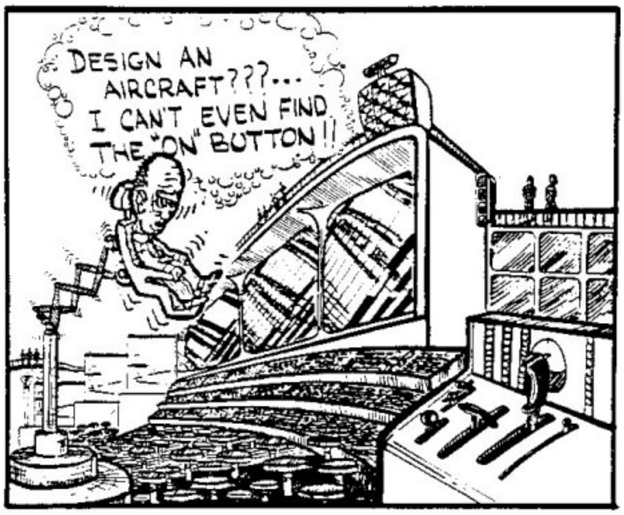
Use of Computers in the (Conceptual) Design Process


DESIGN AN AIRCRAFT???... I CAN'T EVEN FIND THE "ON" BUTTON!!

Short Course Handouts by

Daniel P. Raymer

Advanced Systems Design

North American Aviation (Rockwell)

These notes were written around 1980 as lecture notes for Lee Nicolai and Jay Pinson's Aircraft Design Short Course in Dayton, Ohio. That class was organized as a series of lectures by different experts in various areas, such as Jan Roskam for Stability & Control and Barnes McCormick for High Lift aero. For 5 years or so I gave a presentation on CAD in conceptual design, and always enjoyed the chance to meet so many smart and interesting people. Thanks, Lee!

While out of date in many areas (storage tube?) there is still-relevant material here, and people have asked for a copy.

USE OF COMPUTERS IN THE DESIGN PROCESS

Daniel P. Raymer
Advanced Systems Design
Rockwell International, NAAO

INTRODUCTION

It is difficult to fully comprehend the impact computers have had on aircraft design in the past twenty years or so. Virtually every phase of design in the major aerospace companies is either "on" the computer, heavily dependent upon the computer, or about to be put on the computer.

One yardstick of this infiltration can be found in today's new-hire engineers. Many have never seen, let alone mastered the slide rule. They think nomograms are messages delivered by short people. "Plenimeter" evokes a complete blank, as do most of the arcane mysteries of the drafting board. But, they can program in several languages before they reach college, they have a computer in the closet, and they play video games for relaxation.

The reasons for the widespread use of computers in the design process are obvious; time, cost and quality.

Time savings from the use of computers are usually the major reason for their purchase. Computers can make repetitive calculations millions of times faster than humans, and are (almost) never wrong. Hand calculation of aircraft initial sizing takes about half a day. A similar calculation on a microcomputer program at Rockwell takes a few seconds.

Cost savings result from time savings. Even several hundred dollars per hour of computer charges is a bargain if one engineer can do the work of ten. Minicomputers can cost under twenty dollars an hour, and microcomputers are practically free on an hourly basis.

Quality improvement is the most important but least recognized benefit of computers. First, the greatly reduced design and calculation time allows a vast increase in the number of iterations possible with limited time and funds. Second, the use of common computerized data bases eliminates error propagation in the transfer of data between functional groups. Finally, there are some design and analysis techniques which are feasible on the computer which would be virtually impossible otherwise.

This paper will outline some fundamentals of computers in the aircraft design process. Emphasis will be placed on computer graphics and computer-aided design (CAD), as opposed to computational analysis. Also, conceptual design rather than detail part design will be stressed. Computer equipment will be discussed, particularly the design stations. User interface options will be compared. Primitive graphics capabilities will form a foundation for more sophisticated geometry representations and graphics mathematics. Animation and analysis interfaces will be discussed. Finally, a representative computer-aided conceptual design program will be described, highlighting the integrated use of computer graphics and computational analysis.

COMPUTER GRAPHICS EQUIPMENT

Bits and Bytes

A digital computer can remember whether a quantity is a "1" or a "0". Furthermore, it can add that quantity to another quantity (1 or 0). From this small beginning comes the power to revolutionize the design process, if not the world!

The "1" or "0" quantity is called a "bit" (binary digit) and is physically represented by a current being either "on" or "off". This alone can transmit very little useful information. Instead, a convention has been established in which eight* bits, called one "byte", are used in the binary (base 2) mathematical system to represent the numbers from zero through 127**, which in turn are used to identify both characters and numerals as shown in Table I. This "American Standard Code for Information Interchange" (ASCII) is used for communication between computers and between components of a computer, such as the central processing unit (CPU) and the keyboard (ASCII is the most common convention, but others such as IBM's "EBCDIC" are also in use). Within the computer, pure binary code is usually used for arithmetic operations, with usually four bytes per number in binary exponential notation.

TABLE I. ASCII CODE

BITS b_7 b_6 b_5				ø ø ø	ø ø 1	ø 1 ø	ø 1 1	1 ø ø	1 ø 1	1 1 ø	1 1 1
b_4	b_3	b_2	b_1	CONTROL		HIGH X & Y GRAPHIC INPUT		LOW X		LOW Y	
ø	ø	ø	ø	NUL 0	DLE 16	SP 32	ø 48	@ 64	P 80	` 96	p 112
ø	ø	ø	1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
ø	ø	1	ø	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
ø	ø	1	1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
ø	1	ø	ø	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
ø	1	ø	1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
ø	1	1	ø	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
ø	1	1	1	BEL 7 BELL	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1	ø	ø	ø	BS 8 BACK SPACE	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1	ø	ø	1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1	ø	1	ø	LF 10 LINE FEED	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1	ø	1	1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1	1	ø	ø	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	| 124
1	1	ø	1	CR 13 RETURN	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1	1	1	ø	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
1	1	1	1	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	RUBOUT DEL 127

*Usually

**This uses only seven bits. The eighth bit is frequently used to check for transmission errors, called "parity" checking.

One important parameter in computer operation is the number of bits which can be manipulated at one time. Most personal computers are "eight-bit" machines, in that eight bits are handled at one time. Some personal computers and minicomputers are 16-bit - i.e., two bytes can be manipulated at one time. Most minicomputers are 16 or 32-bit, while most main-frame computers are 32, 60, or 64-bit†. The more bits handled at one time, the better, in terms of arithmetic speed and communication speed to devices such as disks and printers. A "word" is defined as however-many bits that computer can handle at one time - i.e., for a 32-bit machine, a word is 32 bits, or 4 bytes.

Another important computer parameter is the amount of memory available in "core" - i.e. without resorting to a mass storage device such as disks, magnetic tapes, or (horrors!) punched cards. Memory size is measured in thousands (K)†† of bytes. Remember that one byte holds one character, or can hold eight digits (bits) of a binary number. A typical memory size of a minicomputer or recent personal computer is 32K or 64K. Some of the latest main frames can hold over one megabyte†††. This memory size is extremely important because it sets upper limits to program length and data access without time-consuming disk access.

Some computers have a feature called "virtual memory" which allows the computer to pretend it has more memory than it actually has. A program too big to fit in the core memory is read in piecewise, so that only the parts of the program actually being executed are in core. The other parts of the program are automatically read in from the disk as they are needed. Computers without this feature rely on "overlays". Here the programmer must break the program into parts and bring in the appropriate part with specific programmed instructions.

A final parameter important to computer-aided design is speed. This is not one clear number, but is composed of many elements. Most basic of these is clock, or cycle speed. This speed is controlled by an internal clock which pulses at regular intervals, controlling timing of operations. Cycle times in the hundreds of nanoseconds are typical. Another important factor relating to speed is the time required for arithmetic operations. Still another factor is disk access time, and another is communications time to terminals or other peripherals. Due to the complexity of these parameters, it is customary to compare speeds of different computer systems by the time it takes to execute some selected benchmark program. For computer-aided design applications, the benchmark program should include both graphic and arithmetic operations.

†For talking purposes, consider a main-frame as a "big" computer operated by someone else, a minicomputer as a refrigerator-sized computer operated by you, and a personal computer as a desktop computer operated by your kid.

††Actually, K = 2^{10} = 1,024 bits.

†††megabyte = 2^{20} = 1,048,576 bits.

Terminals and Input Devices

The most conspicuous device used for computer-aided design is not the computer, but the terminal. Currently, most terminal displays are of the cathode-ray tube (CRT) type, although that may change. A CRT works like a television screen - an electron beam is focused and aimed by magnetic fields to produce a spot on a phosphorous screen. In a "refresh" tube terminal, the spot almost immediately disappears. Thus, the whole screen must be "repainted" at least 30 times a second to maintain the display. If you wish the display to change, you merely change the beam aiming between repaints. A refresh tube can be of two types - raster or vector. Raster tubes act just like television screens - the entire screen is scanned in a regular pattern, with the electron beam on or off to produce a dot (raster) pattern. A vector tube is not scanned - the electron beam is aimed to paint only those lines desired. Figure 1 diagrams a refresh tube.

