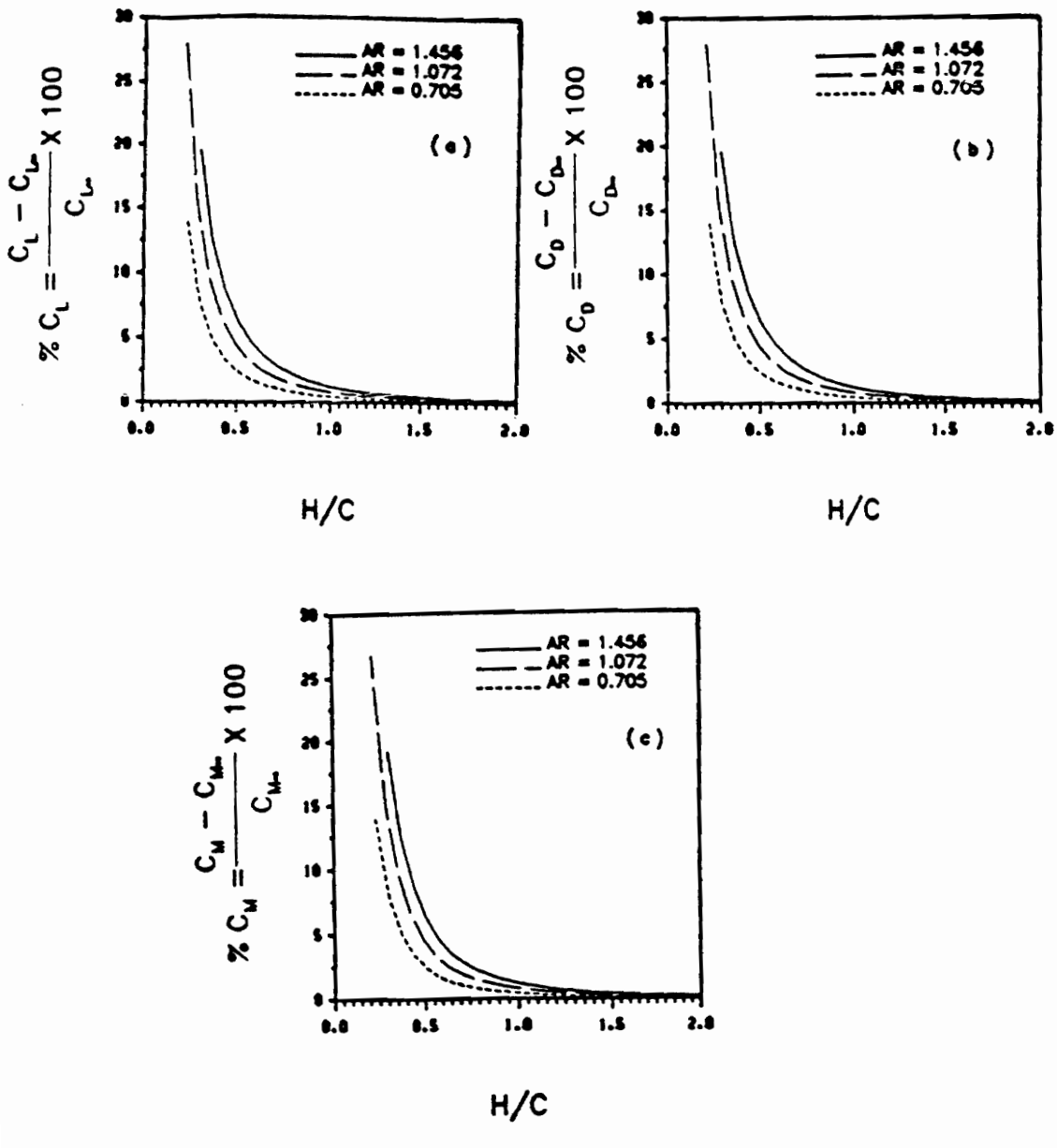

Figure 42. **Computed results for delta wings in ground effect for unsteady flow:** (a) lift, (b) drag and (c) pitching-moment increments at $\alpha = 10°$ and $\gamma = 20°$ as functions of the height of the trailing edge above the ground for different aspect ratios.

Figure 16: Ground effects plot for delta wings by Nuhait[12]

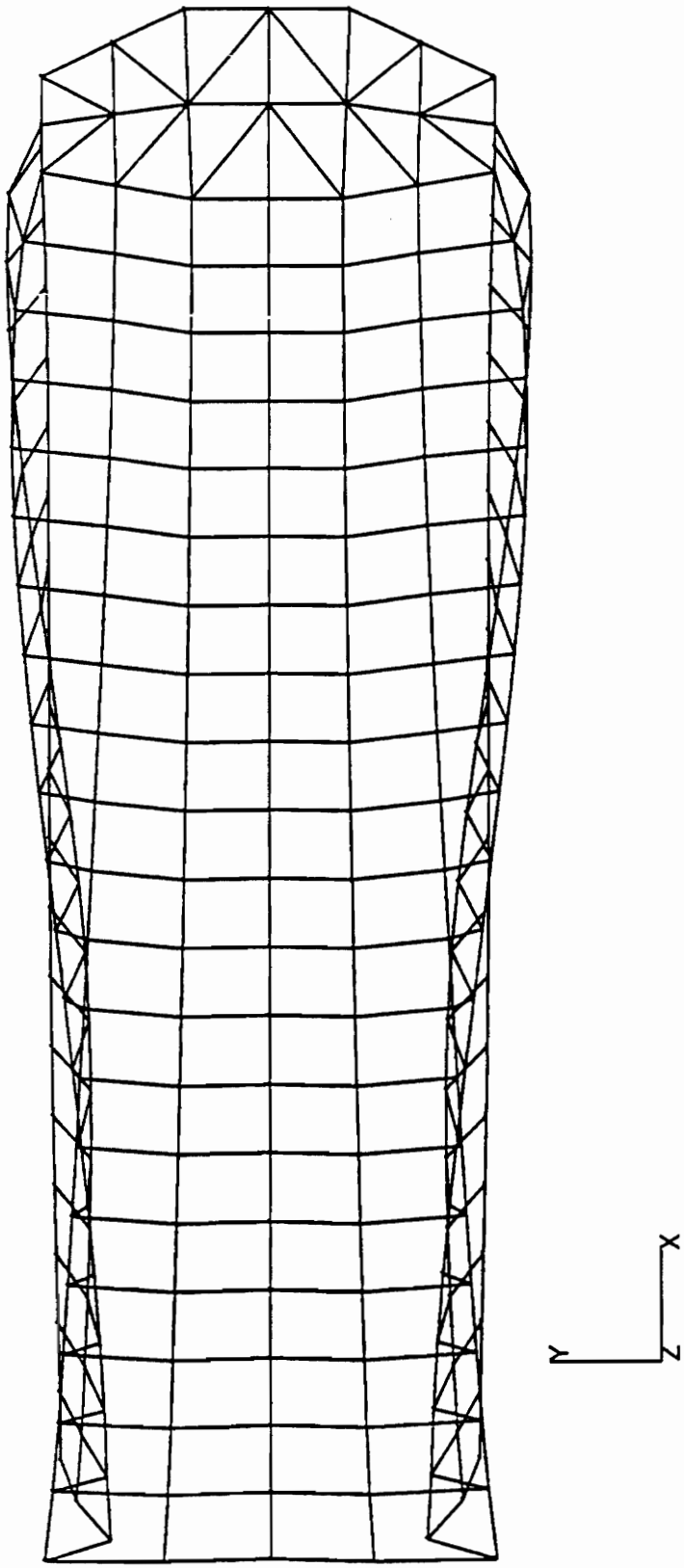

Figure 17a: Wingship planform at height=280 units, angle of attack= 10°, CL= 0.4389

