Alternate Table Look-Up Routine For Real-Time Digital Flight Simulation

Michael K. Sinnett,* James E. Steck,†
Bruce P. Selbert,‡ and Robert B. Oetting‡
University of Missouri—Rolla, Rolla, Missouri

Introduction

THE increase in complexity of flight simulators in the last decade has placed an increasingly larger computational burden on the simulation host computer. In the past, this has been dealt with by using separate processors for different tasks. This is not always financially feasible, especially in a university setting, where an entire simulation might be limited to one or two processors. While a perturbation model, sometimes used in a partial task trainer, can be regarded as a linear system, the total force and moment model of a real-time man-in-the-loop simulation requires a more complex mathematical representation of the vehicle. Coefficients and derivatives are often functions of several variables, including control surface position, angle of attack, Mach number, and several other parameters. These functions seldom can be represented in closed form by a reasonably small number of equations. Often a table is created that consists of an array of known dependent variable values, tabulated at certain known values of one or more independent variables. Table 1 is an example of such a table. On each pass through the simulation program, a search is initiated through the appropriate tables, and an interpolation between known values is performed in each of the independent variable directions. This has been shown to be one of the most computationally intense aspects of the simulation process.[1] In order to retain model complexity and still achieve an acceptable frame rate, table look-up algorithms that reduce the computational load are highly desirable.

Two widely used search procedures are outlined by Rolfe and Staples.[2] The first, Method 1a, performs a top-down search between successive array values and exits once the proper interval is located. The second procedure, Method 1b, also performs a top-down search, but only if the position in the table has changed significantly. This is a method that has been used in industry.[3] The search procedure is employed for each independent variable in a multidimensional table, and the process is repeated for each simulation pass.

Once the proper location in the array is reached, an approximation of the dependent variable must be calculated. This dependent variable is considered to be a function f of n independent variables and is approximated, as F, by carrying out successive linear interpolations in each of the n directions as outlined in Refs. 2 and 3.

Alternate algorithms presented in this paper reduce the table search time by performing a top-down search during only the initialization pass. On all subsequent passes, the algorithm returns to the interval where the value will most likely reside. This is accomplished by tracking the direction and the rate of change of magnitude of each independent variable as it moves through the table. This is referred to as tracking the velocity of the variable. Once in the correct interval, the alternate methods presented here approximate the function over a given subregion of the table by a polynomial that is linear in each independent variable.

All of the above methods produce the same numerical results, but the algorithms presented here require fewer floating point calculations during the real-time portion of the simulation. When used in appropriate situations, these algorithms can significantly reduce look-up and interpolation frame times.

Table 1 Two-dimensional table example

Flap position, deg[a]	Sectional lift coefficient: Angle of attack, deg 0	2	4	6	8
0.0	0.4	0.6	0.8	1.0	1.2
5.0	0.9	1.2	1.5	1.7	1.9

[a]GAW-1 airfoil with a 30% chord single-slotted fowler flap, taken from Ref. 2.

Algorithm Descriptions

Four separate algorithms were developed to investigate table look-up computation speeds and data storage requirements. The first two methods, 1a and 1b, are based on methods presented by Rolfe and Staples.[2] The table look-up process consists of two distinct phases. The first is the table search, which is used to find the proper location in an n-dimensional table. Method 1a performs a top-down search between successive array values and exits once the proper interval is located. Because this search is performed on each independent variable array, n search processes are required for an n-dimensional table look-up routine. The entire process is repeated on each simulation pass. Method 1b performs the top-down search procedure only if the interval in which the independent variable resides has changed since the previous pass.

The alternate methods presented here, methods 2a and 2b, execute the top-down search only during the initialization pass. From then on, methods 2a and 2b return to the same independent variable intervals as on the previous pass, where the independent variable will still most likely reside. If the interval in which one of the independent variables was located has changed in one of the n dimensions, the new algorithms in methods 2a and 2b calculate the velocity of that independent variable through the table and proceed directly to the correct new interval. Little testing or looping is involved, except in cases of a discontinuity in the aircraft flight condition due to impulsive loading or a discrete cockpit event.