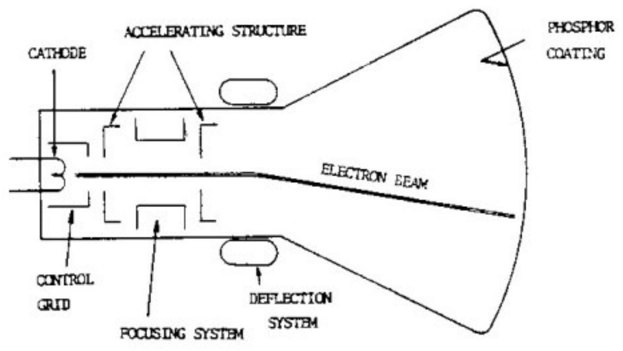


Figure 1. Refresh Tube CRT

A "storage" tube CRT does not continuously repaint itself. Instead, it has two electron producers - an aimable beam like a refresh tube, and a "flood" cathode which sends a broad, continuous stream of electrons. A fine wire screen, called the backplate (see Figure 2), is placed just inside the screen, and is negatively charged. When the aimed electron beam strikes the backplate an emission of electrons is produced, leaving a positively charged area. It is only at the positively charged areas that the flood electrons can pass through the backplate to strike the phosphor screen and cause a spot to appear.

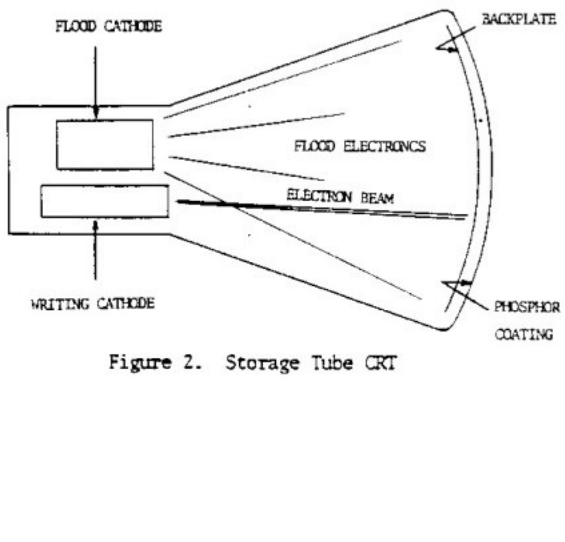


Figure 2. Storage Tube CRT

Once an area is "written on" by the beam, the resulting spot remains visible on the screen. Thus, constant repainting of the screen is avoided. Because of this, there is no limitation to the number of lines that can be drawn on a storage-tube CRT, whereas a vector refresh tube will begin to flicker if the number of lines to be continuously redrawn gets too large. To erase a storage-tube CRT, a brief positive charge is applied to the backplate, producing a quick flash.

Some terminals combine features of refresh and storage tube CRT's. Most graphics are stored on the tube like a storage tube, but a "refresh buffer" is available for text and limited graphics by use of a beam too weak to cause backplate electron emission. For example, in positioning a wheel, the complete aircraft could be drawn on the storage tube, but the wheel could be drawn in refresh, allowing it to be moved around.

Virtually all terminals used for design have a typewriter-like keyboard, and most have some additional types of input devices. Some have function buttons (or keys) used to augment the keyboard. Some terminals also have an adding-machine (10-key) keyboard for entering numbers rapidly.

Most graphics terminals have some means of specifying a location on the screen. Perhaps the most common is the graphics crosshair. This is a faint crossed line on the screen which can be moved around. Its movement can be controlled by thumb-wheels, a joystick, buttons, dials, a tracker ball, or a palm-held device called a "mouse" which is slid around the table by the user.

Another way to move a graphics crosshair on a raster CRT is via a light pen. This device looks like a thick pencil and is actually touched to the screen where it senses the location of the electron beam. From this, the indicated location on the screen can be determined, and the graphics crosshair moved accordingly. For a vector refresh tube, a temporary raster pattern can be requested whenever the light pen is to be used. In addition to controlling a crosshair, the light pen can be used to identify elements on the screen, including line segments and available command words. In one very popular CAD system, most of the work can be done with just the light pen and an occasional press of a function button.

Another very potent input device is the magnetic (or digitizer) tablet. This is a flat surface of virtually any size which uses embedded wires to detect the presence of a pen-like device or hand-held puck. This can also be mechanized acoustically or mechanically. The tablet can be used to control a crosshair, select commands printed on the tablet, or input a drawing taped to its surface. This approach is followed by a major competitor to the system mentioned above, with similar beneficial results.

Output Devices

There are a variety of graphics output devices. Simplest for the user are the screen copy devices. These make a print of the actual images on the screen, usually by electrostatic or photographic means. These have the advantage of speed, but are not very accurate. A common problem of these devices is misalignment of scale in the vertical and horizontal axis, producing a slightly distorted image. This is avoided in more sophisticated electrostatic plotters, which produce a drawing by rasterization of geometric data rather than copying the screen.

For greater accuracy, pen plotters are often used. In these devices an ink pen is mechanically moved across a piece of paper (or vice-versa). Some pen plotters allow multiple ink colors. Pen plotters range from notebook paper size to 20 or more feet long. Their main disadvantage is their comparatively slow speed.

There are other devices which output directly to microfilm, videotape, or motion picture film. The latter two can be used for animation, as described in a later section.

Communications

Computer communications are very important for development of common data bases. The best form of communication is via direct, or "hard" wiring. Interface devices allow computers to talk over direct wires at very high rates of data transmission, called the "baud" rate (bits per second). A slower but widely used type of communication is over the phone lines. This requires conversion from "parallel" (i.e., many bits at a time) computer outputs to "serial" (i.e., one bit at a time) signals. A "modem" (modulator-demodulator) is used to convert the computer's digital signals to analog signals for transfer over phone lines. For fixed installations using the phone lines on a regular basis, the modem is directly wired to the telephone network. For portability and low cost, an "acoustic coupler" can be used. This is a device which transmits and receives high frequency sound through a regular telephone headset placed in a cradle. The acoustically coupled modem is quite slow and suffers from higher transmission errors.

Even slower forms of communication between computers are acceptable for certain applications. Magnetic tape can be used for transfer and storage of large amounts of data. Computer disks, especially the inexpensive and portable "floppy" disks, can also be used for data transfer. Others include the obsolete paper tape and punched cards, and cassette tapes for some personal computers.

Languages

As mentioned earlier, computers understand only "1" and "0". Therefore, actual instructions that can be understood by the computer must be coded as a series of "1"'s and "0"'s. For example, "1000 1010 1110 1111" may mean "add what's in location "1110" to what's in location "1111" and store the result back in location "1111". This type of instruction, called "machine code" is all that the computer can directly understand. Obviously, this is extremely difficult to use for writing programs. To make things a little easier, "assembly code" uses easy-to-remember words instead of binary numbers. Our example in assembly code may read "ADD D,E". This still requires the programmer to keep track of exactly where all the numbers used in the program are stored. However, in the hands of an experienced programmer assembly code is unmatched in speed and thus is frequently used for applications such as trigonometric functions in which speed is critical.

For most of us, higher-order languages are required. There are computer languages such as FORTRAN, BASIC, and PASCAL in which easy-to-remember instructions are used. For example, our previous example becomes something like "A,= B + A". However, this means nothing to the computer. A program in a higher order language is automatically converted into machine code by yet another computer program (itself stored as machine code). There are two options. In the case of BASIC and some other languages, the program can be converted during execution. This is called an "interpreter." The other option is to convert to machine code prior to execution, using a program called a "compiler." In this case, a machine code version of the program is created and stored. The machine code version is called the "object deck" while the original higher-order language program listing is called the "source deck" (or "object code" and "source code"). Interpretive languages have the advantage of being executable without an intermediate step to compile the program but compiled programs are faster in execution.

USER INTERFACE

The most critical component of a computer-aided design system is the manner in which the user and the computer talk to each other. This more than anything else will determine the productivity of the man-machine team.

The user interface has conflicting requirements. For ease of use, the computer should give detailed prompts and instructions to the user. This is especially important for new users. On the other hand, detailed prompts and instructions significantly slow down the experienced user. At least one program has been written which asks the user his experience level, then provides an appropriate level of assistance. Obviously, this requires substantial programming complexity.

Figure 3 shows the unfortunate reaction of many new users to a complicated CAD system.

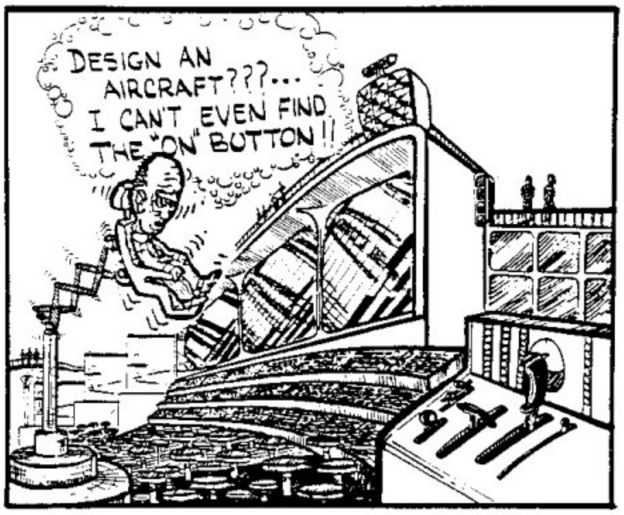

DESIGN AN AIRCRAFT???... I CAN'T EVEN FIND THE "ON" BUTTON!!