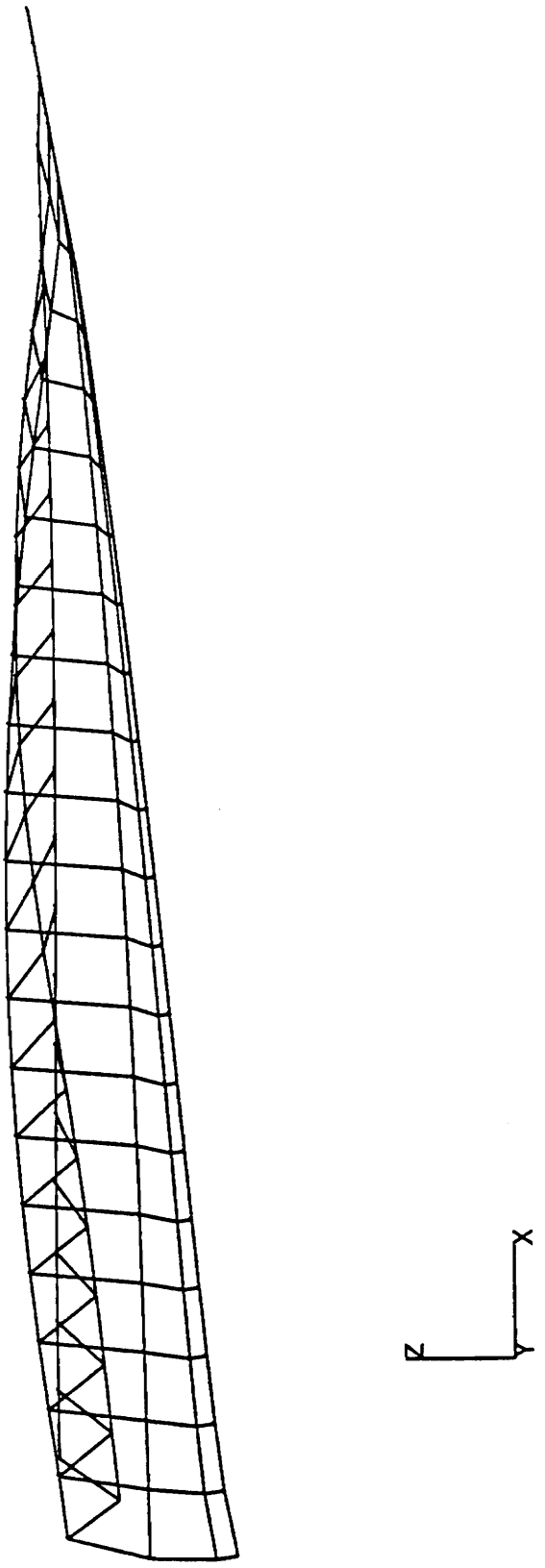

Figure 17b: Side view

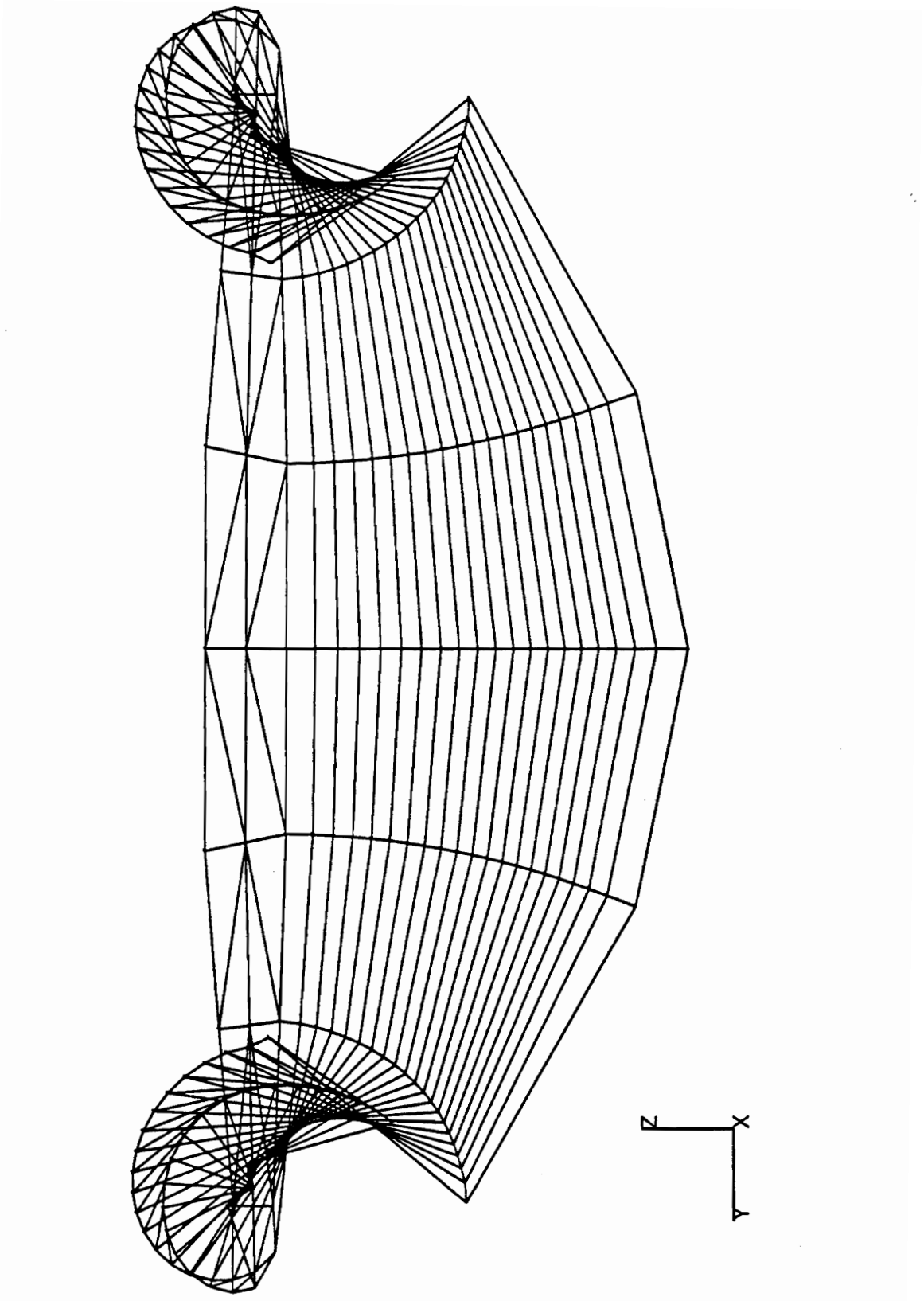

Figure 17c: Back view

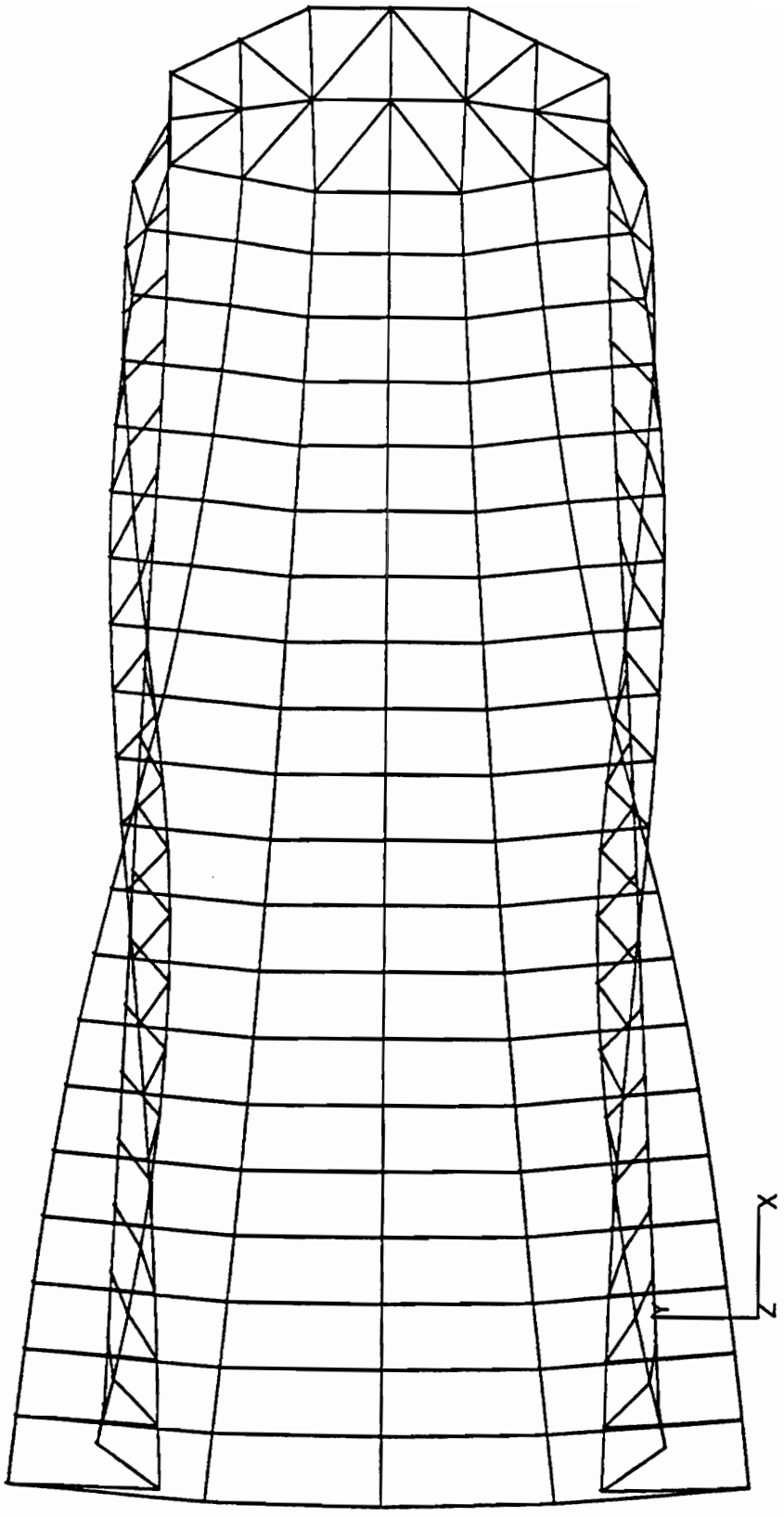

Figure 18a: Top view of the wingship planform at height= 8 units, angle of attack= 10°, CL=0.5623