The second phase of the look-up process is the calculation of an approximation of the dependent variable, which is considered to be a function of n independent variables, X_1, X_2, ..., X_n. The function is represented by method 1a and 1b as a data table, an array of known values of f at different values of the independent variables spaced at intervals throughout the

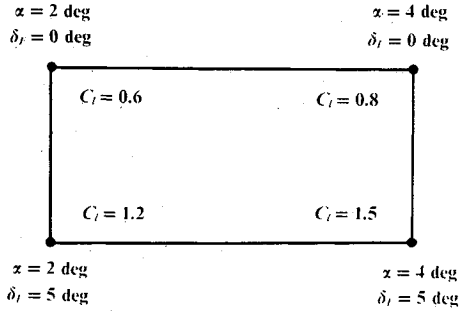

Fig. 1 Typical two-dimensional table subregion.

Received Aug. 14, 1989; presented as Paper 89-3310 at the AIAA Flight Simulation Technologies Conference, Boston, MA, Aug. 14-16, 1989; revision received Nov. 22, 1989.

*Gradute Student and National Science Foundation Fellow, Department of Mechanical and Aerospace Engineering and Engineering Mechanics. Member AIAA.

†Postdoctoral Fellow, Intelligent Systems Center. Member AIAA.

‡Professor, Department of Mechanical and Aerospace Engineering and Engineering Mechanics. Associate Fellow AIAA.

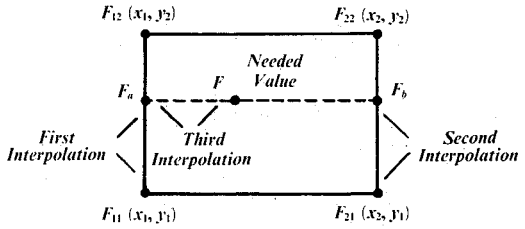

Fig. 2 Two-dimensional linear interpolation procedure.

domain in which f is defined. The domain of f is divided into a number of rectangular subregions, each corner of which is a point at which the function is defined. Figure 1 is an example of a two-dimensional subregion from Table 1.

When a value for f is needed by one of the simulation routines, the table search locates the proper rectangular subregion, and in methods la and 1b, a linear interpolation scheme is employed to approximate f in terms of known values at points nearby. Successive interpolations must be carried out for each of the independent variables in an n-dimensional table, one for X_n, two for X_{n-1}, up to 2^{n-1} interpolations for X_1. For a two-dimensional look-up, two intermediate values must be interpolated, F_a and F_b, before the final approximation F can be found. This procedure is illustrated in Fig. 2.

Methods 2a and 2b recognize that the effect of the successive interpolations is to represent the function f over the given subregion with a polynomial approximation F that is linear in each independent variable X_i. It is further recognized that for an n-dimensional table, each subregion has 2^n corners, at which the value of f and the values of the independent variables $X_1, X_2, \ldots, X_n$ are known. The required n-linear polynomial has 2^n coefficients, one for each term. Thus, for each subregion in an n-dimensional table, we have 2^n equations in 2^n unknowns, and the coefficients are uniquely determined. A polynomial expression to approximate the function in the given subregion is then known. For example, Fig. 1 would yield a system of four equations of the form

$$A_1 + A_2\alpha + A_3\delta_F + A_4\alpha\delta_F = C_l \tag{1}$$

to be solved for the four unknowns A_1, A_2, A_3, and A_4. This yields the bilinear polynomial expression for C_l in the given subregion

$$C_l = 0.40 + 0.10\alpha + 0.10\delta_F + 0.01\alpha\delta_F \tag{2}$$

which can be written in the form

$$C_l = 0.40 + \alpha(0.10 + 0.01\delta_F) + 0.10\delta_F \tag{3}$$

This equation requires only 6 floating point operations to calculate a value for C_l, as compared to the 15 required for the linear interpolation scheme of methods 1a and 1b for a two-dimensional table look-up routine.

Method 2a precalculates the polynomial coefficients for all of the subregions in the table before real-time simulation. This can be done during an initialization pass, or the tables can be preprocessed, the coefficients stored, and then read as input for the actual simulation. One drawback to method 2a is the data storage required for the polynomial coefficients of each table. For a two-dimensional table, four coefficients are needed for each subregion. An interpolation table of n by m values requires $4(n-1)(m-1)$ coefficients to be stored for method 2a. A four-dimensional table would require 16 coefficients for each subregion, or $16(k-1)(m-1)(n-1)(p-1)$ coefficient for a table of k by m by n by p values. It should be emphasized that search speed is not affected by these larger arrays, since searching is performed on only the independent variable arrays.