Figure 3. New User Meets Computer

The computer and the user talk to each other through the terminal. The computer addresses the user mainly through words written on the screen, although graphics and sounds are also used. The user talks to the computer through the devices mentioned before - keyboard, graphics cursor, digitizer, and others. These input devices are used in one of two ways.

Usually, input devices are called by a program statement that instructs the computer to wait for a user input. For example, a statement for keyboard inputs will wait for a line of typed letters followed by the user pressing the return key on the keyboard. Nothing happens until return is pressed. When it is pressed, the typed-in information, expressed as ASCII (or other) characters, is passed to the program as the contents of a variable, and can be acted upon within the program.

Another type of user input is the "interrupt." Here the program is suddenly jumped, by the pressing of a key, from its current task to an alternate task. For example, a long iteration which is not converging to a solution could be aborted by pressing a "break" key, which jumps the program to another task such as waiting for another command.

The simplest type of user interface is the "menu". Here the computer lists the available options, usually with number identifiers, and awaits the user's selection. The user need only type in the name or number of the desired option. For example, a menu may read "(0)-STOP; (1) DESIGN; (2) ANALYZE". The user merely enters 0, 1, or 2. This approach is easy to program and easy to use. Note that the simplest type of menu is the YES/NO prompt. The problem with this approach is the amount of time it takes to specify a particular task if the program has numerous capabilities. Imagine how many menues would have to be printed and selected to tell the computer to scale the tip airfoil of a vertical tail by two percent in thickness!

An alternate approach is the command language. Here the user must learn to type in a number of distinct commands, such as DRAW, ERASE, SCALE, etc... Options can be entered along with the command, allowing infinite flexibility with one input string. However, the user must "speak" the new language, requiring some learning time. Also, the programming required to decode an input can be substantial, especially if some degree of "friendliness" is required. For example, humans tend to leave out the decimal point when typing in a round number such as (2). A simple command interpreter will fail to understand (2) as the (2.0) it was expecting!

A hybrid approach offers many advantages. Here the regularly used options are selected by a command language, while complicated or irregularly used options go into a menu mode. The user may enter the command "DRAG", and be prompted via menues for all inputs required to calculate drag.

Any of these approaches can be used with input devices other than the keyboard. Function keys may substitute for command words or menu responses. The cross-hair or light pen can be used to select from a menu. A detailed list of commands can be physically printed on a digitizer tablet, and commands or menu responses can be selected with the digitizing pen or puck.

The system designer must decide whether to benefit new or occasional users with a menu-oriented system, or benefit experienced users with a faster, command-oriented system, or try for some happy medium.

GRAPHIC PRIMITIVES

Every interactive graphics computer has its own unique graphics program statements-standardization is, alas, in the future. These statements are written in whatever programming language is being used - generally FORTRAN for scientific work. Whatever the system and language, there are common capabilities which must be present in some form or another. These graphics "primitives" are described below.

Most computer graphics are composed of straight lines. To get a curve, a number of very short, straight lines must be drawn. The most basic graphic primitive would be something like LINE (X,Y). This would draw a line from the current (i.e., last) position to a new position (X,Y). Some similar program statements can be found in all computer graphics programs. To move to a new position without leaving a line would require some statement such as MOVE (X,Y). This is used to get to the start point of the next line if it doesn't begin where the last one ended. With these two primitive graphics statements in a higher-order program, virtually anything could be drawn.

Another obvious requirement is an ERASE statement. On storage tube terminals, this must erase the entire picture. With a refresh terminal, it should be possible to specify just certain lines for erase, either by (X,Y) location or some designating device such as a light pen.

Another primitive capability required is the ability to write text and symbols, perhaps by statements entitled TEXT and SYMBOL. These would write text or draw symbols at the current location of a cursor (the cursor is an imaginary entity which exists at the last location specified).

If the picture being drawn is larger than the screen, the picture must be truncated at the screen borders to prevent erroneous lines from appearing. This requires some "clipping" statement, which could also be used to create a viewing window on the screen.

Some more sophisticated computer systems have color and shading capabilities, requiring separate program statements. Other statements include sound generators such as bell or musical tone statements, used to scold, reward, or wake up the user. Statements for outputting graphics on other devices such as pen plotters are also typical.

User inputs are programmed through input statements. A statement such as KEYBOARD will await a line of characters followed by a "return." Various "device" statements will await inputs from the specified device, such as a digitizer which reads in X and Y values. One typical statement simultaneously reads in one character from the keyboard plus the screen location of the crosshair as the key is struck.

There are other specialized program statements for various applications not given here, but a complete listing would be impossible. As mentioned before, those described above are not actual statements for any one machine, but are typical of the functions which similar program statements will provide in an actual computer system. Such statements are written into the program to write menues on the screen, read the user's response, input geometry, plot analysis results, draw figures, or any other features required of a computer graphics program. Table II lists a typical FORTRAN program segment using terminal input, erase, and line drawing statements from the TEKTRONIX PLOT-10 graphics package.

TABLE II. TYPICAL FORTRAN LISTING

```
      SUBROUTINE PMZOOM (HX,HY,*)
C
$INSERT CMOCOM.INSERT.F77
$INSERT TEKCOM.INSERT.F77
C
      DATA KEYN/78/,KEYQ/81/,BIG/9.0E+10/,SMALL/-9.0E+10/
C
C...FUNCTION:
C     RESET SCALE FACTOR AND ORIGIN TO PROVIDE ZOOM CAPABILITY
C...INPUTS:
C     ARGUMENTS
C        HX,HY    POSITION OF LOWER LEFT CORNER OF WINDOW
C     COMMON  /CMOCOM/
C        ITYPE    SEE COMINT
C...OUTPUTS:
C     COMMON  /TEKCOM/
C        FACT     DISPLAY UNITS CONVERSION FACTOR
C        IZOOM    ZOOM INDICATOR
C        XORG     GRAPHICS CURSOR ABSICSSA ORIGIN IN SCREEN INCHES
C        YORG     GRAPHICS CURSOR ORDINATE ORIGIN IN SCREEN INCHES
C...LOCAL VARIABLES:
C        X2,Y2    POSITION OF UPPER RIGHT CORNER OF WINDOW
C
C
C
C DRAW LOWER LEFT CORNER OF WINDOW
C
      CALL PLOT (HX,HY,3)
      CALL PLOT (HX,BIG,2)
      CALL PLOT (HX,HY,3)
      CALL PLOT (BIG,HY,2)
C
C SET UPPER RIGHT CORNER OF WINDOW
C
      CALL CURSOR (X2,Y2,KEY)
      IF (KEY .EQ. KEYQ) RETURN 1
      IF (KEY .EQ. KEYN) GO TO 3610
C
C DRAW UPPER RIGHT CORNER OF WINDOW
C
      CALL PLOT (X2,Y2,3)
      CALL PLOT (X2,SMALL,2)
      CALL PLOT (SMALL,Y2,2)
C
C SET SCALE FACTOR AND ORIGIN
C
      XFACT = 14.0/ABS(X2-HX)
      ZFACT = 10.5/ABS(Y2-HY)
      IF (XFACT .LT. ZFACT) ZFACT=XFACT
      RATIO = ZFACT/FACT
      XORG = (XORG-(XORG+HX*FACT))*RATIO
      YORG = (YORG-(YORG+HY*FACT))*RATIO
      FACT = ZFACT
      GO TO 3620
C
C NUMERIC INPUT
C
3610  CALL WRTUBE (0,0,'(** INPUT ZOOM FACTOR**)')
      CALL COMINT
      IF (ITYPE(1) .LT. 2) GO TO 3610
      CALL REALIN (1,RATIO)
      IF (RATIO.LT.0. .OR. RATIO.GT.1.) GO TO 3610
      IF (RATIO .EQ. 0.) GO TO 3630
      XORG = 7.00*(1.-RATIO)+XORG*RATIO
      YORG = 5.25*(1.-RATIO)+YORG*RATIO
      FACT = FACT*RATIO
3620  IZOOM = 1
C
C CLEAR SCREEN
C
      CALL ERASE
      RETURN
C
C RESET TO AUTO-SCALE
C
3630  IZOOM = 0
      RETURN
      END
```

GRAPHICS MATHEMATICS

Screen Coordinates

The LINE and MOVE statements described above move the cursor in a "screen coordinate system." Usually, the bottom left of the screen is (0,0), with positive X to the right and positive Y in an upward direction. Typically, X and Y may range from zero to one thousand or so. Usually the screen is not square, but wider than it is high, so Y has a smaller range than X.

To draw a line from, say, (-50,-80) to (+1530, +2000), it is necessary to establish a screen scale other than the original screen coordinate system. This is done by multiplying all points (X and Y) by some screen scale factor, and adding to each point some X and Y value. In the example given, the actual X varies over a range of 1580. If the screen coordinate system varies X from 0 to 1,000, the line must be scaled to no more than .611 (1000÷1580) of the full size. Y varies over a range of 2080. If the screen coordinate system varies Y from 0 to 800 (as an example), then the line must be scaled to no more than .385 (800÷2080) of the full size. Of the two scale factors calculated, the Y value is more critical (smaller) and so must be used to scale both X and Y, thus maintaining the desired line shape, but at a size small enough to fit the screen. After scaling, the line endpoints are (-19.23, -30.77) and (589.46, 769.23). The first point is still off the screen past the lower left corner (0,0). To put the line back on the screen, we must add 19.23 to the X value of each point, and 30.77 to the Y value. Now the line goes from (0,0) to (608.69, 800). This exactly fills the screen from top to bottom. To see the ends of the line, a slightly smaller scale factor (say .35) should be used. This process is shown in figure 4.

