An Overview of Interactive Computer Graphics and its Application to Computer-Aided Engineering and Design

Andries van Dam
Department of Computer Science
Brown University

1. Introduction

While the physics community is well acquainted with the use and benefits of batch (plotter) graphics, interactive computer graphics is a relatively new field. Until recently it was an esoteric specialty involving either low-power, relatively inexpensive Tektronix storage tube technology or high-performance but expensive refresh hardware, substantial computer resources, and idiosynchratic, non-transportable software. In the last few years, spectacular improvements in hardware price/performance and in understanding of the techniques for building high-level, transportable applications programs have made graphics an increasingly common technique for illustrating natural and man-made phenomena, and for designing a variety of systems and structures in the various branches of physics, chemistry and especially engineering. The purpose of this brief birds-eye view of interactive graphics is to list the key ideas, and to show how one of the most important application areas, Computer Aided Engineering/Design takes advantage of it. To get a more detailed understanding, the reader is referred to [FOLE81] and [MACH80] for fundamental references in the two fields respectively.

2. Graphics Hardware

2.1. Output

The most common medium today (and for the immediate future) is based on high-bandwidth CRT technology, although some advances in solid state panels such as x-y coincidence addressed plasma panels are making inroads in medium resolution displays.

Two modes using refresh CRTs are common, the older *vector* (also called stroke, steered beam or random scan) technology and the newer *raster* technology. In the former the beam can be deflected from any point to any other on the two-dimensional grid, called the *raster*, both to move it to a new location (MOVE, with the electron beam blanked) and to create long strokes for vectors (DRAW, with the beam unblanked) or short strokes for characters (see Fig. 1 and Fig. 2). The advantage of vector technology is that straight lines can be drawn continuously, crisply and thinly, from any point to any other, typically on a high-resolution, 4096x4096 raster. Furthermore, with special transformation hardware, it is possible to translate, rotate, and scale so-called *wireframe* line drawings in real time, in response to input from an input device such as a "3 degrees of freedom" joystick. Such dynamics makes possible kinaesthetic feedback which is of great value in understanding the 3-dimensional extent of objects such as "data clouds" and other scatter diagrams, mechanical objects such as structures, chamber/detector geometry, etc.

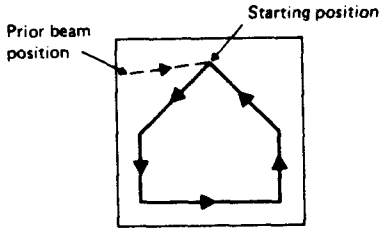


Fig. 1: House outline displayed with random scan.

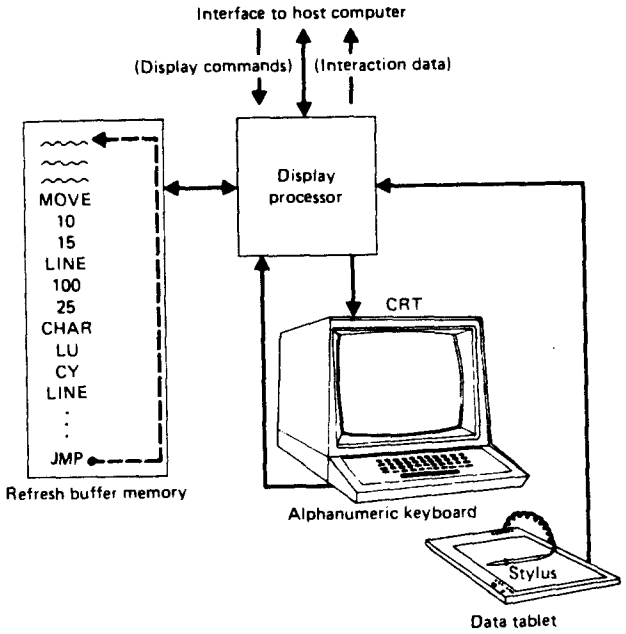


Fig. 2: Typical refresh display device. *Display* list in memory shows symbolic representation of plotting commands followed by values (e.g., x, y coordinates or characters).

While the more expensive vector refreshed display allow modulation of beam current to affect intensity modulation, solid areas are nearly impossible to obtain by shading with many closely-spaced vectors. This is because of the cyclical refresh required to re-activate the phosphor screen. The vector generator can move the beam only so many vector-inches per millisecond, and if there are more to be drawn during a refresh "frame" time of between a 60th and a 30th of a second, the display will flicker, greatly reducing the comprehensibility and effectiveness of the display and tiring the operator.

In raster technology, instead of permitting a random deflection of the beam, a TV-like raster scan is used (see Fig. 3 and Fig. 4). For high-resolution displays typically an odd/even interlace is used, exactly as in commercial TV. Also as in TV, the raster scan is independent of the number of raster elements (conventionally called *pixels*, short for picture elements) that are "turned on",

thereby eliminating flicker and allowing solid (even colored) areas. The disadvantage is that strokes can no longer be made continuously by deflecting the beam between endpoints on the raster, typically passing in between intermediate points and must now be constrained to lie only on raster points. Thus the line is discretely sampled by pixels lying on the raster. This leads to the analog of the "Calcomp staircase" phenomenon of jagged edges known since the days of plotters. While such "jaggies" are highly objectionable on low and medium resolution displays (less than or equal to 512x512 pixel resolution and less than 50 pixels per inch), they are much less noticeable on the higher-resolution screens now commonly offered on professional workstations (1024x768 with at least 72 pixels per inch in one dimension being typical). Indeed on such crisp, high-resolution screens, a page of multi-font, proportionally-spaced, hyphenated and right-justified text with embedded line and tone art is a more than adequate substitute for typeset hardcopy[MEYR82]. Alternatively, several manufacturers are using on-the-fly intensity shading (so-called anti-aliasing) to blur the sharp staircase edges of both vector and character pixel boundaries, which from a distance gives the appearance of smoother edges. This requires more memory to store intensity values, however, but requires less bandwidth than the higher-spatial-resolution displays do.