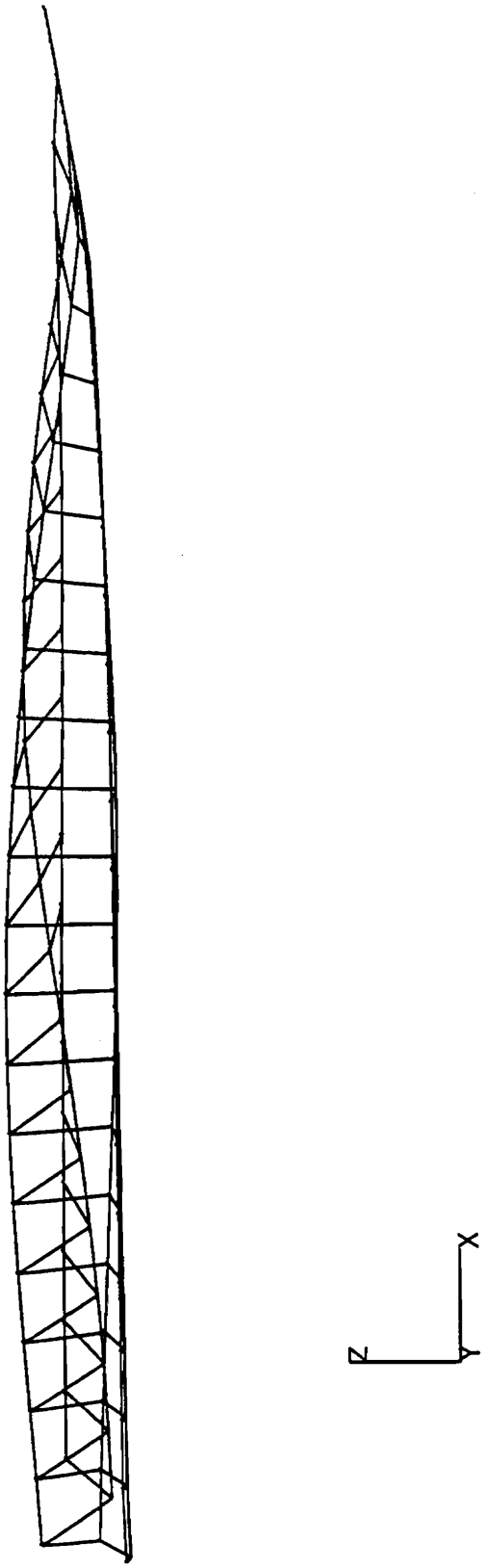

Figure 18b: Side view

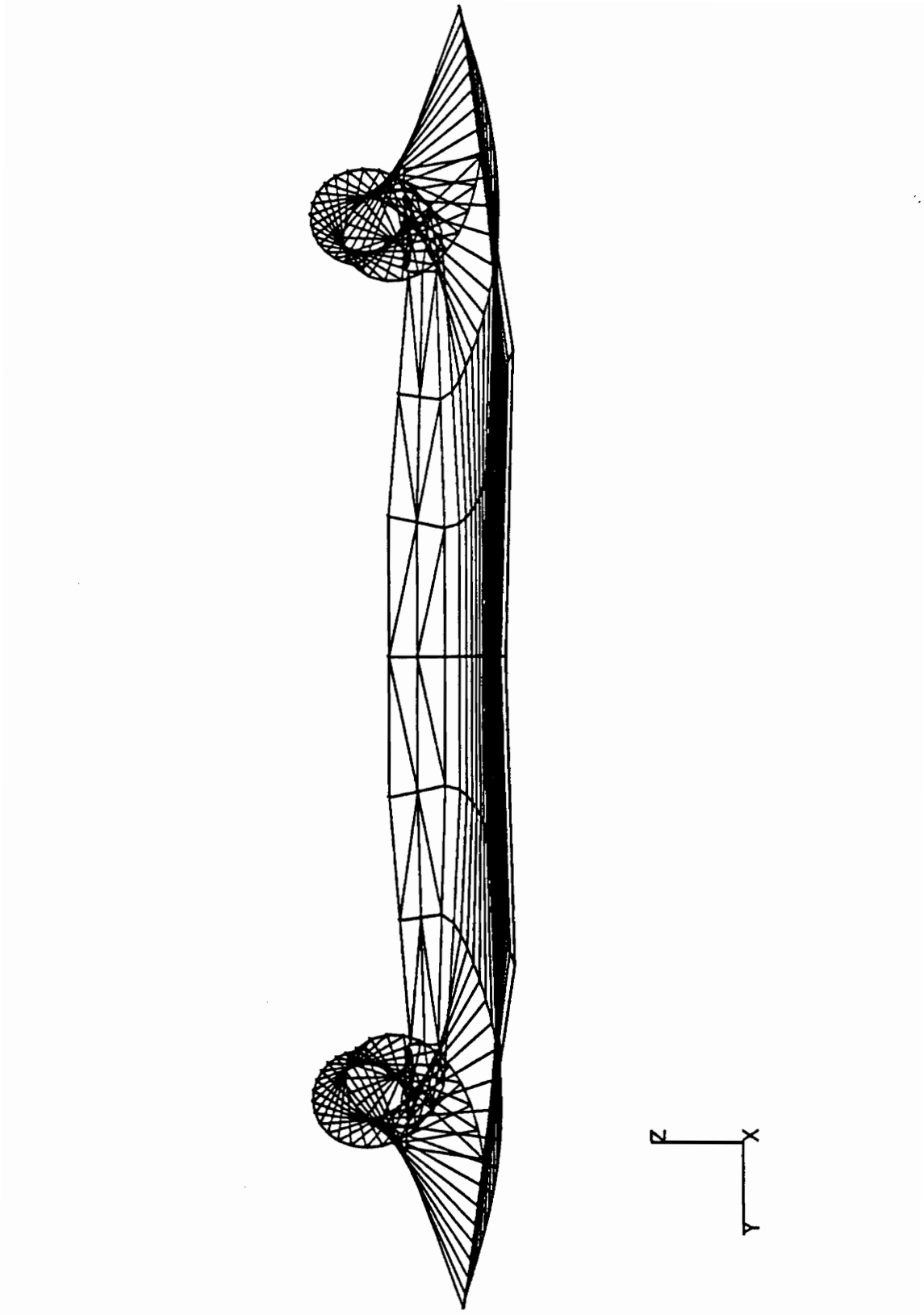

Figure 18c: Back view

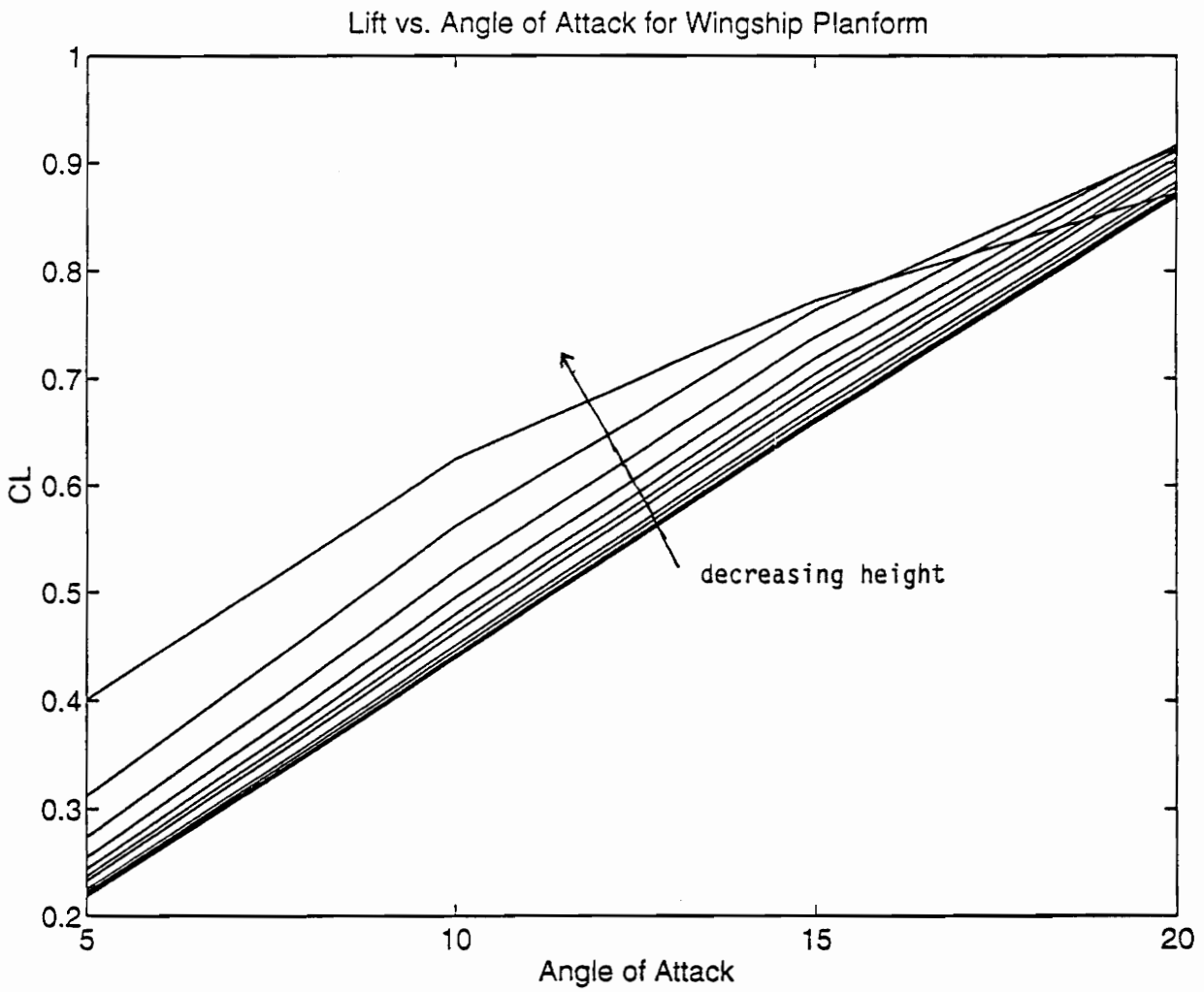

Figure 19: Lift vs. angle of attack for wingship planform

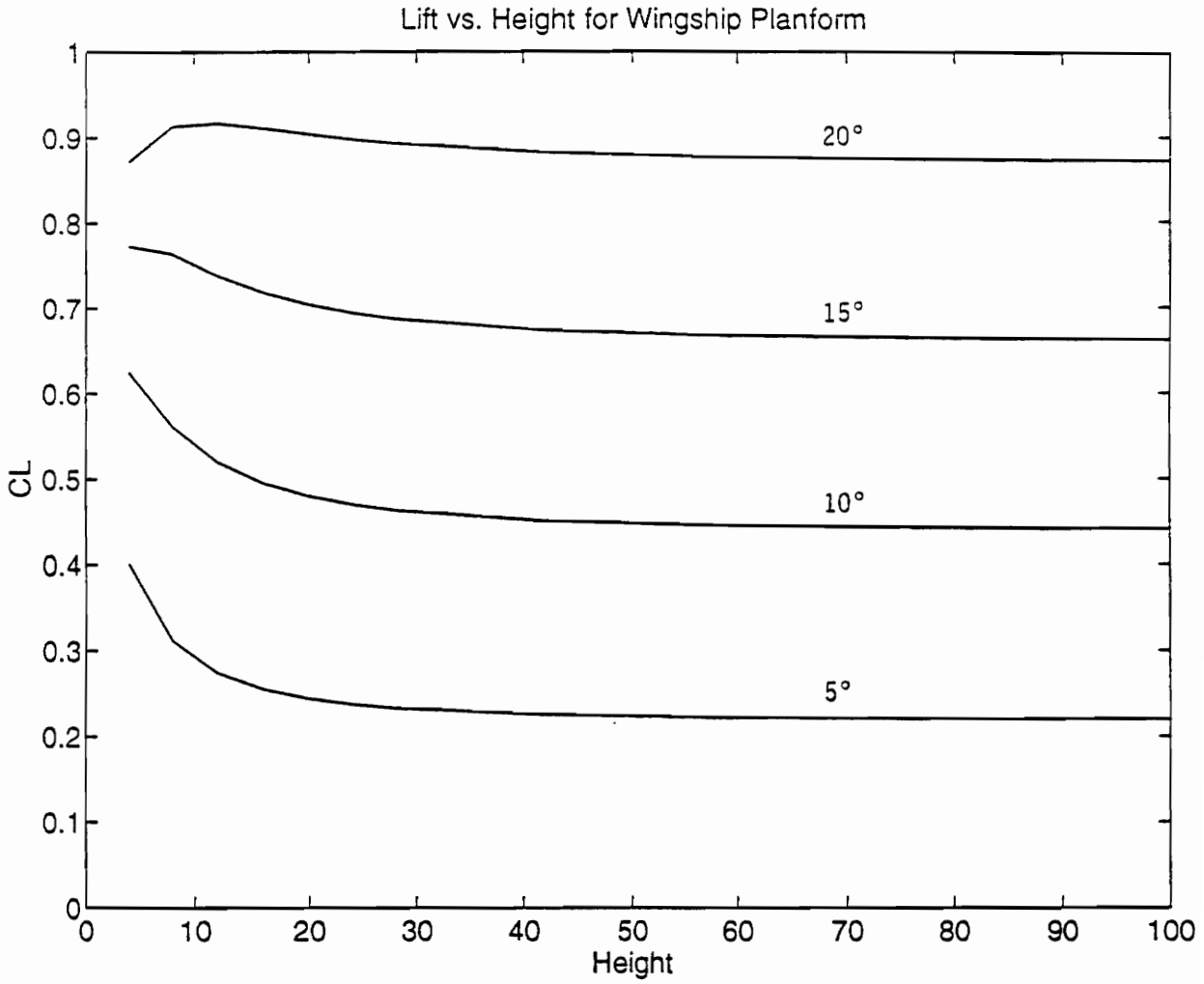

Figure 20: Lift vs. height for wingship planform

These two graphs show a major difference from the earlier graphs of the delta and rectangular wings: at low altitudes and high angles of attack, the coefficient of lift decreases. After experimentation with different parameters, the weighting factor for the Kutta condition was determined to be the cause for this effect. The difference in the wingship planform is that the trailing edge is not straight; when the Kutta condition was removed, the pressure difference was zero at the average x-coordinate of the trailing edge, and the coefficient of lift increased at lower altitudes, as shown in Figure 21. When the pressure difference is forced to approach zero at a trailing edge with different x-coordinates, a decrease in lift is observed at low altitudes and high angles of attack.

Conclusions

The modification of the basic model designed by Mracek, Mook, and Nayfeh has accomplished its goals of making the program accessible to grids created in a finite-element application and including the effects of operating at low altitudes. The rectangular and delta wings compared well with established data for both high and low altitude flight, and some information was gained about the possible behavior of the wingship planform near the surface. The introduction of an arbitrarily shaped body and an additional boundary condition did have some effects on the stability of the

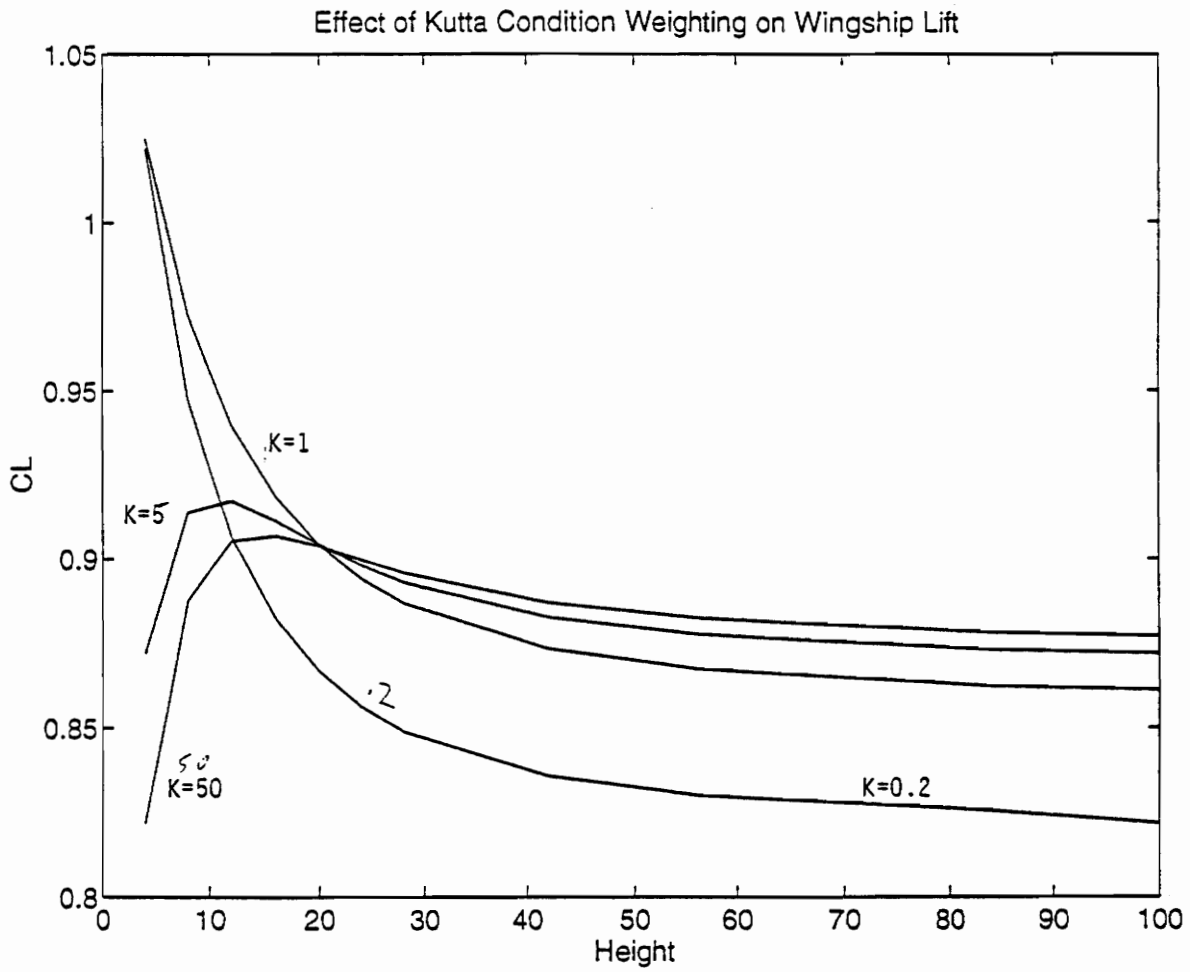

Figure 21: Effect of Kutta Condition Weighting on Wingship Lift