Method 2b uses the same table search procedure and polynomial representation of the interpolation as method 2a, but it does not precalculate the polynomial coefficients before real-time simulation. Instead, each time an interpolation point enters a new table subregion, the coefficients for only that subregion are calculated and stored. This virtually eliminates the additional storage required for the polynomial coefficients. For example, a four-dimensional look-up table would require the storage of only 16 coefficients at any one time. The obvious disadvantage of this method is the additional computation time required for the calculation of coefficients when entering a new subregion.

Computational Results

The four algorithms studied were coded in FORTRAN for two-, three-, and four-dimensional table look-up routines, and test cases were run on an Apollo DN10000. The two-dimensional tables were 14×12, the three-dimensional tables were $14 \times 12 \times 12$, and the four-dimensional tables were $14 \times 12 \times 12 \times 12$ in size. One important measure of real-time computational performance is simulation frame speed. Over 3×10^6 calls to the look-ups were made, and the average frame speed was calculated. In each test case, a new table subregion was entered on 1% of the total number of calls.

The results are presented in Table 2. For the two-dimensional case, method 1a the slowest. This is because a top-down search is employed on each pass. Method 1b is faster because the top-down search is conducted only upon entrance into a new subregion. Since a two-dimensional linear interpolation requires 15 floating point operations and the calculation of a bilinear polynomial requires only 6, methods 2a and 2b are both faster than methods la and 1b. In order to calculate the coefficients of the bilinear polynomial required for the two-dimensional case, method 2b requires the solution of a system of four equations in four unknowns each time a new subregion is entered. These coefficients are stored in a one-dimensional array. Method 2a precalculates these coefficients, but they are stored in a three-dimensional array. For the two-dimensional case, it happens that the overhead due to the calculation of the coefficients is lower than the overhead due to the integer operations associated with manipulating the large three-dimensional array of coefficients. Thus, for the two-dimensional case, method 2b is faster than method 2a. These same comments also apply to the three-dimensional case, where the solution of a system of nine equations in nine unknowns requires less overhead than the integer manipulation of the large four-dimensional coefficient array. In the four-dimensional case, however, both the overhead required to solve a system of 16 equations in 16 unknowns and the integer overhead required to manipulate the five-dimensional array of coefficients are greater than the time required to perform the 15 successive linear interpolations necessary, and method 1b remains the fastest.

The use of method 2a increases the data storage requirements for reasons outlined in the previous section. For the two-dimensional 14×12 table, the required storage increases from 0.7K to 2.3K when changing from method 1b to method 2a. The data storage requirement increases from 8K to 50K for a three-dimensional $14 \times 12 \times 12$ table, and from 97K to 1100K for a $14 \times 12 \times 12 \times 12$ table. Method 2b requires virtually no additional storage.

Table 2 Algorithm frame speed[a]

Method	Two-dimensional	Three-dimensional	Four-dimensional
1a	21.7	33.2	47.2
1b	12.7	20.0	32.6
2a	9.1	17.6	36.3
2b	7.9	15.0	38.6

[a] In μs.

Recommendations and Conclusions

The use of method 2a is recommended for two-dimensional look-up situations. It is the fastest method and carries little additional storage requirements for the polynomial coefficients. In three-dimensional look-up cases, the use of method 2b is recommended, but only if the independent variables are moving into new subregions on 1 or 2% of the total number of calls. If the independent variable subregions are changing more rapidly, then the use of method 2a is suggested. This increases the storage requirements and represents a tradeoff of memory for speed. Neither of methods 2 are recommended for a four-dimensional (or higher) look-up routine. Method 2a drastically increases storage requirements, and method 2b results in an unacceptably high frame time when a new independent variable subregion is entered.

The development of these new algorithms supplies the simulation engineer with another set of tools for table look-up applications. Like any tools, they must be used carefully and only in the appropriate situations. All of the methods described here produce the same numerical result, but the new algorithms presented require fewer floating point calculations during certain real-time portions of the simulation. When used in appropriate situations, these algorithms can significantly reduce look-up frame times.