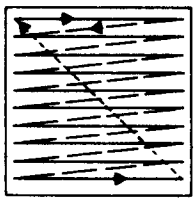


Fig. 3: Raster-scan pattern.

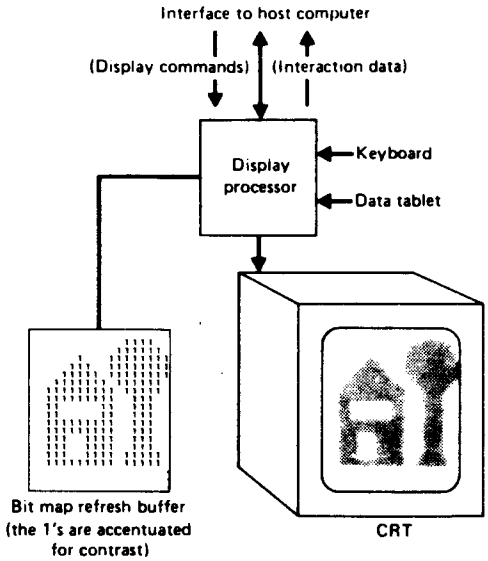


Fig. 4: Typical raster graphics display showing house and tree.

More commonly than anti-aliasing hardware are special processors ("engines") typically made from bit slices which do *scan conversion* from the

encoded digital representation of lines, polygons and characters to pixels in the bit map from which the raster scan is refreshed: from two endpoints of a line to the pixels approximating the line or from polygon vertices to the interior pixels. Equally common are special purpose processors which do "bit-blt" (bit boundary block transfer) or "RasterOp" (raster operation). This operation moves rectangular regions of the bit map in which the pixels are stored, thereby scrolling corresponding portions of the image on the screen containing a piece of text or some graphical output of a process. The full RasterOp allows logical combinations such as OR and XOR between source and destination rectangles. A region of the screen containing a logical unit of information as the output of a process is commonly called a *window* in the text editing/word processing community. Conversely, in the graphics community, window refers to a rectangular portion of the application program's coordinate space (the "world" or "user" coordinate system, typically in floating point), while the rectangular portion of the screen to which the contents of the window are mapped is called a *viewport*. *Clipping* refers to discarding (pieces of) picture elements lying outside the window when the window to viewport mapping is done (see Fig. 5). Often clipping is done in the hardware, after the geometric transformations have been done for vector displays, and as part of scan conversion for raster displays.

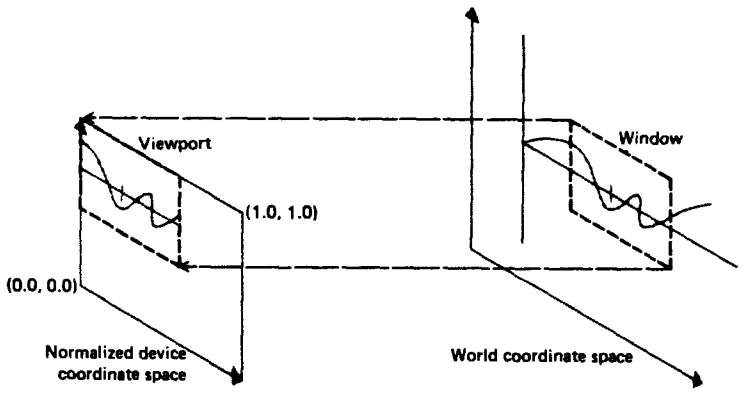


Fig. 5: Mapping of window (world coordinates) to viewport (normalized device coordinates), including clipping.

2.2. Input

The essence of interactive graphics is, of course, the interactive creation and modification of images using a variety of tools. Among these are devices which are *sampled* by the display processor (control dials, cursor-moving data tablets and digitizers, joysticks, trackballs, thumb wheels, etc.) and devices which cause *interrupts* (light pens used to detect a primitive on the screen and alphanumeric and function keyboards which transmit character codes). It is typical for a device which can be used to drive a cursor such as a data tablet or a "mouse" (a hand held device which is rolled on a flat surface to turn x and y direction potentiometers) to have integral buttons which can be used to interrupt the CPU, either to transmit a cursor position or to pick an element on the screen so as to identify it to the program. For example, in most modern raster graphics based *workstations* (powerful microprocessor based autonomous computers connected in a local area network with file and printer servers[MEYR82]) the tablet or mouse is used to position the cursor and select graphical symbols called "icons" from a menu to enter commands. This iconographic interface is

much easier and faster than having the user typing in commands as on traditional alphanumeric displays attached to timesharing systems. [SMIT82].

3. Device-Independent Graphics Software

3.1. Output

The major thrust of the last decade of software development in graphics has been to define machine- and device-independent graphics packages to eliminate manufacturer lock-in by allowing applications programs (and programmers) to be transportable. The 3-D ACM SIGGRAPH Core Graphics package [GSPC79] and its ISO 2-D derivative, GKS (Graphical Kernel System [ISO81]), are the most prominent examples of such packages now enjoying wide acceptance. In effect such a graphics package virtualizes both graphic input and output much as FORTRAN and its successor high-level programming languages virtualized computer instruction sets and input/output devices. Also, in the same way that FORTRAN compilers often don't make use of fancier instructions found on many computers, the graphics package may not take maximum advantage of individual capabilities of a particular display system, in order to preserve transportability.