program; the delta wing solution oscillated slightly for some test runs, and it remains to be established that the decrease in lift predicted at high angles of attack for the wingship are reasonable and not just the model becoming inaccurate at the limits of its applicability. Attempts were also made to do unsteady flow prediction by changing the altitude, thus simulating a takeoff or landing; these attempts were unsuccessful because the coefficient of lift would not return to its steady state value after the motion occurred. Further research needs to be done to correct or compensate for the numerical instability noticed in the delta wing experiments, to correct the unsteady model, and to establish the applicability of the model to different bodies input from a finite-element application.

References

[1] Kandil, O. A., D. T. Mook, and A. H. Nayfeh, "Nonlinear Prediction of Aerodynamic Loads on Lifting Surfaces". *J. Aircraft*, Vol.13 No.1, pg 22-27.

[2] Ermolenko, S. D., "Nonlinear Theory of Small Aspect Ration Wings". *Soviet Aeronautics*, Vol. 9, pp. 5-11, March 1966.

[3] Belotserkovskiy, S. M., "Calculation of the Flow around Wings of Arbitrary Planforms in a Wide Range of Angles of Attack", *TT F-12*, 291, May 1969, NASA.

[4] Mook, D. T., and S. A. Maddox, "Extension of a Vortex-Lattice Method to Include the Effects of Leading-Edge Separation", *Journal of Aircraft*, Vol. 11, Feb. 1974, pp. 127-128.

[5] Rehbach, C., "Calculation of Flows around Zero-Thickness Wings with Evalute Vortex Sheets," *TT F-15*, 183, 1973, NASA.

[6] Kim, M. J. *Application of panel methods for subsonic aerodynamics*. Ph. D. Dissertation, Virginia Polytechninc Institute and State University, Blacksburg, VA, 1985.

[7] Kim, M. J. and D. T. Mook, "Application of continuous vorticity panels to general unsteady 2-d lifting flows," *Journal of Aircraft* 23, 464, 1986.