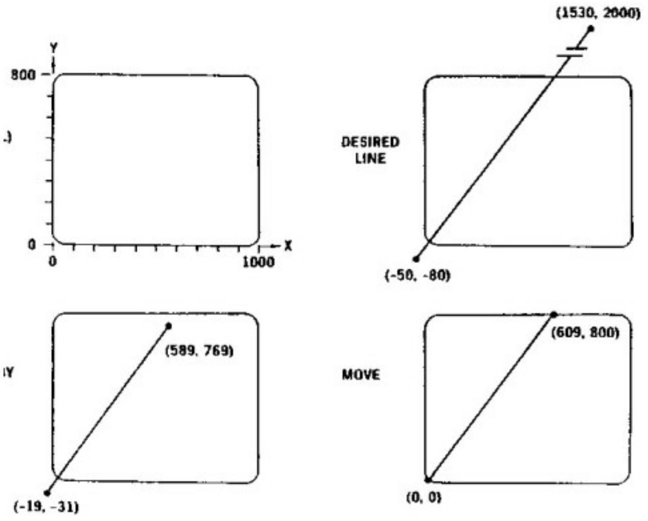


Figure 4. Screen Coordinate Transformations

If every LINE and MOVE statement for a drawing has the same scale factor and X and Y offsets applied, the resulting picture on the screen will be an exact, scaled image of the original geometry. This process is analogous to making a fortieth scale layout of an aircraft rather than using fifty-foot long drafting tables. As long as the scale factor is known, the actual size drawn is a matter of convenience.

If a drawing has been created on the screen in the manner described above, it is simple to zoom in on one portion of the drawing. This is done by using a larger scale factor than the one calculated, and use X and Y offset values to place the desired portion of the drawing on the screen. Note that a clipping command such as described previously must now be used.

3-D Mathematics

This procedure suffices for two dimensional drawings, but what about drawing a three dimensional body? If some object, such as a cube, is defined by lines connecting points in three dimensions (i.e., (0, 0, 5) to (0,5,5)) the two dimensional projections orthogonal to the cube's three-dimensional axis system can be easily drawn. This is done just by using two of the three coordinate values. For a side view, connect X and Z points (etc). This method isn't capable of orthographic projections or perspectives. These are accomplished with what are referred to as "homogeneous coordinate transformations."

Homogeneous coordinate transformations use four-by-four matrix multiplications to perform three-dimensional scale changes, translations, orthographic rotations, and perspectives. A dummy constant, "h", is used to increase the three-dimensional (X,Y,Z) points to four dimensions. Each (X,Y,Z) is multiplied by h to produce a point (hX,hY,hZ,h). The new point (X',Y',Z') created by the desired operation is found by dividing the points (hX',hY',hZ',h') by h', i.e.:

$$X' = \frac{hX'}{h'} \quad ; \quad Y' = \frac{hY'}{h'} \quad ; \quad Z' = \frac{hZ'}{h'}$$

where

$$[hX', hY', hZ', h'] = [hX, hY, hZ, h] \times \left[R_{4X4} \right]$$

R is the operating matrix, defined below, to produce the desired scale change, translation, orthographic rotation, or perspective.

The selection of h is entirely arbitrary, so it is usually selected to be one. In this case X = hX, Y = hY, and Z = hZ. Note however, that if h is selected to be zero, X, Y, and Z are infinite. Below we assume h is one, giving the R matrices for the four operations.

Note that scale and rotation operations can be performed with just the upper left three-by-three submatrices, since the last row and column are from the identity matrix.

Scale change $R = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ a is X scale factor
b is Y scale factor
c is Z scale factor
1 = no change

Translation $R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ e & f & g & 1 \end{bmatrix}$ e = X offset
f = Y offset
g = Z offset

Rotation about X (roll)

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma & 0 \\ 0 & -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Y (pitch)

$$R = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Z (yaw)

$$R = \begin{bmatrix} \cos\beta & \sin\beta & 0 & 0 \\ -\sin\beta & \cos\beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective along Z axis

d = distance

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective along Y axis

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1/d \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective along X axis

$$R = \begin{bmatrix} 0 & 0 & 0 & -1/d \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To obtain the two-dimensional rotated orthographic view of a three-dimensional body, multiply the (X,Y,Z) of every point successively by roll, pitch, and yaw rotation matrices. Note that the order in which these are done affects the final image, so the order must remain consistent. This results in new three-dimensional points such that a side view of the new points is identical to the view obtained if the original object were being observed in side view and then rotated through the angles used. Similarly, top and rear view of the rotated points show the view if the object were originally being viewed in top or rear view. Thus, a screen display using two of the three coordinate values for each point will produce the desired two-dimensional rotated orthographic view.

To produce a perspective, first produce the three-dimensional rotated points, then move the viewpoint along the axis appropriate to the desired view. For example, if the original position (no rotation) is selected to be the side view, perform the desired rotation, then use the Y-axis perspective R-matrix, and finally use the resultant X and Z values to create the two-dimensional screen image.

Figure 5 shows a display created by this technique from a three-dimensional data base. Note the vanishing point effect created by the homogeneous transformations.

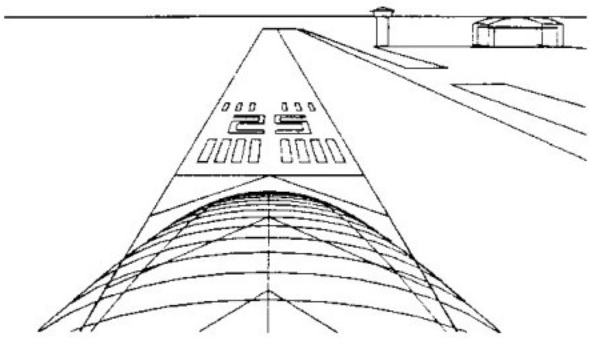


Figure 5. Perspective Display of Runway

Homogeneous coordinate transformations and their derivations are described in a number of computer graphics textbooks. An excellent and concise derivation can be found in the course notes for the University of Michigan's short course on Computer-Aided Design, by Dr. R. L. Phillips. Detailed treatments of computer graphics are available in "Interactive Computer Graphics", by W. Giloi (Prentice-Hall, New Jersey, 1978), and "Fundamentals of Interactive Computer Graphics", by J. Foley and A. Van Dam (Addison-Wesley Publishing Co., Mass., 1982). Another excellent reference is "Computational Geometry for Design and Manufacture," by Faux and Pratt (J. Wiley & Sons, N.Y., 1979).

GEOMETRIC REPRESENTATION

2-D vs 3-D

The preceding section assumes that you, in fact, know the endpoints of the lines that you wish to draw on the screen. If those lines are to produce a drawing of an aircraft, some form of geometric representation must be used. This can be two-dimensional (2-D) or three-dimensional (3-D).

A 2-D geometric representation emulates a piece of paper. Line endpoints* are stored as (X,Y) points on a drawing area. Side, top and rear views are created by separately specifying lines in each view. These lines have no inherent connectivity. An ignorant user could draw a square in side view, a circle in top view, and a triangle in rear view, and the computer wouldn't protest.

An improvement to this is the so-called "2-1/2-D" system. Here algorithms calculate from the first two views where the third view of a point must lie. However, the resulting point is still stored as an (X,Y) point on a drawing area. There is no 3-D surface, so cross-section cuts must be developed individually, as on a drafting board. The 2-D side and top view lines can also be used in a quasi-automatic fashion to assist in the construction of cross-sections if the appropriate programs are available.

In contrast, a 3-D system has no storage of separate views at all. (X,Y,Z) points, or equations to produce them, are stored. Desired 2-D views are produced automatically using the mathematics from the previous section. Because the 3-D surfaces are fully defined, cross-section cuts can be produced automatically.

The simplest type of 3-D geometry representation is the 2-D system with the addition of a depth or thickness value (or values). This is typical for production design applications, but not very useful for conceptual design.

A more useful 3-D representation is obtained by a mesh of (X,Y,Z) points lying on the desired surface. These points must have some ordering scheme to be useful. One simple scheme is to use an equal number of points in a series of parallel cross sections (Figure 6). This geometric representation is not very accurate, as the computer draws straight lines between the stored points, but is very rapid to develop and display.

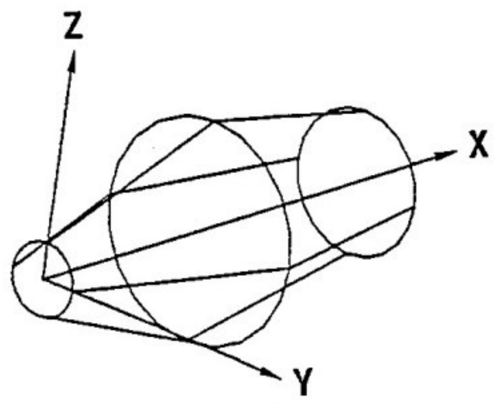


Figure 6. Surface Point Storage

*Or equations to produce line endpoints

Longitudinal Control Lines

Much improved surface definition can be obtained by storing a number of 3-D longitudinal control lines (Figure 7). These lines give points at any section location which are then used to automatically develop cross sections in a manner analogous to manual lofting. The longitudinal lines can be stored with a variety of techniques.