Fig. 6 shows that the application programmer defines an application-dependent data structure, the application *model*, and creates and modifies it with the application program. This program then calls on the graphics package, a set of subroutines rather than a graphical language for the sake of ease of implementation and compatibility with procedural languages, to map the data structure to a "view" on the screen. Notice the parallel to data base terminology where the user also has access to views of the data base. The graphics package clips all the *primitives* (points, lines, polygons, text strings) passed by the application program to the window boundaries and performs the window to viewport mapping. The viewport is typically expressed as a subset of the [0.0,1.0] x and y intervals bounding the virtual screen, in so-called *normalized device coordinates*. These are then mapped to the particular display device's physical coordinates (usually an integer raster coordinate system) by the package's device driver as part of compiling device-dependent display instructions. Note that for 3D applications the window to viewport mapping is generalized to clipping and projection from a 3D *view volume* (perspective cone or parallelepiped) to the viewport.

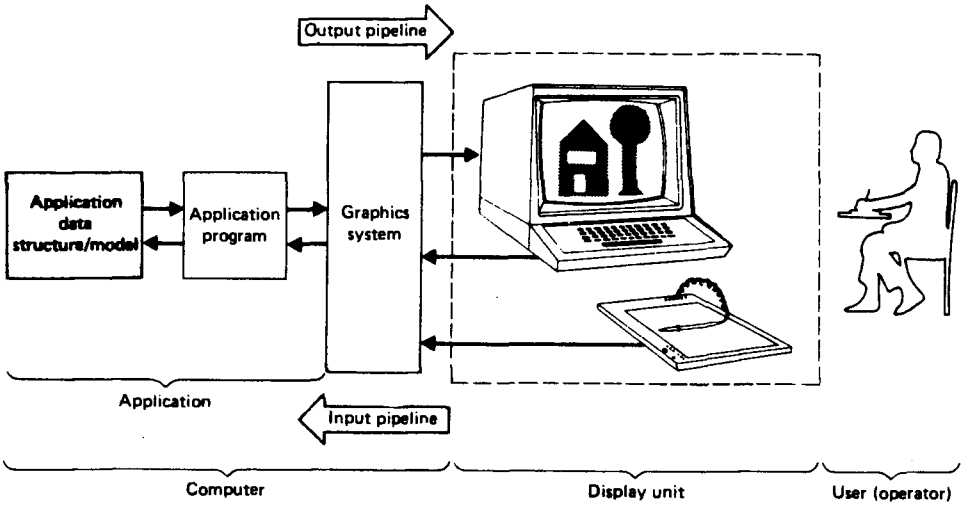


Fig. 6: The programmer's conceptual model of interactive graphics.

As the first example, we show below a simple graph plotting program which uses a plotting procedure to draw a x-y graph, with a marker denoting each data value on a set of coordinate axes (for which the obvious code is not shown). Note that procedure SAMPLE_PLOT first sets up a window in the world coordinate system, then a viewport on the virtual screen in normalized device coordinates. Next a *segment* is created, a named collection of primitives which can be manipulated as a unit with the DELETE_SEGMENT operation (or be toggled between visible and invisible or pickable and not pickable). Roughly speaking, the segment groups a collection of primitives as a procedure groups a collection of programming language statements. These primitives are defined by moving a point in the world coordinate system, analogous to drawing primitives on a plotter by moving the pen; this point is called the CP, for Current Position. In the Core package, primitives are defined in the world coordinate system from the previous CP to the specified endpoint, saving the application programmer the work of specifying both endpoints explicitly. In the ISO standard, GKS, there is no CP which the programmer must remember - all coordinates are specified explicitly.

Primitives and segments both may have *attributes*, which typically control their appearance. The linestyle attribute of a line, for example, could be used to control whether the line appears thick or thin, solid or dashed, with or without arrow heads, etc. Similarly there may be intensity and/or color attributes. The TEXT command defines the character string passed as argument starting at the CP, with a default size and font. Sophisticated graphics package allow many parameters which define intercharacter spacing and other typographic attributes.

Sample Plot Program

```
procedure SAMPLE_PLOT
begin
  WINDOW(xw_min, xw_max, yw_min, yw_max);
  VIEWPORT (x_min, x_max, y_min, y_max);
  CREATE_SEGMENT(1);                    {segment id is 1}
    PLOT_GRAPH (x_array, y_array, n, '+ ');
    MOVE_ABS_2 (5.5, 8.5);
    TEXT ('cern_demo ');
  CLOSE_SEGMENT
end;
```

```
procedure PLOT_GRAPH(x,y: array[1..100] of real,
                              n: integer
                              marker: char);
{define a graph with n ≤ 100 data points, with marker}
begin
  {first define axes}
  AXES(0.0, 0.0, 15.0, 6.0);
  {next the x,y data; set line style}
  SET_LINESTYLE('dashed ');
  MOVE_ABS_2(x[1], y[1]);
  TEXT(marker);
  for i:=2 to n do
    begin
      LINE_ABS_2(x[i],y[1]);
      TEXT(marker)
    end
end {PLOT_GRAPH}
```

3.2. Input/Interaction

There are a number of ways of grouping physical input devices into logical/virtual ones, as exhibited by the various graphics packages. In the Core, there are two sampled logical input devices, the ***valuator*** (typically a control dial or other potentiometer) which returns a scalar floating point value between 0 and 1, and the ***locator*** (typically a data tablet/digitizer, joystick or mouse) which returns a two component vector of floating point values indicating where the cursor is in the normalized device coordinate space on the screen. The three event-causing devices are the ***keyboard*** or text device (the alphanumeric keyboard) which returns a character string and a string length, the ***button*** or choice device (for function keys) which returns an integer identifier of the button activated, and the ***pick*** device (a lightpen or a physical locator device with an integral interrupt button) which allows the system to be interrupted with information that can be used by the software to deduce which element on the screen was "pointed at" and return its *segment id* (and in some packages such as the Core and GKS, its *primitive id* as well). This id will then presumably be mapped back to the corresponding item in the application model which the user selected for further operation. The event-causing devices have a time-out parameter that is used to send an interrupt to the system when the specified amount of time has elapsed without user interaction - this allows the application program to remind the user to act and prevents the application program (and the host computer) from being occupied indefinitely when the user walks away without having logged off.