[8] Mook, D. T. and B. Dong, "Application to vortex dynamics to simulations of two-dimensional wakes," *International Symposium on Nonsteady Fluids Dynamics*, Toronto, Canada, June 4-7, 1990.

[9] Mracek, C. P. *A Vortex Panel Method for Potential Flow with Applications to Dynamics and Controls*. Ph. D. Dissertation, Virginia Polytechninc Institute and State University, Blacksburg, VA, 1988.

[10] Elzebda, J. M., D. T. Mook, and A. H. Nayfeh. "Numerical Simulation of Wingships", Intersociety High Performance Marine Vehicle Conference and Exhibit, Arlington, VA, June 24-27, 1992.

[11] Nuhait, A. O. and D. T. Mook, "Numerical Simulation of Feedback Control of Aerodynamic Configurations in Steady and Unsteady Ground Effects," Ph. D. Dissertation, Virginia Polytechninc Institute and State University, Blacksburg, VA, 1988.

[12] Williamson, R. E. and H. F. Trotter, **Multivariable Mathematics**. Prentice-Hall, Inc. New Jersey, 1979, pp. 28-31.

Appendix

The sections of the appendix are as follows: instructions for the use of the model, listings of the programs EDGE.F and P3G.F, an example input file, and listsings of the subroutines SETELE and ISITOV. All of the sections are given on separate pages.

Instructions for use of the Three-dimensional
Vortex-Panel program and its support

There are four fortran programs which can be used to simulate the flow of fluid over a three-dimensional body. There are currently two ways of generating the input data for the main computational program '3dg' written by Dr. Curtis P. Mracek and modified by Scott R. Simmons: a sphere, a rectangular wing, and a delta wing may be generated by the program 'inpforg' also written by Mraceka and modified by Simmons, or the geometry may be generated in a finite element code preprocessor, Patran for example, and the body information written out in the form of a neutral file. This neutral file, along with specific data about the leading and trailing edges generated by the 'edge' program, can be read by the program 'p3g' and converted into the form necessary for reading into the '3dg' program. Subroutines developed by Mike Smith and have been added to the main program to output the body and wake geometries in the neutral file format readable by Patran.

To use the 'inpfor' option for the three shapes defined, simply run the 'inpfor' program, then run the main program. A useful way on this machine (the HP) to run the program in the background is the command:

```
nohup 3d > temp.dat &
```

The results for the body and wake are given as the neutral data files Body.pat and Wake.pat, and can be shown by the following set of commands from the HP:

```
xhost +                          allow writing to the screen
ftp 128.173.160.4                transfer neutral files to tenkay
telnet 128.173.160.4             telnet to tenkay and login
DISPLAY= 128.173.7.191:0         set display to this screen
export DISPLAY                   tell apollo to use this DISPLAY
patran                           execute patran

(From within Patran)
    XWN                          set device mnemonic to X-Windows
    Use menus to load the
    neutral files
    Plot with the ,H,O
    options for Hardcopy.
```