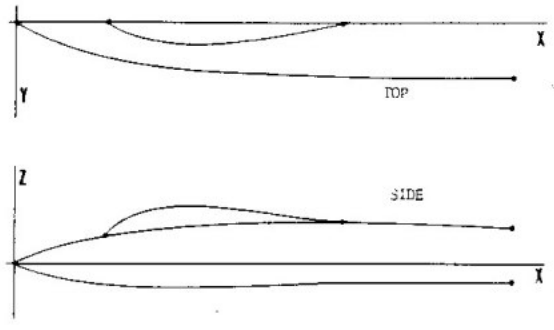


Figure 7. Longitudinal Control Lines

Simplest for the programmer and computer is the storage of lines in equational form. For example, the cubic equations below will fully define a line in three dimensions, and are rapid and easy for the computer to calculate given an x value. However, the coefficients A,B,C,D,E,F,G, and H have no direct meaning to the designer, and thus cannot be used as the input form.

$$Y = Ax^3+Bx^2+Cx+D$$

$$Z = Ex^3+Fx^2+Gx+H$$

A designer controls shape on the drafting board by specification of points and slopes, using the straight edge, protractor, french curve, and flexible spline. In a similar fashion, the cubic equation above can be controlled over a range by specifying a total of four points and slopes (first derivatives). For example, the coefficients could be generated from four points input by the designer with a light pen or tablet. Alternately, two points (the endpoints) and the slopes at those endpoints could be input.

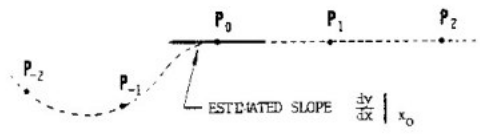
Note that this last approach prevents the designer from specifying any points within the curve-only at the endpoints. To provide control within the curve and still allow specifying slopes at both endpoints a curve of higher degree than the cubic (3rd degree) must be used. A fourth degree (quartic) curve allows specification of five conditions, such as endpoints (2), slopes at endpoints (2), and one point between the endpoints. A fifth degree (quartic) curve allows six conditions, such as endpoints (2), slopes at endpoints (2), an intermediate point, and the slope at that point. Alternately, one could use a quintic equation to control location, slope, and curvature (second derivative) at two endpoints.

Higher degree curves are, of course, possible, but run into problems of computational time and waviness. As a general rule-of-thumb, the lowest-degree curve that meets the specific applications requirements should be used.

There are other line definition techniques which can be used, such as "tension splines" or "b-splines." Descriptions of these can be found in the references listed on page 7.

Slope Estimating Techniques

In establishing longitudinal control lines, the designer may not always know the desired slope at a particular point. Instead, he may want to select the slope that gives a smooth curve connecting a number of points. On the drafting board, this is done visually. Hiroshi Akima of the U.S. Department of Commerce has developed a simple equation that mathematically replicates the "eyeball" slope estimation of an experienced draftsman. This equation uses two points on either side of the point for which a slope is to be estimated. The equation gives "nice" slopes producing smooth curves when combined with a curve equation such as the polynomials described earlier. One especially desired feature of Akima's equation, given below, is that it "recognizes" the situation in which three colinear points transit to a point off the line, and assigns a slope parallel to the line. This is shown in Figure 8.



$$\frac{dy}{dx}\Big|_{xo} = \frac{C_{-1}\,|C_2 - C_1| + C_1\,|C_{-1} - C_{-2}|}{|C_2 - C_1| + |C_{-1} - C_{-2}|}$$

where

$$C_{-2} = \frac{y_{-1} - y_{-2}}{x_{-1} - x_{-2}} \;;\; C_{-1} = \frac{y_o - y_{-1}}{x_o - x_{-1}} \;;\; C_1 = \frac{y_1 - y_o}{x_1 - x_o} \;;\; C_2 = \frac{y_2 - y_1}{x_2 - x_1}$$

Figure 8. Akima Slope Estimator

Robert Maier of Rockwell has developed a modified form of Akima's equation which provides a closer match to the slopes of continuous functions, although requiring more computational steps.

$$\frac{dy}{dx}\Big|_{xo} = \frac{C_{-1}\left[\frac{|C_2 - C_1|}{X_2 - X_o}\right]^{1/\pi}(X_1 - X_o) + C_1\left[\frac{|C_{-1} - C_{-2}|}{X_o - X_{-2}}\right]^{1/\pi}(X_o - X_{-1})}{\left[\frac{|C_2 - C_1|}{X_2 - X_o}\right]^{1/\pi}(X_1 - X_o) + \left[\frac{|C_{-1} - C_{-2}|}{X_o - X_{-2}}\right]^{1/\pi}(X_o - X_{-1})}$$

Either equation cannot be used for assigning slopes to the first two or last two points of a series of points. For these cases, a cubic can be fit using the first three (or last three) points with the slope at the third point as estimated using one of the equations above.

Geometry Storage

Whatever the techniques used to input points, slopes, or other conditions, there are two choices for storage of geometry. Either the calculations to produce equation coefficients can be performed as the user makes inputs, and the resulting coefficients can be stored, or the user's inputs (points, slopes, etc) can be stored, and coefficient calculations made only as the drawing is being created.

The first approach is quicker for drawing the finished geometry, but slower during design. The reverse is true of the second approach. The second approach is probably more compact in terms of data storage. For example, a cubic equation in three dimensions requires storage of eight coefficients (A through H) plus two X-values to indicate boundaries for a total of ten. If the same curve is stored as points and slopes, two (X,Y,Z) points (start and end) plus four slope values (dy/dx and dz/dx at both ends) must be stored, totaling ten numerical values. Note, however, that if a second cubic must be stored which smoothly follows a previous cubic, only five addition values must be stored in the second approach whereas nine new values must be stored for the first approach.

Parametric Representation

The equations to date have all been in "functional" form, i.e., one axis value such as Y is expressed in terms of another, orthogonal axis value such as X. This approach is intuitively simple and suffices for many applications. However, a functional representation cannot deal with a situation in which there is more than one Y value at a given X location. To deal with such a case, a "parametric" representation must be used.

A parametric equation relates axis quantities such as X,Y, and Z to a quantity called the "parametric variable", typically t. This quantity t represents distance along the curve measured from some arbitrary reference point (t = o). The units of t are arbitrary, thus typically the equations are written such that t varies from zero to one along the desired curve. (Figure 9)

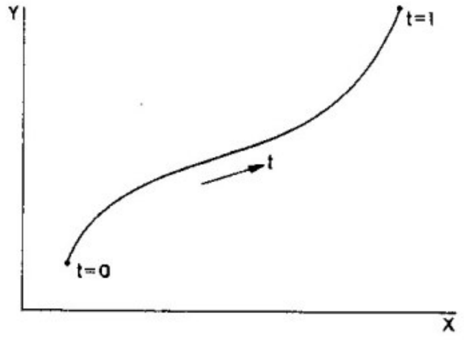


Figure 9. Parametric Curve

One nice feature of parametric representation is that it automatically bounds the curve, whereas in functional form one must store the desired first and last points to bound the curve.

The cubic equations used before to define a line in three dimensions are shown below in parametric form.

$$X = At^3+Bt^2+Ct+D$$

$$Y = Et^3+Ft^2+Gt+H \qquad \text{Note: } \frac{dy}{dx} = \frac{dy}{dt} \div \frac{dx}{dt}$$

$$Z = It^3+Jt^2+Kt+L$$

Note that since X, Y and Z are defined by separate equations, they can have multiple values. In fact, a parametric curve can loop across itself. Also, the range of t is arbitrary. Coefficients A through L are found as before, by specifying conditions such as points and slopes at various locations.