A useful way of thinking about user/computer dialogue is to use Finite State Automaton terminology. Thus the application program waits in a ***state*** until the user causes an event to occur with one of the three interrupting devices, and then reacts by changing the state of the display and/or of the data structure in response, thereby making a *transition* to a new state, and again waiting for the user. In this simple "ping-pong" dialogue model the program spends most of its time in an event-driven loop, waiting for the user to take action. The general outline of such a program is shown in high-level pseudo-code below:

```
initialize, including the start-up display
repeat
  provide a choice/pose a question
  wait for the user to respond with a proper input
  case to the corresponding procedure which deals with that input
until stop action is input
```

Next we show the "mainline" of a simple but typical interactive layout program, that can be used to ***add*** symbols (shown in a menu of such symbols/icons) to the drawing, to ***delete*** previously placed symbols from a drawing, to ***add a title*** to such a layout, and to ***change a view*** to pan and/or zoom over the drawing. The complete program is developed to show the common aspects of interactive graphics programs in [FOLE81].

```
program LAYOUT (input, output);
{declare variables}

begin
 initialize system, setup Window, Viewport; plot initial display, menu;
 SCREEN_FEEDBACK('please make a choice');
 main_done:=false;
 repeat
  {wait for user generated event}
  WAIT_BUTTON(wait_time, button_number);
  case button_number of
   0: SCREEN_FEEDBACK('waiting for you...')
   1: NEW_TITLE;
   2: ADD_SYMBOL;
   3: DELETE_SYMBOL;
   4: CHANGE_VIEW
   5: main_done:=true;
   else: SCREEN_FEEDBACK ('illegal selection')
  end {case}
 until main_done
 cleanup
end.
```

It is important to realize that it is typically neither easy nor successful to graft an interactive interface onto a program designed exclusively to run in batch. The interactive program should be designed around the interaction, not the computation, which is easily invoked at the appropriate time. The *ergonomics*/human factors of application programs have commonly been found to be more critical to the success of a program than even its functional capabilities. With the advent of raster graphics based professional workstations a body of knowledge about proper interactive program design is emerging and many people expect that the biggest advance in computing for the 1980s will come from having the concerns of implementers shift from conserving computing resources to making users more productive by giving them an optimal user interface [BROW83]. Optimality here refers to ease of learning, ease of use, coherence and uniformity of style, expressive power and making the system forgiving of user error (e.g., by allowing the user to UNDO in reverse order all of the commands executed during a given session).

3.3. Summary of Commands for a Simple Graphics Package

1. Graphic output primitives	*Function*
MOVE_ABS_2(x, y)	move the CP to (*x*, *y*)
MOVE_REL_2(dx, dy)	move the CP to (CP_x + *dx*, CP_y + *dy*)
POINT_ABS_2(x, y)	define a point at (*x*, *y*)
POINT_REL_2(dx, dy)	define a point at (CP_x + *dx*, CP_y + *dy*)
LINE_ABS_2(x, y)	define a line from CP to (*x*, *y*)
LINE_REL_2(dx, dy)	define a line from CP to (CP_x + *dx*, CP_y + *dy*)
POLYGON(x_array, y_array, n)	define a closed polygon with *n* vertices starting and ending at (*x_array*[1], *y_array*[1])
TEXT(string)	define a string specified by *string* at the CP

2. Attribute setting	*Function*
SET_LINESTYLE(style)	set the line style for all following primitives until reset
SET_COLOR(color)	set the color for all following primitives until reset
3. Segment control	
CREATE_SEGMENT(segment_name)	start a new segment with an integer name
CLOSE_SEGMENT	close the currently open segment
DELETE_SEGMENT(segment_name)	delete *segment_name*
RENAME_SEGMENT(old_name, new_name)	rename segment *old_name* to *new_name*
SET_VISIBILITY(segment_name, on/off)	make *segment_name* visible or invisible
TRANSLATE_IMAGE_2(segment_name, dx, dy)	translate *segment_name*'s image by *dx* and *dy* in NDC units
4. Viewing operation	
WINDOW(x_min, x_max, y_min, y_max)	specify window in world coordinates
VIEWPORT_2(x_min, x_max, y_min, y_max)	specify viewport in NDC units
5. Input	
READ_LOCATOR(x, y)	sample locator position on screen, return NDC value in *x, y*
READ_VALUATOR(n, value)	sample the *n*th valuator, return fraction between 0 and 1 in *value*
*WAIT_BUTTON(time, button_name)**	return 0 or name of button pressed in *button_name*
*WAIT_PICK(time, segment_name)**	return 0 or name of segment containing picked primitive in *segment_name*
*WAIT_KEYBOARD(time, text, length)**	read typed characters into *text*, return 0 or number of characters typed in *length*
INVERSE_2(x_ndc, y_ndc, x_wc, y_wc)	convert an *x, y* in normalized device coordinates (typically read from locator) to world coordinates using current window and viewport specification
6. Control	*Function*
INITIALIZE	clear screen; set default window, viewport; display cursor at locator *x, y*
TERMINATE	clear screen and close the graphics device
INQUIRE_COLOR	return current color attribute

*Each of the *WAIT* procedures waits for either time-out (in which case it returns 0) or the user-activated event (and returns the associated event data).