References

[1]Forsstrom, K. S., "Array Processors in Real-Time Flight Simulation," IEEE Paper 83-0062, 1983.

[2]Rolfe, J. M., and Staples, K. J., "Representation of Aerodynamic Data," *Flight Simulation*, Cambridge Univ. Press, Cambridge, England, 1986, pp. 53–60.

[3]McDonnell Aircraft Company, "Hybrid Library Summary," Flight Simulation Laboratory D-251, Dec. 1982.

[4]Wentz, W. H., Jr., and Seetharam, H. C., "Development of a Fowler Flap System for a High Performance General Aviation Airfoil," NASA CR-2443, Dec. 1974.

Connection Between Leading-Edge Sweep, Vortex Lift, and Vortex Strength for Delta Wings

Michael J. Hemsch*
PRC Aerospace Technologies Division, Hampton, Virginia
and
James M. Luckring†
NASA Langley Research Center, Hampton, Virginia

Nomenclature

A = coefficient of power-law function
C_L = total lift coefficient referenced to planform area
$C_{L,nl}$ = nonlinear component of lift coefficient, $C_L - K_p\alpha$
$C_{L,p}$ = component of lift coefficient due to attached flow without leading-edge suction
$C_{L,v}$ = vortex lift component
f,g = similarity functions
K = Sychev similarity parameter, $\tan\alpha/\tan\epsilon$
K_p = slope of lift coefficient at zero angle of attack
K_v = suction analogy coefficient for separated flow
ℓ = length of wing centerline chord
ℓ_{le} = length of delta wing leading edge
M_∞ = Mach number
S_p = wing planform area
s = semispan of wing
α = angle of attack
Γ = vortex circulation
ϵ = wing apex half angle
Λ = leading-edge sweep

Introduction

THE purpose of this Note is to clarify the effect of leading-edge sweep on the vortex lift and leading-edge vortex strength of a slender wing. It is often believed that an increase in sweep increases both the vortex lift and the vortex strength. However, the opposite is true. Using the suction analogy,[1] similarity,[2] and numerical and experimental data, we derive simple formulas giving the actual dependence for delta wings. Some important aspects of leading-edge vortex lift as delineated by Polhamus[1] are reviewed first.

Difference Between Vortex and Nonlinear Lift

As part of his work on the leading-edge suction analogy for sharp-edged delta wings, Polhamus analyzed the effect of the loss of leading-edge suction due to leading-edge separation.[1] He showed that the variation with angle of attack of the zero-suction attached-flow lift is given by

$$C_{L,p} = K_p \sin\alpha \cos^2\alpha \tag{1}$$

where K_p is used to designate the lift-coefficient slope at zero angle of attack. Hence, as defined by Polhamus, the lift that should be associated with the leading-edge vortex (i.e., the so-called vortex lift $C_{L,v}$) is given by

$$C_{L,v} = C_L - K_p \sin\alpha \cos^2\alpha \tag{2}$$

Unfortunately, $C_{L,v}$ is often confused with the so-called nonlinear increment of lift $C_{L,nl}$, which is given by

$$C_{L,nl} = C_L - K_p\alpha \tag{3}$$

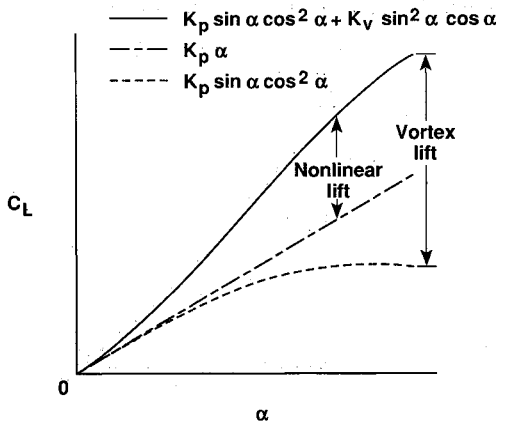

Fig. 1 Difference between vortex lift and nonlinear lift.

Received Oct. 30, 1989; revision received Dec. 18, 1989.

*Engineering Specialist; currently with Lockheed Engineering & Sciences Company, Hampton, VA. Associate Fellow AIAA.

†Research Engineer. Associate Fellow AIAA.