An alternate to equational representation of lines is the "Bezier-curve." P. Bezier of Renault has developed a parametric curve based solely on storage of points. These points act like magnets, pulling the curve towards themselves. The first and last points stored are the endpoints of the curve. This representation is very good for interactive manipulation by designers who can visually see the impact of moving points around (Figure 10). The equations are as follows, and rely on weighting values which are applied to the stored points.

$$X = \sum_{i=0}^{n} X_i J_i$$

$$Y = \sum_{i=0}^{n} Y_i J_i$$

$$Z = \sum_{i=1}^{n} Z_i J_i$$

where

(X_i, Y_i, Z_i) are coordinates of point i

n is number of points used

$$J_i = \frac{n!}{i!\,(n-i)!}\, t^i\, (1-t)^{n-i}$$

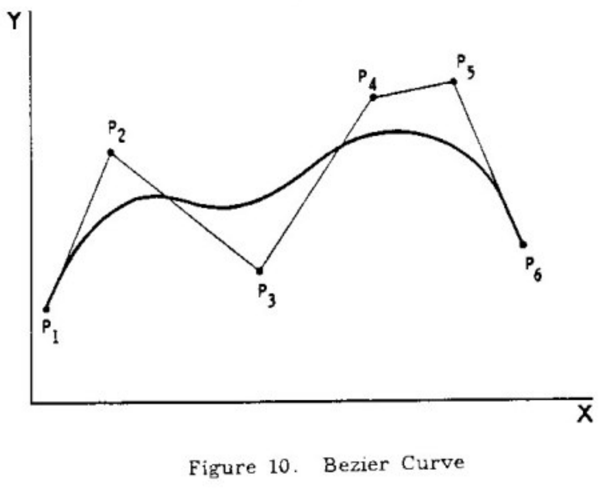
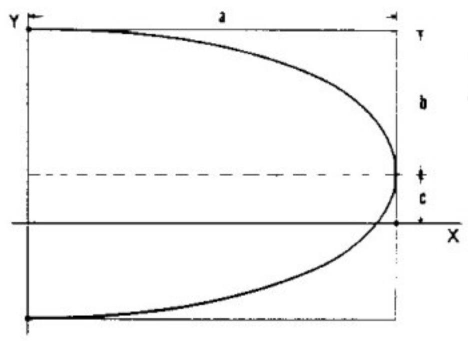


Figure 10. Bezier Curve

Note that a Bezier curve is of (n-1) order, i.e., if five points are used, this yields a fourth-order curve. Also, the Bezier curve is tangent to its endpoints along the line from the endpoint to the next adjacent point.

Cross Section Generation

With longitudinal control lines available, generation of cross sections can be done in a number of ways. Probably the simplest technique is to use three 2-D longitudinal control lines to give cross section values for upper centerline, lower centerline, and maximum half-breadth and then develop elliptical cross sections using the equations below. This is illustrated in Figure 11.

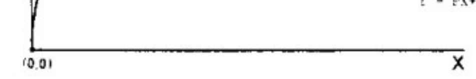


$$Y = c \pm b\sqrt{1 - x^2/a^2}$$

$$\text{where} \begin{cases} a = X_m \\ b = (Y_u - Y_l) \div 2 \\ c = Y_u - b \end{cases}$$

Figure 11. Elliptical Cross Section From Three Values

While this may be suitable for a "quicky" design program, it is obviously too limited for serious work. A substantial improvement is obtained by using three 3-D longitudinal lines controlling the endpoints of two conic curves in each cross section, assuming that the cross section slopes are "square" (see Figure 12) and using a constant ρ value*. Conic equations are shown below. Note that if ρ is chosen as (.4142), a circle will result when the endpoints are equidistant from the tangent intersection.



$$D = (1-\rho)\frac{A}{2}$$

$$E = (1+\rho)\frac{B}{2}$$

$$K = (E - BD/A)^2/D(B-E)$$

$$S = BK$$

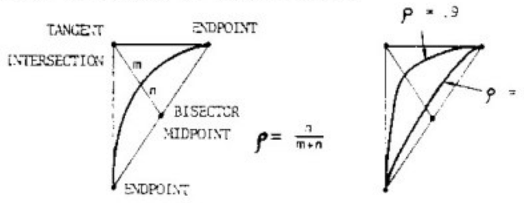
$$R = (K/4 - B/A)K$$

$$P = -(K/4 - B/A) - K/4$$

$$Y = PX + \sqrt{RX^2 + SX}$$

Figure 12. Conic Equation

*ρ value is defined as shown below:



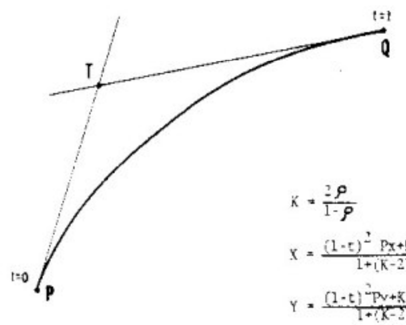
Note that this assumes the curve starts from the point (0,0), which is unlikely. To translate the curve to a starting point (X_0, Y_0), subtract X_0 from A and Y_0 from B, calculate P, R, and S, then use

$$Y = P(X-X_0) + \sqrt{R(X-X_0)^2 + S(X-X_0)} + Y_0$$

I.e., we have substituted $(X+X_0)$ and $(Y+Y_0)$ for (X) and (Y) in the defining equation.

More flexibility can be obtained by allowing ρ to vary from nose to tail. This requires a 2-D longituding control line for each conic. Alternately, the conic shape could be controlled not by ρ value, but by an actual point (D,E) on the surface, allowing the first two calculations in Figure 10 to be skipped. A 3-D longitudinal control line is required for each conic to specify (D,E).

A generalized, parametric conic is shown in Figure 13. Here slopes are arbitrary and t, the parametric variable, varies from (0) to (1). Simple geometry is used to determine the location of T, the tangent intersection, from endpoints P and Q and desired slopes. This curve requires 3-D longitudinal control lines for P, Q, and T, and a 2-D longitudinal control line for p.



$$K = \frac{2\rho}{1-\rho}$$

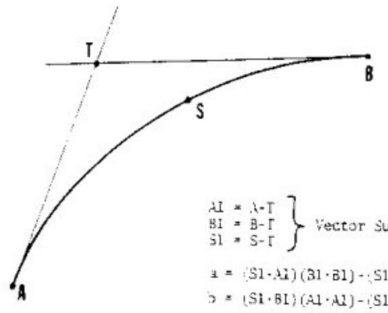
$$X = \frac{(1-t)^2\ Px+Kt(1-t)Tx+t^2Qx}{1+(K-2)(t)(1-t)}$$

$$Y = \frac{(1-t)^2Py+Kt(1-t)Ty+t^2Qy}{1+(K-2)(t)(1-t)}$$

Figure 13. Parametric Conic Using ρ

As before, an alternate approach uses a point on the curve instead of p. This is shown in Figure 14. Here 3-D longitudinal control lines are required for A, B, S, and T, for each conic in the cross section.

X is a vector. To find X_x and X_y for a given t, use either the X terms or the Y terms of the vectors A, B, and T to find Q, then again use either the X terms or the Y terms of the vectors A, B, T, and Q in the X expression. This was spelled out in the last example.



$$\left.\begin{array}{l} A1 = A-T \\ B1 = B-T \\ S1 = S-T \end{array}\right\}\text{ Vector Subtraction}$$

$$\left.\begin{array}{l} a = (S1\cdot A1)(B1\cdot B1)-(S1\cdot B1)(A1\cdot B1) \\ b = (S1\cdot B1)(A1\cdot A1)-(S1\cdot A1)(A1\cdot B1) \\ c = (A1\cdot A1)(B1\cdot B1)-(A1\cdot B1)(A1\cdot B1) \end{array}\right\}\text{ scalors produced by dot products}$$

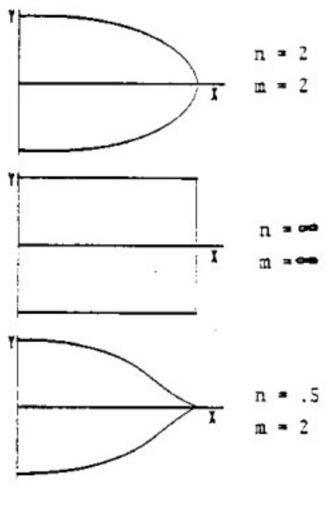
$$v = \frac{c-a-b}{2\sqrt{ab}}\ \Big\}\text{ scalar}$$

$$Q = \left(\frac{1}{1+v}\right)\left(\frac{A+B}{2}\right)+\left(\frac{v}{1+v}\right)\left(T\right) \qquad \text{Q is a vector}$$

$$X = \frac{2[A-(1+v)Q+B]t^2+[-3A+2(1+v)Q-B]t+A}{2(1-v)t^2-2(1-v)t+1}$$

Figure 14. Parametric Three-Point Conic

As in the case of longitudinal lines, cubics or higher degree equations can be used for cross section definition from point and slope values produced by longitudinal control lines. Another cross section representation with desirable features is the superellipse. This equation has the advantage of producing anything from a circle or ellipse to a nearly-perfect square, from only one equation, as shown in Figure 15.



$$\left(\frac{X}{a}\right)^n + \left(\frac{Y}{b}\right)^m = 1$$

Figure 15. Super Ellipse

Surface Patches

A more sophisticated technique than the longitudinal control line is the surface patch. A surface patch is a set of equations defining a surface mathematically within a bounded region, hence the name "patch." Surface patches are usually parametric, and for convenience the patch is usually bounded between zero and one in the two parametric directions (see Figure 16).
Surface patches allow greater flexibility of design than longitudinal control line methods, offer better control of smoothness, and are simpler for calculation of geometric derivatives such as slope and curvature in arbitrary directions.

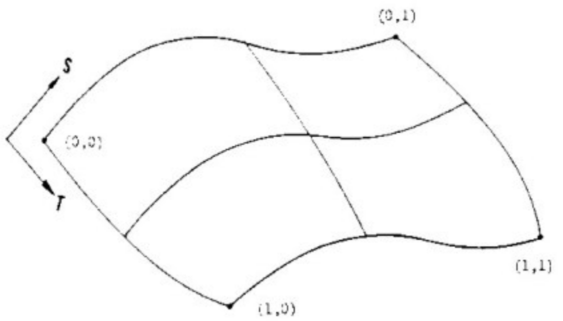


Figure 16. Parametric Surface Patch

A surface patch can be developed with equations of any degree. "BiCubic" patches are common. Rockwell uses a special "BiQuartic" patch which features a Bezier-like representation. "BiQuintics" and higher can be used. Again, the higher-degree the equation, the better the control over slopes and curvature, but the longer the computation time and the more waviness encountered.