3.4. Current Standards

Among well-known older standards are Tektronix' Plot-10 which is relatively high-level but runs only on the Tektronix, GD-3, an old CERN standard, and GPGS [VAND77] and Gino-F [GINO76] which were the first widely-used and truly device-independent packages for interactive graphics. Among the newer standards are the aforementioned Core and GKS. More limited, primarily plotting-oriented packages include DISSPLA, TEL-A-GRAF and SAS-GRAPH.

Such plotting utilities are often much richer than interactive graphics packages in the variety of plots they allow the programmer to specify, having built-in graphs and charts, various kinds of "graph paper", automatic scaling and labeling, a full set of fonts in various sizes with which to do typography, and many other high-level features. It is not uncommon to build such utilities "on top of" a device-independent graphics package, thereby using the package's plotting and viewing/clipping facilities.

Other packages which are frequently used in conjunction with or on top of device-independent graphics packages include *modeling packages*. These allow the programmer to construct object hierarchies in which objects are composed of lower-level objects and/or primitives. Such sub-objects are *instanced* with translation, rotation and scale parameters used to define position, orientation and size, respectively, of the sub-object. An application program can then call on both the graphics package and the modeling package to do all the work not strictly related to analysis of the model, i.e., using them to let the user construct the model interactively on the screen prior to analysis.

4. Applications in CAE/CAD

4.1. Historical Perspective

The origins of this explosively growing applications area of computer graphics lie in the field of Computer-Aided Manufacturing (CAM) which started effectively with the widespread adoption of APT, a parts programming language for numerically-controlled machine tools developed at MIT in the mid 1950s. In the early 1960s GM and Ford in the automobile industry and Boeing and other large aerospace corporations used the very first interactive graphics systems for interactive blueprint production ("design drafting"). Much development was done in conjunction with or based on Ivan Sutherland's seminal work on Sketchpad [SUTH63] done in 1963 at MIT, showing the power of interactive graphics for creating and editing repetitive line drawings such as those found in blueprints. Since these drawings were specified by engineers to document their designs, the field of using graphics for design became known as Computer-Aided Design (CAD). CAM, then, refers to the manufacturing of components which have typically been designed on computer-based CAD systems. Particularly in the case of electronics, design can be split into roughly two phases; *logical* design which is largely implementation-independent, and *physical* design which must take into account physical component characteristics. Historically, CAD has put much more emphasis on the physical design phase than on the logical design phase, at least in part because it was easier to "computerize" and in part because it integrates with CAM. In the mid-70s, Computer-Aided Engineering (CAE) started developing, taking as its domain the "up-front" analysis and logical design. For example, in electronic CAE/CAD systems a designer can go from logic diagrams to placement and routing of ICs, including extensive simulation and verification at a variety of levels, all on a graphics workstation with extensive computing resources. A CAD/CAM system can then take the detailed board specification and produce a variety of production data for manufacturing, drilling, wiring, soldering, etc. of circuit boards.

The driving forces for the rapid introduction of automation in the engineering design and manufacturing process have been the sharply-increasing manpower costs and decreasing hardware costs, leading to a general recognition that automation must be used to increase the productivity of designers and implementers at all levels. Ideally one should be able to go from a conceptual design in the back of one's head to a complete part, without leaving the design station.

According to Machover [MACH80], in 1970 there were about 200 workstations for several-hundred users with a capital investment on the order of $1-million. By 1980, there were more than 12,000 workstations for more than 25,000 users with on the order of $1-billion of capital investment. A number of manufacturers such as Computervision, Applicon, IBM, Matra and Lockheed are selling hardware and/or software systems for both electronic and mechanical CAD/CAM, the two areas in which the state-of-the-art has progressed the furthest and in which the maximum payoff has been realized. So far, CAE has not yet begun to have much market penetration and there are remarkably few examples of CAD/CAM systems, let alone CAE/CAD/CAM systems that satisfy the obvious criteria of having good price/performance, high functionality and especially a totally integrated environment with good ergonomics, (rather than a hodge-podge of unintegrated subsystems, each with its own idiosyncratic behavior).

Because of the high degree of automation in Japanese industry, other western countries are finally taking a strong interest in automating both their factories via CAM and their designers and implementers via CAE/CAD tools. Only in the past several years have a significant number of universities been taking the field seriously, by introducing courses in the area, or even better, integrating CAE/CAD/CAM into many courses in their engineering curriculum. In the U.S., industry and government are beginning to address the teaching/training problem, typically by tax legislation which makes it very inexpensive for industry to donate modern equipment to universities.

4.2. What is the State-of-the-Art at CERN Today?

As an example of common practice at CERN in 1982, we take a superficial look at the process for designing and manufacturing integrated circuits.

1. The ***designer*** creates a logical design and schematics, using manufacturers' data, technical publications, etc., either on paper and/or as a formal specification, using a procedural language and debugging environment for simulating both the logical behavior and structure of the design [BARB81].
2. The ***layout technician***, by hand and using his experience and his practiced eye, transcribes the (hand-drawn) schematics by
   - ***assigning*** logical functions to packages
   - ***placing*** packages/components, identifying offboard signals to connectors, etc., and
   - ***routing*** as a function of board density, component technology, etc.

   The difficulty of the layout job is a function of the size of the board and the number of components, the density, the technology and the number of layers. The tools are colored pencils and erasers on large pieces of paper!
3. The ***technician*** digitizes the schematics on the Quest PCB design system and gets color check plots which must be carefully checked against the original sketches.
4. The ***Quest System*** checks geometric design rules and notifies the user of errors. Note that an even more common, completely manual technique of

pasting up tape strips on mylar is much worse than using Quest! As is, using Quest is a very time-intensive process and detecting and correcting errors, especially design errors, at this late stage is especially time-consuming and costly. Clearly, it would be far more desirable to have design errors at any stage propagate (semi-) automatically through the rest of the pipeline, which is possible only on a fully-automated, integrated system.