To use the Patran option, get into Patran and develop the geometry for the body. Make sure to note the nodes of the leading and trailing edges as well as the leading node and any corner nodes. Save the geometry data into a neutral file and transfer it to this machine (the HP). Generate the edge data file using the 'edge' program and the information noted above. Use the 'p3g' program to convert the geometry neutral file and the edge data file into the input file for the '3dg' main program, and continue as above to see the results.

```
**********************************************************************
*                                                                    *
*                            Edge                                    *
*                                                                    *
*     This program receives the edge data for an arbitrary           *
*     body and puts it in a form readable by the program p3g.        *
*                                                                    *
**********************************************************************
C
C
      Implicit Integer (A-Z)
      Parameter(N=100)
      Character*20 NAME
      Integer i,NSTR,NLEDGE,NTEDGE,NCOR
      Dimension NL1(N),NL2(N),NT1(N),NT2(N),CN(N),iflag(N)
C
C     Get the file name for the edge data
C
      Write(6,*) 'Input the name for the edge data file: '
      Read(5,*) NAME
      Open(1,file=NAME,status='Unknown')
C
C     Input and write the data
C
      Write(6,*) 'Input the node for the start of the potential'
      Write(6,*) '(Usually node 1): '
      Read(5,*) NSTR
      Write(1,*) 'Node for the start of the potential:'
      Write(1,*) NSTR
      Write(1,*) ' '
C
      Write(6,*) 'Input the number of leading edges: '
      Read(5,*) NLEDGE
      Write(1,*) 'The number of leading edges: '
      Write(1,*) NLEDGE
      Write(1,*) ' '
      if (nledge.eq.0) goto 11
C
      Write(1,*) 'Line segments of the leading edges: '
      Write(6,*) 'Input the pair of nodes describing each leading edge.'
      Write(6,*) 'Separate entries by a comma; put a 1 if it is a non-'
      WRITE(6,*) 'convecting edge, and a 0 if it convects.'
      Do 10 i=1,NLEDGE
        Write(6,*) 'Edge #',i,': '
        Read(5,*) NL1(i),NL2(i),iflag(i)
        Write(1,*) NL1(i),NL2(i),iflag(i)
10      Continue
      Write(1,*) ' '
C
11    Write(6,*) 'Input the number of trailing edges: '
      Read(5,*) NTEDGE
      Write(1,*) 'The number of trailing edges: '
      Write(1,*) NTEDGE
```

```
      Write(1,*) ' '
      if (ntedge.eq.0) goto 21
C
      Write(1,*) 'Line segments of the trailing edges: '
      Write(6,*) 'Input the nodes describing each trailing edge;'
      Do 20 i=1,NTEDGE
        Write(6,*) 'Edge #',i,': '
        Read(5,*) NT1(i),NT2(i)
        Write(1,*) NT1(i),NT2(i)
20      Continue
      Write(1,*) ' '
C
21    Write(6,*) 'Input the number of corners: '
      Read(5,*) NCOR
      Write(1,*) 'The number of corners: '
      Write(1,*) NCOR
      Write(1,*) ' '
      if (ncor.eq.0) goto 31
C
      Write(1,*) 'Corner nodes: '
      Write(6,*) 'Input the corner nodes;'
      Do 30 i=1,NCOR
        Write(6,*) 'Node #',i,': '
        Read(5,*) CN(i)
        Write(1,*) CN(i)
30    Continue
31    Stop
      End
```

```
*************************************************************************
*                                                                       *
*                        **************                                 *
*                        *     p3     *                                 *
*                        **************                                 *
*                                                                       *
*       THIS PROGRAM GENERATES THE INPUT DATA FOR VARIOUS SHAPES        *
*       DEFINED IN A PATRAN NEUTRAL FILE.  THE OUTPUT IS A              *
*       DATA FILE COMPATIBLE WITH THE THREE-DIMENSIONAL VORTEX          *
*       PROGRAM.                                                        *
*                                                                       *
*************************************************************************
      IMPLICIT DOUBLE PRECISION(H,X-Z)
      Implicit Integer(A-G,I-W)
      PARAMETER (N=500,M=100)
      COMMON/P3/ IDEN(N),XX(N),YY(N),ZZ(N),N1(N),N2(N),N3(N),NODE,NELE
      COMMON/EDGE1/ NL1(M),NL2(M),NT1(M),NT2(M),CN(M),iflag(N)
      COMMON/EDGE2/ NSTR,NLEDGE,NTEDGE,NCOR
      COMMON/WRITE/ LEN(M),TEN(M),EN(M),HEIGHT
C
C     Decide height of body above ground
C
      WRITE(6,*) 'Input the height of the wing above ground:'
      READ(5,*) HEIGHT
C
C     Get the input data from the Patran Neutral File
C
      CALL READER
C
C     Get the input from the Edge Input File
C
      CALL EDGE
C
C     Write the input file 'INPUT.DAT' for the 3d program
C
      CALL WRITER
C
C     End Program
C
      STOP
      END
C
C
*************************************************************************
*                                                                       *
*                          Subroutine Reader                            *
*                                                                       *
*************************************************************************
*                                                                       *
*     Node:  number of nodes                                            *
*     Nele:  number of elements                                         *
*     xx,yy,zz:  coordinates of nodes                                   *
*     n1,n2,n3:  nodes of the triangle                                  *
```

```
*                                                                      *
************************************************************************
C
C
      Subroutine Reader
      IMPLICIT DOUBLE PRECISION(H,X-Z)
      Implicit Integer(A-G,I-W)
      PARAMETER (N=500,M=100)
      COMMON/P3/ IDEN(N),XX(N),YY(N),ZZ(N),N1(N),N2(N),N3(N),NODE,NELE
      Character*70 title
   51 FORMAT(A70)
   52 FORMAT(3F16.9)
C
C     Read the patran neutral file
C
      write(6,*) 'Enter the name of the Patran neutral file: '
      read(5,*) title
      open(1,file=title,status='unknown')
      read(1,51) title
      read(1,51) title
      read(1,*) i1,i2,i3,i4,node,nele,i7,i8,i9
      read(1,51) title
      do 10 i=1,node
        read(1,51) title
        read(1,52) xx(i),yy(i),zz(i)
        read(1,51) title
10      continue
      do 20 i=1,nele
        read(1,51) title
        read(1,51) title
        read(1,*) n1(i),n2(i),n3(i)
20      continue
      close(1)
      return
      end
C
C
************************************************************************
*                                                                      *
*                     Subroutine Edge                                  *
*                                                                      *
************************************************************************
*                                                                      *
*     Iden:  flag for development of potential paths                   *
*     N11,N12:  nodes of a leading egde segment                        *
*     Nt1,Nt2:  nodes of a trailing edge segment                       *
*     iflag:  for edges, 0 if convecting, 1 if non-convecting          *
*     Cn:  corner nodes                                                *
*     Nstr:  node for the start of potential flow                      *
*     Nledge:  number of leading edge segments                         *
*     Ntedge:  number of trailing edge segments                        *
*                                                                      *
************************************************************************
```

```
C
C
      Subroutine Edge
      IMPLICIT DOUBLE PRECISION(H,X-Z)
      Implicit Integer(A-G,I-W)
      PARAMETER (N=500,M=100)
      COMMON/P3/ IDEN(N),XX(N),YY(N),ZZ(N),N1(N),N2(N),N3(N),NODE,NELE
      COMMON/EDGE1/ NL1(M),NL2(M),NT1(M),NT2(M),CN(M),iflag(N)
      COMMON/EDGE2/ NSTR,NLEDGE,NTEDGE,NCOR
      Character*70 title
   51 FORMAT(A70)
C
C     Read the Edge file
C
      write(6,*) 'Enter the name of the Edge data file: '
      read(5,51) title
      open(2,file=title,status='unknown')
      read(2,51) title
      read(2,*) NSTR
      read(2,51) title
      read(2,51) title
      read(2,*) NLEDGE
      read(2,51) title
      read(2,51) title
      if (nledge.eq.0) goto 31
      do 30 i=1,NLEDGE
        read(2,*) NL1(i),NL2(i),iflag(i)
30      continue
      read(2,51) title
      read(2,51) title
31    read(2,*) NTEDGE
      read(2,51) title
      read(2,51) title
      if (ntedge.eq.0) goto 41
      do 40 i=1,NTEDGE
        read(2,*) NT1(i),NT2(i)
40      continue
      read(2,51) title
      read(2,51) title
41    read(2,*) NCOR
      if (ncor.eq.0) goto 55
      read(2,51) title
      read(2,51) title
      do 50,i=1,NCOR
        read(2,*) CN(i)
50      continue
55    close(2)
      return
      end
C
C
*********************************************************************
*                                                                   *
```

```
*                      Subroutine Writer                                *
*                                                                       *
*************************************************************************
*                                                                       *
*     Len:  leading edge nodes                                          *
*     Ten:  trailing edge nodes                                         *
*     En:   edge nodes                                                  *
*     EC:   edge cores                                                  *
*                                                                       *
*************************************************************************
C
C
      Subroutine Writer
      IMPLICIT DOUBLE PRECISION(H,X-Z)
      Implicit Integer(A-G,I-W)
      PARAMETER (N=500,M=100)
      COMMON/P3/ IDEN(N),XX(N),YY(N),ZZ(N),N1(N),N2(N),N3(N),NODE,NELE
      COMMON/EDGE1/ NL1(M),NL2(M),NT1(M),NT2(M),CN(M),iflag(N)
      COMMON/EDGE2/ NSTR,NLEDGE,NTEDGE,NCOR
      COMMON/WRITE/ LEN(M),TEN(M),EN(M),HEIGHT
      INTEGER EC1(M),EC2(M)
      Character*70 title
   51 FORMAT(A70)
   52 FORMAT(3I7)
   53 FORMAT(3F20.15)
      open(3,file='INPUT.DAT',status='unknown')
C
C     Input Title and initial information
C
      WRITE(6,*) '   INPUT THE FIRST LINE OF THE TITLE'
      READ(5,51) TITLE
      WRITE(3,51) TITLE
      WRITE(6,*) '   INPUT THE SECOND LINE OF THE TITLE'
      READ(5,51) TITLE
      WRITE(3,51) TITLE
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='CONTINUATION FROM ANOTHER RUN'
      WRITE(3,51) TITLE
      TITLE='   N'
      WRITE(3,51) TITLE
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='IS ITERATION REQUIRED'
      WRITE(3,51) TITLE