A special patch that is well known is the Coons-patch. This parametric patch provides positional and slope continuity across the boundary to adjacent patches, and also has terms explicitly dealing with twisting of the surface along the boundary curves. The Coons-patch is represented solely by four boundary curves preventing explicit control of the interior of the patch. For a detailed treatment of the Coons Patch, see "Surfaces for Computer-Aided Design of Space Forms", MIT Project MAC TR-41, June 1967, by Steven A. Coons.

Patches typically have the same problem encountered with longitudinal line equations; the defining coefficients aren't very meaningful to the designer. This can be solved in much the same manner, by allowing the user to specify certain conditions, such as points, slopes, or curvatures. Similarly, the Akima equation can be used to establish slopes from a series of points. Patch equations of various forms can be found in the references given on page 7.

Solid Modelling

"Solid Modelling" is a term frequently encountered today in computer-aided design, and can have several meanings. The "classical" (i.e., four years old!) definition of solid modelling is based on the use of built-up "primitives." These primitives are simple solid bodies such as spheres, cylinders, and cubes which are assembled together to make an object. The primitives can be positive or negative (i.e., a hole), and are summed using Boolean algebra.

This technique has important advantages in terms of computation speed for tasks such as hidden line removal or radar corss section estimation because the properties of the surface, such as surface normal or curvature, are known in advance. On the other hand, it is limited to fairly simple shapes, and precludes exact representation of complex objects such as a highly blended wing.

Recently a broader definition of "solid modelling" has been applied. This defines a solid model as any geometric representation which has a single, unambiguous real-world counterpart. This encompasses the "constructive solid geometry (CSG)" described above, and also includes boundary representations, such as the surface patches described earlier, and other forms such as "sweep" representations in which a 2-D outline is extruded through space in some fashion to form a solid. Thus, solid modelling actually relates to almost any representations now in use, except for the simplest wire frame technique such as shown before in figure 6. Note that the computation speed advantages attributed to constructive solid geometry do not apply to other "solid models" such as the boundary representation.

Local Axis Systems

The previous discussion has implicitly assumed a single "global" XYZ coordinate system. This can be quite cumbersome, for it requires changing every coordinate point of an object (such as a wing) if some translation or rotation is desired. Also, the large numbers required in a global-only system to define an object far away from the origin cause a loss in the number of available significant digits for accurate positioning. This is especially troublesome in iterative operations. A much more flexible system is the use of a separate local axis system for each different object, shown in Figure 16.

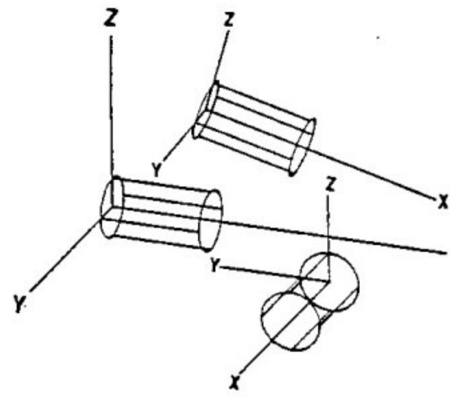


Figure 17. Local Axis Systems

The homogeneous coordinate system described earlier is used for this. Each separate object, such as a wing or engine, has values stored with it for X,Y,Z, and roll, pitch, and yaw. When the object is drawn, the roll, pitch, and yaw terms are used in the homogeneous rotation matrix equations, then the X,Y, and Z values are added to each point. These initial steps place the object in the correct location and orientation within the overall entity to be drawn (i.e., the airplane), then the homogeneous transformations are again used to create the desired display.

Frequently in design problems the desired orientation is not explicitly known in terms of roll, pitch, and yaw of an object. In such cases, it is necessary to allow those values to be determined from a desired orientation specified by the designer. This is done with the aid of direction cosines. If an object has an orthogonal local axis system, the following equation converts points on the object into the global axis system.

$$(X,Y,Z)_{global} = (X,Y,Z)_{local} \times \left[R_{3x3}\right]$$

where

$$R = \begin{bmatrix} L_X & L_Y & L_Z \\ M_X & M_Y & M_Z \\ N_X & N_Y & N_Z \end{bmatrix}$$

where the local axis system is specified by three known points; P_1, the axis origin, P_2, a point on the local X-axis, and P_3, a point on the local XY plane. L, M, and N terms are calculated as follows:

$$L_x = \frac{X_2 - X_1}{g};\ L_y = \frac{Y_2 - Y_1}{g};\ L_z = \frac{Z_2 - Z_1}{g}$$

where

$$g = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}$$

$$N_X = \frac{R_X}{h};\ N_Y = \frac{R_Y}{h};\ N_Z = \frac{R_Z}{h}$$

where

$$R_X = (Y_2 - Y_1)(Z_3 - Z_1) - (Y_3 - Y_1)(Z_2 - Z_1)$$

$$R_Y = (Z_2 - Z_1)(X_3 - X_1) - (Z_3 - Z_1)(X_2 - X_1)$$

$$R_Z = (X_2 - X_1)(Y_3 - Y_1) - (X_3 - X_1)(Y_2 - Y_1)$$

and

$$h = \sqrt{R_X^2 + R_Y^2 + R_Z^2}$$

$$M_X = N_Y L_Z - N_Z L_Y$$

$$M_Y = N_Z L_X - N_X L_Z$$

$$M_Z = N_X L_Y - N_Y L_X$$

But this is the same equation as used in the homogeneous rotational transformations (h=1) described earlier, so the terms of the R matrices must be equal. To take advantage of this, the roll, pitch, and yaw transformation matrices must be combined to a single matrix. This requires selection of the order in which these matrices will be applied. If they are combined first as roll times pitch, and then times yaw, the resulting R matrix is formed:

$$R = \begin{bmatrix} [\cos\alpha\ \cos\beta] & [\cos\alpha\ \sin\beta] & [-\sin\alpha] & [0] \\ [(\sin\gamma\ \sin\alpha\ \cos\beta)-\cos\gamma\ \sin\beta] & [(\sin\gamma\ \sin\alpha\ \sin\beta)+\cos\gamma\ \cos\beta] & [\sin\gamma\ \cos\alpha] & [0] \\ [(\cos\gamma\ \sin\alpha\ \cos\beta)+\sin\gamma\ \sin\beta] & [(\cos\gamma\ \sin\alpha\ \sin\beta)-\sin\gamma\ \cos\beta] & [\cos\gamma\ \cos\alpha] & [0] \\ [0] & [0] & [0] & [1] \end{bmatrix},$$

Equating terms to the previous matrix gives nine equations (below) in three unknown variables.

$$\cos\alpha\ \cos\beta = L_X$$
$$\cos\alpha\ \sin\beta = L_Y$$
$$-\sin\alpha = L_Z$$
$$\sin\gamma\ \sin\alpha\ \cos\beta - \cos\gamma\ \sin\beta = M_X$$
$$\sin\gamma\ \sin\alpha\ \sin\beta + \cos\gamma\ \cos\beta = M_Y$$
$$\sin\gamma\ \cos\alpha = M_Z$$
$$\cos\gamma\ \sin\alpha\ \cos\beta + \sin\gamma\ \sin\beta = N_X$$
$$\cos\gamma\ \sin\alpha\ \sin\beta - \sin\gamma\ \cos\beta = N_Y$$
$$\cos\gamma\ \cos\alpha = N_Z$$

From the equations we obtain:

$$\alpha = \sin^{-1}(-L_Z)$$
$$\beta = \cos^{-1}\left(\frac{LX}{\cos\alpha}\right) = \sin^{-1}\left(\frac{Ly}{\cos\alpha}\right)$$
$$\gamma = \sin^{-1}\left(\frac{MZ}{\cos\alpha}\right) = \cos^{-1}\left(\frac{N_Z}{\cos\alpha}\right)$$

Either of the two possible values of α are correct. β and γ are uniquely determined once α is selected.

The required X, Y, and Z local axis origin values are the coordinates of P_1, which were subtracted from the coordinates of P_2 and P_3 to get a pure rotation problem prior to calculation of L, M, and N terms.

There is a trivial case to consider. If $\cos\alpha = 0$, there is no solution by this method. This occurs when the local X' axis aligns with the global Z axis. This is indicated when $|LZ| = 1$ and $Lx = Ly = 0$, thus $\alpha = (-L_Z) \times \pi/2$.

As we have selected rotation order of roll, then pitch, then yaw, it follows that when pitch is $(-\pi/2)$, the roll and yaw must sum to the total angle θ between the local Y' axis and the global Y axis. If the pitch is $+\pi/2$, the yaw has negative sign, thus;

$$\beta - L_Z\ \gamma = \theta, \text{ where} \tag{46}$$

$$\theta = \sin^{-1}(-M_X) = \cos^{-1}(M_Y) \tag{47}$$

Any combination of β and γ meeting equation 46 is correct.

In the preceding calculations for L, M, and N terms, it is wise to test that those terms do not exceed by some small amount the allowable range of minus one to positive one. Also, if a term is very close to zero, one, or minus one, set it to exactly that value. This minimizes difficulties arising from computer precision.