5. The *Quest System* produces "documentation" automatically which is error-free: lists of masks, solder masks, parts; NC tapes, photo plotter tapes for masks, etc.

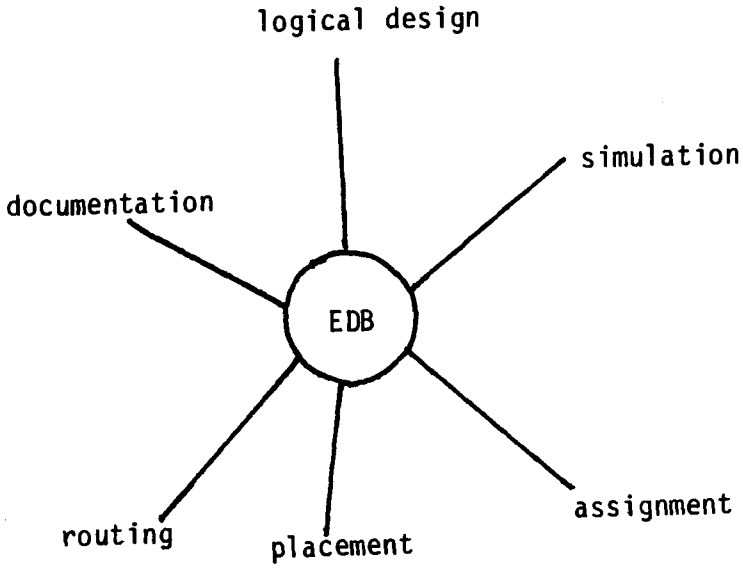
In summary, Quest is a very useful CAM system but automates only a small part of the total design/implementation pipeline and is therefore of increasingly limited use as newer, richer systems become cost-effective.

4.3. A Summary of Key Ideas

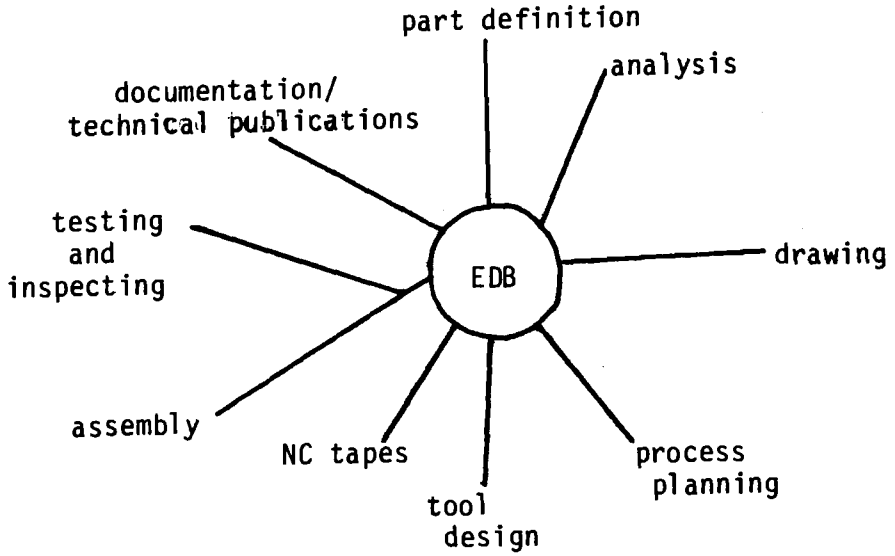
1. In a somewhat similar application field, office automation, the first thing to be automated was the means of production, with the computerized phototypesetter. Next came the automation of the clerical staff with word processing systems. Now we are seeing a move to automate professionals, i.e., authors and other knowledge workers. In the design field, the corresponding development was first to automate manufacturing/production with CAM. Next came automation for physical design done primarily by highly-skilled technicians and craftsmen, via CAD, and we're now seeing the introduction of CAE tools for the designer. As in office automation, it is this third type of professional whose success or failure has the most impact on a project and whose time is most expensive. Also, automating the front-end of a pipeline allows design errors to be caught and corrected at the least expensive point and allows cut-and-try design iterations without major investments in (wasted) implementation.

2. The most important point of automation is that the engineering database (EDB) is at the center of the entire system (See Fig. 7). To a first approximation, all subsystems read and/or write this central, integrated set of descriptions at various levels of what is being designed and implemented. Common terminology refers to design as the front-end of the pipeline, covering logical and physical design, and then refers to most of the CAM activity as "post-processing". In most systems today there are unfortunate and artificial boundaries between stages of the pipeline because there is no central engineering database. Typically for electronics, CAE front-ends for logical design are grafted onto pre-existing CAD/CAM back-ends for physical design and manufacturing.

   For example, consider this simplified summary of the IC design process: the logical design front-end covers architectural design, system design, logic design and design validation through simulation, testing, timing analysis, etc. Then there is a conversion of logical to physical design which often involves schematic entry/digitizing and assignment of logical functions to gates, plus board geometry determination. A complete *net list* detailing the schematic may be specified to a back-end CAD/CAM system which handles the physical design consisting of placement, routing and design rule checks, followed by documentation and manufacturing. The CAD/CAM diagrams may be "back-annotated" by the CAD/CAM system for use by the CAE system, with the transcription possibly being manual. This type of "impedance matching", with its attendant time-consuming, error-prone and costly data conversion, is now recognized as a major bottleneck



a.) electrical design



b.) mechanical design

Fig. 7: The Engineering Data Base at the center of all processing

to be eliminated, and the newest systems being designed have a genuine systems architecture designed a priori rather than being a jumble of pieces shoehorned into a "system".