      TITLE='   N'
      WRITE(3,51) TITLE
      TITLE=' '
      WRITE(3,51) TITLE
C
C     Write Geometry from Patran file
C
      TITLE='THE NUMBER OF TRIANGULAR ELEMENTS'
```

```
      WRITE(3,51) TITLE
      WRITE(3,*) nele
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE ORDERING OF THE TRIANGULAR ELEMENTS'
      WRITE(3,51) TITLE
      do 60 i=1,nele
        WRITE(3,52) N1(i),N2(i),N3(i)
60      continue
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NUMBER OF NODES'
      WRITE(3,51) TITLE
      WRITE(3,*) NODE
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE LOCATIONS OF THE NODES IN THE BODY FRAME'
      WRITE(3,51) TITLE
      do 70 i=1,node
        WRITE(3,53) XX(i),YY(i),ZZ(i)
70      continue
      TITLE=' '
      WRITE(3,51) TITLE
C
C     Calculate and write data from the Edge data file
C
      TITLE='THE NUMBER OF EDGE CORES'
      WRITE(3,51) TITLE
      NEDGE=NLEDGE+NTEDGE
      WRITE(3,*) NEDGE
      if(nedge.eq.0) goto 91
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NODES OF THE EDGE CORES'
      WRITE(3,51) TITLE
      nlp1=NLEDGE+1
      etotal=nledge+ntedge
      Do 80 i=1,NLEDGE
        EC1(i)=NL1(i)
        EC2(i)=NL2(i)
80      Continue
      Do 81 i=nlp1,etotal
        EC1(i)=NT1(i-NLEDGE)
        EC2(i)=NT2(i-NLEDGE)
81      Continue
      call rorder(etotal,EC1,EC2,iflag)
      Do 82 i=1,etotal
        Write(3,*) EC1(i),EC2(i)
82      continue
91    TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NUMBER OF CONVECTING EDGE CORES'
      WRITE(3,51) TITLE
```

```
      ncec=nedge
      do 100 i=1,etotal
        ncec=ncec-iflag(i)
100     continue
      WRITE(3,*) ncec
      p=0
      l=0
      if(ncec.eq.0) goto 275
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NODES OF THE CONVECTING EDGE CORES'
      WRITE(3,51) TITLE
      Do 120 i=1,etotal
        if(iflag(i).eq.1) goto 120
        Write(3,*) EC1(i),EC2(i),nstr
120     Continue
C
C     Convert the segment information to node information
C
      Do 150 i=1,NTEDGE+1
        TEN(i)=0
150     Continue
      Do 155 i=1,etotal+2
        EN(i)=0
155     Continue
      k=0
      Do 160 i=1,etotal
        Do 170 j=1,etotal+1
          if (EC1(i).eq.EN(j)) goto 160
170       continue
        if(iflag(i).eq.1) goto 160
        k=k+1
        EN(k)=EC1(i)
160     Continue
      Do 180 i=1,etotal
        Do 190 j=1,etotal+1
          if (EC2(i).eq.EN(j)) goto 180
190       continue
        if(iflag(i).eq.1) goto 180
        k=k+1
        EN(k)=EC2(i)
180     Continue
      l=0
      Do 200 i=1,NTEDGE
        Do 210 j=1,NTEDGE+1
          if (NT1(i).eq.TEN(j)) goto 200
210       continue
        l=l+1
        TEN(l)=Nt1(i)
200     Continue
      Do 220 i=1,NTEDGE
        Do 230 j=1,NTEDGE+1
          if (NT2(i).eq.TEN(j)) goto 220
```

```
230       continue
          l=l+1
          TEN(l)=Nt2(i)
220       Continue
C
C     Organize Edge Nodes
C
      call order(k,EN)
      call order(l,TEN)
C
C     Continue with output
C
275   TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NUMBER OF STARTING WAKE POSITIONS'
      WRITE(3,51) TITLE
      WRITE(3,*) k
      if(k.eq.0) goto 281
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NUMBER OF THE NODE AND ITS START'
      WRITE(3,51) TITLE
      Do 280 i=1,k
        write(3,*) EN(i),XX(EN(i)),YY(EN(i)),ZZ(EN(i))
280     continue
281   TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NUMBER OF NODES FOR THE KUTTA CONDITION'
      WRITE(3,51) TITLE
      WRITE(3,*) l
      if(l.eq.0) goto 291
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NODES WHERE THE KUTTA IS IMPOSED'
      WRITE(3,51) TITLE
      X=50.0D0
      Do 290 i=1,l
        Write(3,*) TEN(i),X
290     Continue
291   TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NUMBER OF CORNERS'
      WRITE(3,51) TITLE
      WRITE(3,*) NCOR
      if(ncor.eq.0) goto 301
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE CORNER NODES ARE'
      WRITE(3,51) TITLE
      call order(ncor,CN)
      Do 300 i=1,NCOR
        WRITE(3,*) CN(i)
300     continue
```

```
301   TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE NUMBER OF STARTING LOCATIONS FOR POT'
      WRITE(3,51) TITLE
      WRITE(3,*) 1
      TITLE=' '
      WRITE(3,51) TITLE
C
C     Get the node for the starting of the potential
C
      TITLE='THE NODES FOR THE STARTING THE POTENTIAL'
      WRITE(3,51) TITLE
      WRITE(3,*) NSTR
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE PATHS FOR THE POTENTIAL'
      WRITE(3,51) TITLE
C
C     WRITING THE PATH FOR THE POTENTIALS
C
      DO 310 I=1,Node
        IDEN(I)=0
  310   CONTINUE
      Isum=0
      it=0
      IDEN(NSTR)=1
      Do While(Isum.lt.Node)
       it=it+1
       DO 320 I=1,Node
        IF(IDEN(I).EQ.0)THEN
C
C     THE POTENTIAL AT THIS NODE HAS NOT BEEN FOUND YET
C     SEEING IF ANY OTHER NODE ON A TRINGLE IS KNOWN
C
          DO 330 J=1,Nele
            IF(n1(J).EQ.I.OR.n2(J).EQ.I.OR.n3(J).EQ.I)THEN
C
C     FOUND AN ELEMENT WITH THAT NODE FINDING IF ANOTHER NODE IS KNOWN
C
              IF((IDEN(n1(J)).EQ.1).and.(XX(n1(j)).lt.XX(i))) THEN
                WRITE(3,*) n1(J),I,J
                IDEN(I)=1
                GOTO 320
                END IF
              IF((IDEN(n2(J)).EQ.1).and.(XX(n2(j)).lt.XX(i))) THEN
                WRITE(3,*) n2(J),I,J
                IDEN(I)=1
                GOTO 320
                END IF
              IF((IDEN(n3(J)).EQ.1).and.(XX(n3(j)).lt.XX(i))) THEN
                WRITE(3,*) n3(J),I,J
                IDEN(I)=1
                GOTO 320
```

```
                END IF
              IF(IDEN(n1(J)).EQ.1)THEN
               WRITE(3,*) n1(J),I,J
               IDEN(I)=1
               GOTO 320
               END IF
             IF(IDEN(n2(J)).EQ.1)THEN
               WRITE(3,*) n2(J),I,J
               IDEN(I)=1
               GOTO 320
               END IF
             IF(IDEN(n3(J)).EQ.1)THEN
               WRITE(3,*) n3(J),I,J
               IDEN(I)=1
               GOTO 320
               END IF
              END IF
330         CONTINUE
          END IF
320    CONTINUE
       Isum=0
       Do 335 i=1,Node
         Isum=Isum+Iden(i)
335    continue
     End do
C
C      FINISHING OUT THE INPUT DATA
C
      WRITE(6,*)'   INPUT THE NUMBER OF TIME STEPS'
      READ(5,*) NTIME
      WRITE(6,*)'   INPUT THE NUMBER OF WAKES TO KEEP'
      READ(5,*) NKEEP
      WRITE(6,*)'   INPUT THE TIME INCREMENT'
      READ(5,*) DTIME
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE TIME PARAMETERS'
      WRITE(3,51) TITLE
      WRITE(3,*) NTIME,NKEEP,DTIME
      X=0.00001D0
      DIST=1000
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE CUTOFFS AND THE SPLIT DISTANCE'
      WRITE(3,51) TITLE
      WRITE(3,*) X,X,DIST
      WRITE(6,*)'   INPUT THE ROLL, PITCH AND YAW ANGLES'
      READ(5,*) XR,XP,YAW
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE ROLL, PITCH AND YAW ANGLES'
      WRITE(3,51) TITLE
      WRITE(3,*) XR,XP,YAW
```

```
      X=0.0D0
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE POINT ABOUT WHICH THE MOMENT IS CALCULATED'
      WRITE(3,51) TITLE
      WRITE(3,*) X,X,HEIGHT
      X=0.00027
      Y=0.00440
      Z=0.00467
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE MOMENT OF INERTIA VALUES'
      WRITE(3,51) TITLE
      WRITE(3,*) X,Y,Z
      X=-0.2860
      Y=0.0000
      Z=0.0000
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE CENTER OF MASS LOCATION'
      WRITE(3,51) TITLE
      WRITE(3,*) X,Y,Z
      X=0.0D0
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='THE DAMPING COEFICIENTS'
      WRITE(3,51) TITLE
      WRITE(3,*) X,X,X
      X=1.2D0
      Y=9.8D0
      Z=0.284
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='MASS PROPERTIES'
      WRITE(3,51) TITLE
      WRITE(3,*) X,Y,Z
      X=0.4767D0
      Y=14.00000D0
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='REFERENCE LENGTH AND VELOCITY'
      WRITE(3,51) TITLE
      WRITE(3,*) X,Y
      TITLE=' '
      WRITE(3,51) TITLE
C
C     Determine length properties
C
      XMax=0
      YMax=0
      ZMax=0
      XMin=0
      YMin=0
```

```
      ZMin=0
      XSum=0
      YSum=0
      ZSum=0
      Do 340 i=1,node
        if(XMax.lt.XX(i)) XMax=XX(i)
        if(YMax.lt.YY(i)) YMax=YY(i)
        if(ZMax.lt.ZZ(i)) ZMax=ZZ(i)
        if(XMin.gt.XX(i)) XMin=XX(i)
        if(YMin.gt.YY(i)) YMin=YY(i)
        if(ZMin.gt.ZZ(i)) ZMin=ZZ(i)
        XSum=XSum+XX(i)
        YSum=YSum+YY(i)
        ZSum=ZSum+ZZ(i)
340     Continue
      YWidth=YMax-YMin
      XLength=XMax-XMin
      XCenter=-XSum/Node
      X=0.02146
      Y=0.14650
      Z=0.29300
      TITLE='LENGTH PROPERTIES'
      WRITE(3,51) TITLE
   54 Format(6(F10.5))
      WRITE(3,54) X,Y,Z,YWidth,XLength,XCenter
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='INITIAL EULER ANGLES OF THE WING ROLL-PITCH-YAW'
      WRITE(3,51) TITLE
      WRITE(3,*) XR,XP,YAW
      X=0.0D0
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='INITIAL ANGULAR RATES OF THE WING ROLL-PITCH-YAW'
      WRITE(3,51) TITLE
      WRITE(3,*) X,X,X
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='LOCAL PRESURES CALCULATED'
      WRITE(3,51) TITLE
      TITLE='  N'
      WRITE(3,51) TITLE
      TITLE=' '
      WRITE(3,51) TITLE
      TITLE='HEIGHT ABOVE GROUND'
      WRITE(3,51) TITLE
      WRITE(3,*) HEIGHT
      Return
      End
*****************************************************************************
*                                                                           *
*                        Subroutine Order                                   *
*                                                                           *
```

```
************************************************************************
*                                                                      *
*     This subroutine takes any given matrix and reorders the          *
*     contents in ascending numerical order                            *
*                                                                      *
************************************************************************
      Subroutine ORDER(imax,Matrix)
      Implicit Integer(A-G,I-W)
      INTEGER Matrix(100),idum
C
      Do i=1,imax-1
        ip1=i+1
        Do j=ip1,imax
          if(Matrix(i).GT.Matrix(j)) then
            idum=Matrix(j)
            Matrix(j)=Matrix(i)
            Matrix(i)=idum
            endif
          enddo
        enddo
      return
      end
************************************************************************
*                                                                      *
*                      Subroutine Rorder                               *
*                                                                      *
************************************************************************
*                                                                      *
*     This subroutine takes any given matrix and reorders the          *