This capability can be greatly expanded through the use of geometric relationships to establish desired orientations. For example, rotations about some point can be made in the global axis system by selecting three points on each object (such as (0,0,0), (0,0,1), and (1,0,0)) and rotating them as desired, then calculating the resultant X,Y,Z, roll, pitch, and yaw. This can also be used to swing objects about some trunion axis such as shown in Figure 18. Description of these techniques is beyond this paper, but they are reasonably easy to derive from a foundation in analytical geometry. Be advised that the trivial cases (i.e., divide by zero) are waiting to trap the unwary.

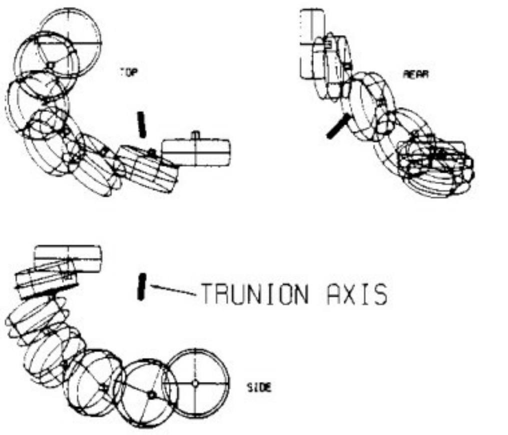


Figure 18. Wheel Rotation About a Trunnion

ANIMATION

Animation is emerging as a potent future tool for aircraft design. It offers capabilities to visualize kinematics, time-dependent analysis, and fourth-dimension solution spaces. Also, animation is an excellent briefing tool.

Classical animation is done one frame at a time. Cartoonists draw Mickey in a pose, then draw another picture with him moved just a fraction of an inch, and so-on. The same process is used in computer animation.

Using the homogeneous coordinate transformations, it is easy to roll an object around in any direction. Also, the shape of any object can be altered on the computer. If local axis systems are employed, the relative positioning of objects can be easily varied. Figure 19 shows an ejection sequence "animated" in this fashion.

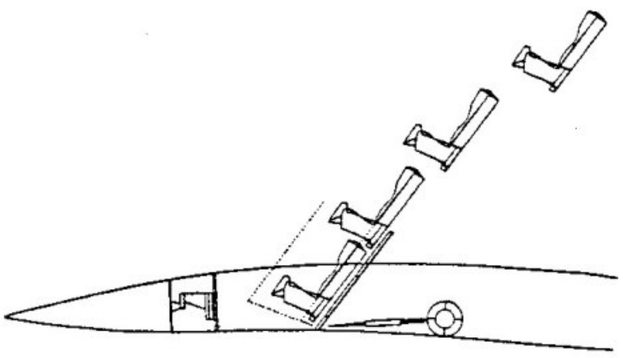


Figure 19. Ejection Sequence

To perform computer animation, a program is written which makes any or all of these motions in very small incremental steps. When each frame is drawn, the image is captured on either film or videotape and the computer draws the next frame.

Frames can be taken with special devices sold for that purpose. It is also possible to use a regular terminal and a motion picture camera capable of single frame exposure.

This requires a bit of special circuitry*. The computer must be able to signal the camera to take a picture. This can be done by tying in to the terminal pin used to trigger a hardcopy, or it can be done by painting a dot in a corner of the screen which is sensed by a taped-on photocell. The camera then shoots a frame and advances itself. The next frame can then be drawn. Note that some storage tubes automatically shift the screen coordinate system after each erase to prevent "wearing" a spot into the screen. This requires a number of erase commands (eight on Tektronix scopes) to cycle the coordinate system back to its original location. Otherwise, the animated image will jump around a small but annoying amount.

With current computer graphics equipment, real-time animation is extremely expensive. However, sophisticated aircraft simulators are capable of producing amazingly realistic animated images which respond in real time to both pilot inputs and the inputs of the instructors. These use special techniques for rapid shading, coloring, and hidden-line removal, including the use of microprocessors dedicated to those tasks. In the future these capabilities should be available for computer-aided design.

INTERACTIVE ANALYSIS

As promised, this paper has focused on design aspects of the use of computers. However, interactive analysis plays an important part in an integrated design system. Interactive analysis includes geometric analysis such as volume and wetted area, as well as specific applications analysis such as wave drag or radar cross section.

There are two broad types of analysis programs: those that use actual three-dimensional geometry, and those that use scalar quantities extracted from the geometry. In the first category fall geometric calculations such as volume and wetted area (Figure 20) as well as supersonic wave drag, radar cross section, and other calculations in which the vehicle's actual shape is used. In the latter category falls calculations such as statistical weight estimation, most first-order aerodynamic methods, and tail sizing by volume coefficients. These later cases typically require additional interaction by the user prior to analysis. As an example, it may be necessary to indicate the part of the fuselage length corresponding to avionics bays prior to a statistical weight estimation.

*See "Tekniques" Vol. 1, No. 5, Published by Tektronix (page 5)

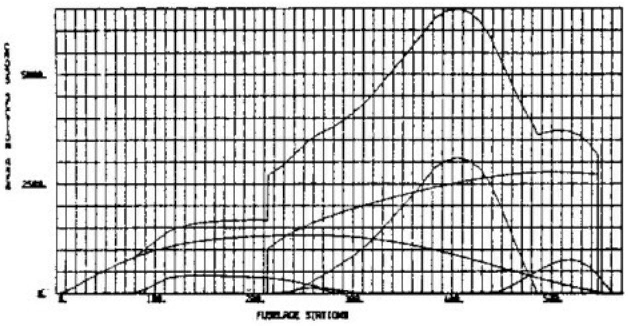


Figure 20. Volume Distribution Analysis

The purpose of interactive analysis is, of course, to guide the designer. To accomplish this, the analysis results must be presented in a fashion which permits some "preferred path" change to be recognized. For example, a supersonic wave drag evaluation could indicate which portions of the vehicle contribute the most to drag. Aerodynamic center could be compared to center of gravity to allow wing relocation or internal component rearrangement. Therefore, these routines _must_ be "on-line", i.e., they must be executed at _the_ terminal by the designer and produce results on the screen within a reasonable time (less than five minutes is desirable).

The programming for interactive analysis can use any of the user-interface techniques described earlier. For example, menues and prompts can be used to lead the user through the particular analysis. Alternately, an analysis command language could be devised, or some mixture of the two. Cursor and light pen devices can be used to interact with the program, along with function keys or other devices mentioned earlier.

Data can be displayed tabularly or as 2-D graphs, or even as three-dimensional surfaces. Color, if available, is extremely useful for clear presentation of data.

For interactive design applications, analysis programs must have a great deal of interactive capabilities. Input data sets must be capable of being edited one item at a time, then re-analyzed. Even more powerful capabilities can be developed by specifying a range for some input variable(s), and producing a plot of the impact of the changes.

Computer-aided aircraft design without interactive, on-line analysis of the major vehicle drivers only scratches the surface of the computer's capability for improving design productivity.

INITIAL GRAPHICS EXCHANGE SPECIFICATION (IGES)

One critical requirement for a usable computer-aided aircraft design capability is the ability to interface data both to and from analysis programs and other design programs. There are two ways to accomplish this.

The "brute-force" approach is to write a separate interface program to convert to and from the data format for each program communicated with. This is a good approach for a limited number of interfaces, as it requires conversion of only those data required and can be tailored to handle specific problems. However, it is too cumbersome for interfacing between a large number of programs. Also, it is difficult to patch an additional program into the network.

Another approach is to have each program convert its geometric data into a standard, or "neutral" format. This may be more difficult to program initially, but leads to large downstream savings as the number of interfaced programs grows.

The National Bureau of Standards is developing such a neutral format, called the Interactive Graphics Exchange Specification (IGES). This format handles a variety of geometric entities such as lines, points, and arcs, as well as text and related arrows, labels, etc. Currently, the IGES is more oriented to 2-D CAD drawings, but new releases of IGES are incorporating parametric surface patches and other capabilities desired for conceptual design geometry.

IGES is being implemented by virtually all CAD/CAM vendors. In the future IGES should allow direct data exchange between widely different programs, including the vital transition from conceptual CAD to detail-design CAD/CAM.

SUMMARY

This paper has attempted to encapsulate, in 15 short pages, a large and dynamic field. The basic concepts of computer graphics have been introduced, with emphasis on application to the area of aircraft design. As mentioned earlier, the focus has been on conceptual design rather than detailed parts design.

The attached article summarizes a design system developed at Rockwell and tailored towards conceptual design, called the Configuration Development System (CDS). In reading the article, note the use of features described above such as a hybrid command language employing prompts for analysis routines. Also note that CDS was developed for computers using storage tube terminals (permitting display of very complex designs), which strongly influenced the user interface.

Following the article are briefing charts which illustrate, in some detail, creation of a typical aircraft design on CDS. Note the interaction between design and analysis. All calculation shown are interactive, and take under three minutes each.

ACKNOWLEDGEMENTS

The author extends sincere appreciation to the following people, without whose advice and "sharp red pencils" this paper would not have been possible: James Berry, Mark Killian, Robert Maier, James Klimmek, Charles Holt, and Kathie Raymer. Also, thanks is given to Marty Trent for the artwork of Figure 3.