3. Later this decade, we may expect that so-called "knowledge-based expert systems" will be used to assist designers by letting them ask questions and even giving them suggestions on how to proceed. Artificial Intelligence (AI) techniques for storing domain-dependent knowledge have become commercially viable for such diverse fields as medical diagnosis and oil well log interpretation. In essence, deep knowledge of design rules must be captured in programmatic form, including common design heuristics. When such techniques become part of the system, it will evolve from a passive "super drafting table cum designer's notebook cum analysis tool" to a genuine "designer's assistant" with which one can carry on a dialogue as with a skilled technician or colleague.
4. Training and education will become ever more important in the new age of computer-assisted tools and methodologies. It is now considered that the half-life of an engineer's knowledge is only a few years and that life-long education will be required in order to make him/her function cost-effectively. Fortunately, computer-based education can be delivered on the same workstation used for design, but it will take an enormous investment in authoring the appropriate electronic documents and lessons before this technique becomes truly practical.
5 One of the great advantages of automation is that it can eliminate much manual data entry and conversion, thereby reducing cost, error rate and tedium, while increasing job satisfaction, quality of the product, time to design and ability to modify, enhance and extend. These claims are well-documented both in design and office automation.
6. Modern systems tend to retain familiar, proven metaphors, and mimic and improve existing work habits. As with software design, modularity is encouraged, indeed enforced. The designer can simultaneously decompose the system top-down and make bottom-up use of standard building block components. Modern design therefore creates hierarchies of functions and objects.

4.4. Some Potential Advantages

- increase productivity, free time from dull, repetitive tasks for creativity
- decrease cost of design and manufacturing, material waste
- shorten turn-around time ("time is money")
- allow more experimentation, alternatives, special cases; create a family of designs and spin-offs
- improve quality, yield, accuracy, number of errors, (re)liability/safety
- allow tackling of complex jobs
- standardization

4.5. Some Application Areas

- IC, PCB, PW design/layout/manufacturing
- wiring and electrical diagrams
- piping
- plant and other architectural design

- Numerical Control of parts manufacturing
- 2D, 3D mechanical design and drafting auto, ship, plane... hulls, structure, engines,...
- tool design
- mapping and cartography

4.6. A Commercial Example of Computer Aided Engineering [DATA82]

The SCALDsystem (Structured Computer Aided Logic Design) consists of a set of hardware, procedures, and programs that can reduce design time for a digital electronic system by a factor of 10, the vendor says. The system is configured around interactive, real-time graphic design stations for schematic editing; the stations can share application programs with a mainframe and exchange data at 800K bps.

The engineer using the system can begin a design with a block diagram comprised of a few functional items. These can then be subdivided, with added detail, until each block is described on the screen in terms of actual components. After the drawings have been created, using a graphics editor in the Unix environment, the SCALD compiler expands the design and checks interfaces to ensure consistency.

A timing verifier assists the engineer throughout the design process in detecting logic-level timing errors such as races, clock glitches, setup and hold violations and pulse width errors. A logic simulator provides interactive simulation at speeds faster than conventional gate-array simulators, the vendor says. The simulator makes it possible to debug microcode firmware and software as well as hardware. A graphics design database is included in the package, and by operating the logic simulator on the database the need to define and maintain a separate high-level model for a design is eliminated. An interface from the design database to physical design systems, as well as to other analysis and testing tools, is provided by a post processor.

Hardware elements of the system include a desktop graphics design workstation based on the Intel 8086 microprocessor and a cluster controller designed around the MC68000 microprocessor. The controller can support up to four design stations in a cluster configuration and link clusters to a network or host mainframe.

Each design station consists of a 20-inch crt with 1,024x768 resolution, full ASCII keyboard with programmable function keys, a graphics tablet, and two microprocessors. The raster scan display has a refresh rate of 60 Hz with four intensity levels. The SCALDsystem design stations are $35,000 apiece or $138,000 for a four-station system from VALID LOGIC SYSTEMS, Sunnyvale, Calif.

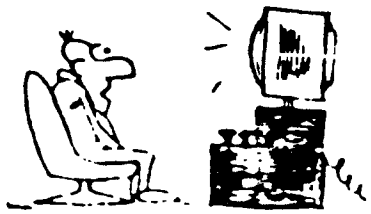
5. Conclusion

During the next few years a revolution in computing environments is going to take place that is at least as profound as the change from a batch processing environment to a time-sharing environment with alphanumeric displays running at 9600 baud. We are entering the age of adequate computing power through workstations connected over high-bandwidth local area networks connected via broadband backbone networks. Each workstation will have the computational power of roughly one-quarter to one-half of today's VAX/780. Most exciting, a high-resolution bit map raster graphics display will be an integral part of each workstation, allowing a high-bandwidth, highly-interactive user/computer dialogue with dedicated computer resources. In 1983, low-end workstations will cost less than $10,000 each -- indeed Apple's MacIntosh personal workstation will

reportedly cost less than $3000 (for quantity 1, less in bulk), and with appropriate file, printer and mainframe computation *servers* connected on the resource-sharing network can be used for many office automation and design automation functions. Such low-priced stations will completely replace alphanumeric displays. Naturally, many jobs will require larger real and virtual memory, larger disk storage, "landscape" screens of 17" or even 19" diameter, etc., and may therefore cost many tens of thousands of dollars, especially with hardware floating-point and high-resolution color. Families of workstations will emerge, with common, highly-ergonomic user interfaces, allowing the user to switch without interruption, let alone reprogramming to more powerful members of the family when his resource requirements exceed those of the workstation on his desk. Mentor Graphics Idea 1000 electronic-design system running on an Apollo Domain network [SWER82] is a modern CAE system, as will be Hewlett Packard's CAE/CAD/CAM offerings on their new HP 9000 workstations to be delivered in 1983. Well before the end of the decade, the graphics-based user interface emphasizing pointing and menu-picking rather than typing will have become the standard. The field of CAE/CAD/CAM in particular will greatly benefit from the improvements in computing power and the user interface, to counteract today's problemsas summarized in Fig. 8.