*     contents in ascending numerical order, and it reorders the       *
*     associated matrices accordingly                                  *
*                                                                      *
************************************************************************
      Subroutine RORDER(imax,Mat1,Mat2,Mat3)
      Implicit Integer(A-G,I-W)
      INTEGER Mat1(100),Mat2(100),Mat3(100)
C
      Do i=1,imax-1
        ip1=i+1
        Do j=ip1,imax
          if(Mat1(i).GT.Mat1(j)) then
            idum1=Mat1(j)
            Mat1(j)=Mat1(i)
            Mat1(i)=idum1
            idum2=Mat2(j)
            Mat2(j)=Mat2(i)
            Mat2(i)=idum2
            idum3=Mat3(j)
            Mat3(j)=Mat3(i)
            Mat3(i)=idum3
            endif
          enddo
        enddo
```

```
return
end
```

```
wship


CONTINUATION FROM ANOTHER RUN
   N

IS ITERATION REQUIRED
   N

THE NUMBER OF TRIANGULAR ELEMENTS
 24

THE ORDERING OF THE TRIANGULAR ELEMENTS
      5       4       1
      1       2       5
      6       5       2
      2       3       6
      4       5       7
      8       7       5
      5       6       8
      9       8       6
      3      10       6
     12       6      10
     13      12      10
     10      11      13
      6      12       9
     14       9      12
     15      14      12
     12      13      15
     11      16      13
     18      13      16
     16      17      18
     19      18      17
     20      15      13
     13      18      20
     21      20      18
     18      19      21

THE NUMBER OF NODES
 21

THE LOCATIONS OF THE NODES IN THE BODY FRAME
 -10.000000000000000  33.000000000000000    .000000000000000
  -5.000000000000000  22.500000000000000    .000000000000000
    .000000000000000  12.000000000000000    .000000000000000
 -17.000000000000000  33.000000000000000    .000000000000000
 -15.500000000000000  22.250000000000000    .000000000000000
 -14.000000000000000  11.500000000000000    .000000000000000
 -24.000000000000000  33.000000000000000    .000000000000000
 -26.000000000000000  22.000000000000000    .000000000000000
 -28.000000000000000  11.000000000000000    .000000000000000
    .000000000000000    .000000000000000    .000000000000000
    .000000000000000 -12.000000000000000    .000000000000000
```

```
 -14.000000000000000     .000000000000000     .000000000000000
 -14.000000000000000 -11.500000000000000     .000000000000000
 -28.000000000000000     .000000000000000     .000000000000000
 -28.000000000000000 -11.000000000000000     .000000000000000
  -5.000000000000000 -22.500000000000000     .000000000000000
 -10.000000000000000 -33.000000000000000     .000000000000000
 -15.500000000000000 -22.250000000000000     .000000000000000
 -17.000000000000000 -33.000000000000000     .000000000000000
 -26.000000000000000 -22.000000000000000     .000000000000000
 -24.000000000000000 -33.000000000000000     .000000000000000

THE NUMBER OF EDGE CORES
 16

THE NODES OF THE EDGE CORES
 1 2
 2 3
 3 10
 4 1
 7 4
 7 8
 8 9
 9 14
 10 11
 11 16
 14 15
 15 20
 16 17
 17 19
 19 21
 20 21

THE NUMBER OF CONVECTING EDGE CORES
 10

THE NODES OF THE CONVECTING EDGE CORES
 4 1 10
 7 4 10
 7 8 10
 8 9 10
 9 14 10
 14 15 10
 15 20 10
 17 19 10
 19 21 10
 20 21 10

THE NUMBER OF STARTING WAKE POSITIONS
 11

THE NUMBER OF THE NODE AND ITS START
 1 -10.0 33.0 .0
 4 -17.0 33.0 .0
```

```
 7 -24.0 33.0 .0
 8 -26.0 22.0 .0
 9 -28.0 11.0 .0
 14 -28.0 .0 .0
 15 -28.0 -11.0 .0
 17 -10.0 -33.0 .0
 19 -17.0 -33.0 .0
 20 -26.0 -22.0 .0
 21 -24.0 -33.0 .0

THE NUMBER OF NODES FOR THE KUTTA CONDITION
 7

THE NODES WHERE THE KUTTA IS IMPOSED
 7 5.0
 8 5.0
 9 5.0
 14 5.0
 15 5.0
 20 5.0
 21 5.0

THE NUMBER OF CORNERS
 4

THE CORNER NODES ARE
 1
 7
 17
 21

THE NUMBER OF STARTING LOCATIONS FOR POT
 1

THE NODES FOR THE STARTING THE POTENTIAL
 10

THE PATHS FOR THE POTENTIAL
 10 3 9
 3 6 4
 6 8 7
 8 9 8
 10 11 12
 6 12 10
 12 13 11
 9 14 14
 14 15 15
 13 16 17
 16 17 19
 13 18 18
 18 19 20
 15 20 21
 20 21 23
```

```
 6 2 3
 2 5 2
 5 7 5
 5 1 1
 5 4 1

THE TIME PARAMETERS
 60 20 10

THE CUTOFFS AND THE SPLIT DISTANCE
 1.000000000000000E-05 1.000000000000000E-05 1000

THE ROLL, PITCH AND YAW ANGLES
 .0 10.0 .0

THE POINT ABOUT WHICH THE MOMENT IS CALCULATED
 .0 .0 15.0

THE MOMENT OF INERTIA VALUES
 2.700000000000000E-04 4.400000000000000E-03 4.670000000000000E-03

THE CENTER OF MASS LOCATION
 -.286 .0 .0

THE DAMPING COEFICIENTS
 .0 .0 .0

MASS PROPERTIES
 1.2 9.8 .2839999999999999

REFERENCE LENGTH AND VELOCITY
 .4767 14.0

LENGTH PROPERTIES
    .02146    .14650    .29300  66.00000  28.00000  15.28571

INITIAL EULER ANGLES OF THE WING ROLL-PITCH-YAW
 .0 10.0 .0

INITIAL ANGULAR RATES OF THE WING ROLL-PITCH-YAW
 .0 .0 .0

LOCAL PRESURES CALCULATED
  N

HEIGHT ABOVE GROUND
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                     ***************                                  C
C                     *    SETELE     *                                C
C                     ***************                                  C
C                                                                      C
C       This routine orients the triangle so the first side is the     C
C       longest one.                                                   C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      Subroutine SETELE
      Implicit Double Precision(A-H,O-Z)
      PARAMETER (ME=366,MN=216,MEQ=850,MUN=500,MCI=80,MW=80)
      COMMON/ELDATA/ NELE,NODE,N(ME,3),XNODE(MN),YNODE(MN),ZNODE(MN)
C
C     Loop over all elements
C
      do 100 i=1,nele
        n1=N(i,1)
        n2=N(i,2)
        n3=N(i,3)
        x1=XNODE(n1)
        y1=YNODE(n1)
        z1=ZNODE(n1)
        x2=XNODE(n2)
        y2=YNODE(n2)
        z2=ZNODE(n2)
        x3=XNODE(n3)
        y3=YNODE(n3)
        z3=ZNODE(n3)

C
C     Determine lengths of sides
C
        r1=dsqrt((x1-x2)**2.d0+(y1-y2)**2.d0+(z1-z2)**2.d0)
        r2=dsqrt((x2-x3)**2.d0+(y2-y3)**2.d0+(z2-z3)**2.d0)
        r3=dsqrt((x3-x1)**2.d0+(y3-y1)**2.d0+(z3-z1)**2.d0)
C
C     Reorder nodes
C
        if((r2.gt.r3).and.(r2.gt.r1)) then
          N(i,1)=n2
          N(i,2)=n3
          N(i,3)=n1
        else if((r3.gt.r1).and.(r3.gt.r2)) then
          N(i,1)=n3
          N(i,2)=n1
          N(i,3)=n2
          endif
  100 CONTINUE
      return
      end
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                      ***************                                 C
C                      *    ISITOV   *                                 C
C                      ***************                                 C
C                                                                      C
C       THIS ROUTINE CALCULATES WHETHER THE POINT IN QUESTION          C
C       IS TOO CLOSE TO THE BODY AND IF IT IS MOVES IT.                C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE ISITOV(XP1,YP1,ZP1)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER (ME=366,MN=216,MEQ=850,MUN=500,MCI=80,MW=80)
      COMMON/MODATA/ PMOMX,PMOMY,PMOMZ,SPAN,CHORD
      COMMON/GROUND/ HEIGHT
      COMMON/ERRORS/ IERROR,KERROR,MERROR,NERROR
      COMMON/ELDATA/ NELE,NODE,N(ME,3),XNODE(MN),YNODE(MN),ZNODE(MN)
      COMMON/CORNER/ AELE(ME),BELE(ME),CELE(ME),
     .               DIRCOS(ME,3,3),DCI(ME,3,3)
      COMMON/CONSTS/ FOURPI,PI,CUTOFF,CUTWK,OFF
C
C     Set the offset value
C
      eps=0.000001d0
C
C     Exiting if the wake point is past the body
C
      if(DABS(XP1).gt.CHORD) goto 5
C
C     LOOPING THROUGH THE ELEMENTS TO CHECK EACH ONE
C
      DO 10 I=1,NELE
C
C     SEEING IF THE PROJECTION IS INSIDE OR ALONG THE ELEMENT
C
        X1=Xnode(N(i,1))
        Y1=Ynode(N(i,1))
        Z1=Znode(N(i,1))
        X2=Xnode(N(i,2))
        Y2=Ynode(N(i,2))
        Z2=Znode(N(i,2))
        X3=Xnode(N(i,3))
        Y3=Ynode(N(i,3))
        Z3=Znode(N(i,3))
        XMIN=min(X1,X2,X3)
        YMIN=min(Y1,Y2,Y3)
        ZMIN=min(Z1,Z2,Z3)
        XMAX=max(X1,X2,X3)
        YMAX=max(Y1,Y2,Y3)
        ZMAX=max(Z1,Z2,Z3)
C
        If((XP1.GE.XMIN).AND.(XP1.LE.XMAX).AND.
     .     (YP1.GE.YMIN).AND.(YP1.LE.YMAX)) Then
```

```
C
C     The wake point is over or under the element. Determine the
C     distance from the point to the plane using geometry.
C
          if((x1.eq.x2).and.(x2.eq.x3)) then
            a=1.d0
            b=0.d0
            c=0.d0
            d=x1
          else if((z1.eq.z2).and.(z2.eq.z3)) then
            a=0.d0
            b=0.d0
            c=1.d0
            d=z1
          else if((z1.eq.z2).and.(x1.eq.x2)) then
            a=(z3-z1)/(x3-x1)
            b=0.d0
            c=-1.d0
            d=x1*(z3-z1)/(x3-x1)-z1
          else if((z1.eq.z3).and.(x1.eq.x3)) then
            a=(z2-z1)/(x2-x1)
            b=0.d0
            c=-1.d0
            d=x1*(z2-z1)/(x2-x1)-z1
          else
            b=((x2-x1)*(z3-z1)-(x3-x1)*(z2-z1))/
     .        ((x2-x1)*(y3-y1)-(x3-x1)*(y2-y1))
            a=(z2-z1)-b*(y2-y1)
            c=-(x2-x1)
            d=x1*z2-x2*z1+b*(x2*y1-x1*y2)
            endif
   20     dist=DABS((d-a*xp1-b*yp1-c*zp1)/(DSQRT(a*a+b*b+c*c)))
          iredo=0
C
   30     if(dist.LE.OFF) then
C
C     The wake point is too close to the surface of the body
C
            write(6,*) ''
            write(6,*) 'The point: '
            write(6,500) XP1,YP1,ZP1
            write(6,*) 'is too close to the surface of: '
            write(6,500) X1,Y1,Z1
            write(6,500) X2,Y2,Z2
            write(6,500) X3,Y3,Z3
            write(6,*) 'Distance=',dist
 500        FORMAT(3F10.6)
            ZP1=ZP1+dabs((OFF-dist)*dircos(i,3,3))
            write(6,*) 'It was moved to: '
            write(6,500) xp1,yp1,zp1
            iredo=0
            endif
C
```

```
C     Limiting correction for wakepoints
C
          if(zp1.lt.ZMIN) then
            write(6,*) ''
            write(6,*) 'The point: '
            write(6,500) XP1,YP1,ZP1
            write(6,*) 'is inside the surface of: '
            write(6,500) X1,Y1,Z1
            write(6,500) X2,Y2,Z2
            write(6,500) X3,Y3,Z3
            write(6,*) 'Distance=',dist
            ZP1=ZP1+dabs(dist*dircos(i,3,3))
            write(6,*) 'Point moved to: '
            write(6,500) XP1,YP1,ZP1
            iredo=1
            endif
          if(iredo.eq.1) goto 20
          Endif
10      Continue
5     RETURN
      END
```

Vita

Scott R. Simmons was born in Richmond, Virginia on May 7, 1970. He attended the University of Tennessee at Knoxville and graduated with a B. S. in Engineering Science and Mechanics in 1992. He is currently completing his M. S. degree at Virginia Polytechnic and State University.

Mr. Simmons has worked at Oak Ridge National Laboratory in energy and robotics research, and he will begin work at Cummins Engine Company in Columbus, Indiana pending completion of his M. S. degree. Mr. Simmons has Engineer in Training status and is a member of Tau Beta Pi, Phi Kappa Phi, and the Society for Engineering Science and Mechanics.