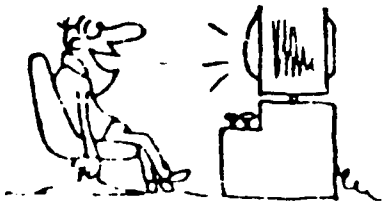
DIE EINFÜHRUNGSPHASEN EINES CAD - SYSTEMS

1. Skepsis

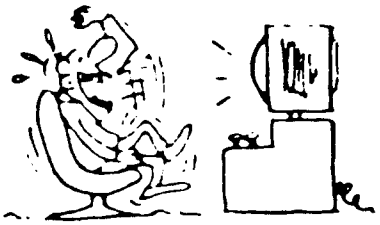


2. Spannung

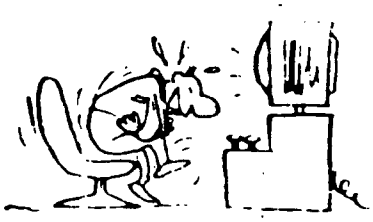
3. Erstaunen



4. Begeisterung

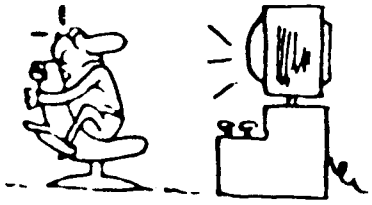


5. Enthusiasmus

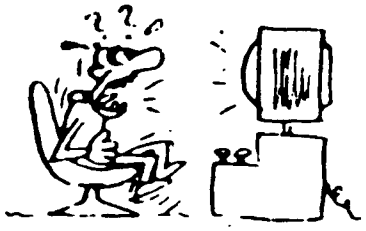


6. Ernüchterung

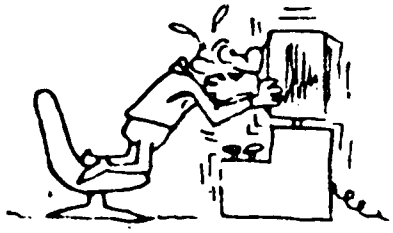
7. Erschrecken



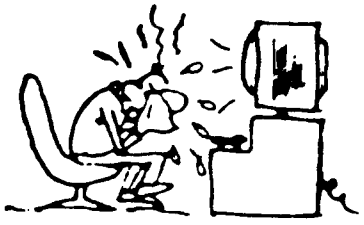
8. Sprachlosigkeit



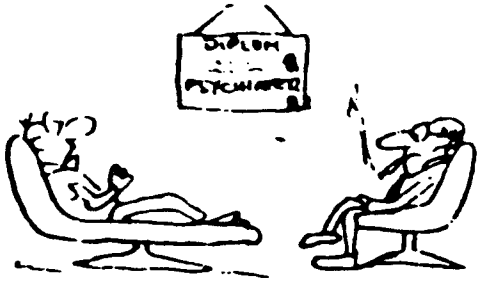
9. Entsetzen



10. Frustration



11. Entspannung



6. References

[BARB81]
Barbacci, M., C. Halatsis, J. Joosten, M. Letheren, A. van Dam, "Simulation of a Horizontal Bit-Sliced Processor Using the ISPS Architecture Simulation Facility", *IEEE Transactions on Computers* (special issue on firmware engineering), February 1981.

[BROW83]
Brown, M.H., N. Meyrowitz and A. van Dam, "Personal Computer Networks and Graphical Animation: Rationale and Practice for Education", *Proceedings SIGSE*, February 1983.

[DATA82]
"Computer Aided Engineering", *Datamation*, December 1982, pp. 168.

[FOLE81]
Foley, J. D., A. van Dam, *Fundamentals of Interactive Computer Graphics*, The Systems Programming Series, Addison Wesley Publishing Company, December 1981.

[GINO76]
"Gino-F User Manual," Issue 2, Computer-Aided Design Centre, Cambridge, England, December 1976.

[GSPC79]
"Status Report of the Graphics Standards Committee," *Computer Graphics*, 13(3), August 1979.

[ISO81]
International Standards Organization, "Graphical Kernel System (GKS), Version 6.6," May 1981.

[MACH80]
Machover, C., R.E. Blauth, *The CAD/CAM Handbook*, Computervision Corporation, Bedford, MA, November 1980.

[MEYR82]
Meyrowitz, N., A. van Dam, "Interactive Editing Systems: Part I and Part II", *ACM Computing Surveys*, 14(3), September 1982, pp. 321-415.

[SMIT82]
Smith, D.C., C. Irby, R. Kimball, B. Verplank and E. Harslem "Designing the Star user interface," *BYTE* **7**, 4 (Apr. 1982), 242-282.

[SUTH63]
Sutherland, I.E., *SKETCHPAD: A Man-Machine Graphical Communication System*, SJCC 1963, Spartan Books, Baltimore, MD, pp. 329.

[SWER82]
Swerling, S. G. H. Langeler, "CAE tool supports both design and routine tasks", *ElectronicDesign*, Hayden Publishing Co., Inc., May 1982.

[VAND77]
van den Bos, J., L. C. Caruthers and A. van Dam, "GPGS: A Device-independent General Purpose Graphic System," SIGGRAPH '77 Proceedings, published as *Computer Graphics*, 11(2), Summer 1977, pp. 112-119.