Graphical Information Resources: Maps and Beyond

Michael Lesk

Bell Communications Research
Morristown, NJ 07960

ABSTRACT

The rise of computer graphics offers a new challenge for information retrieval: how to search and retrieve information which is partly or wholly graphical. As an example, procedures for handling geographical information, such as street maps and directories are explained. With this data, it is possible to find routes on maps, retrieve locations and names of people or businesses, and draw maps. But a comparison of these programs with programs for face processing or computer typesetting makes clear how far we are from general purpose routines. Today successful graphics routines contain a great deal of local domain knowledge. There is no analog of the simple keyword systems that handle textual documents in any subject area. Just as computational linguists have found that subject matter expertise is necessary to do really sophisticated processing of English, it seems also necessary to sophisticated processing of pictures; the difference is that we don't know how to do unsophisticated processing of graphics.

1. Introduction.

Recent developments in technology are making pictorial and graphical information much more important. Bitmap display technology is now available in many terminals and workstations, e.g. the TTY 5620 and the Sun and Apollo workstations. The conventional description of the workstation/terminal of the future is now the "5M" station: one Mips processor speed, one Mbyte main memory, one Mpixel display, one Mbit network connection, and a mouse. The Macintosh even provides a bitmap display at a price competitive with ordinary terminals. Simultaneously, laser printers are getting much cheaper, with products such as the Imagen 8/300, the HP Laserjet, and the Applewriter all available at low prices and all capable of printing good quality graphic images. Storage of pictures is possible today in analog form on optical laser discs, and digital storage of picture is very likely for the future. Wideband communications systems will make it possible to transmit pictures easily. Finally, special purpose graphical processors from companies like Silicon Graphics and HP are performing previously slow geometric operations in real-time.

Just as the development of computerized typesetting for text produced large volumes of bibliographic information in machine readable form, and a demand for on-line text retrieval systems, we can expect the development of large pictorial databases. Videodiscs with tens of thousands of still frames already exist; more elaborate, digitized reference libraries are likely to arrive soon. How will we file and retrieve such information?

In the past, graphical data has been viewed as a black box; an image is stored, along with a few words describing it, and all indexing, retrieval, and arrangement is based on the text label, rather than the picture. What should be done in the future? Somehow, we need to process the content of the graphics image, and do retrieval based on its properties, rather than just its labels. This talk discusses a particular example of graphical information, and how it was organized and stored; it will then proceed to deplore the lack of general graphics tools, and hope for better tools in the future.

2. Map Processing.

One special purpose subject area in which pictorial information has been digitized is in the representation of maps. Maps, like bibliographies, are permanent reference works that get updated, and so their storage and production by computer would be very cost-effective. Unfortunately, maps also require a very high resolution printing system, typically in multiple colors, and thus are difficult to produce by computer means. Nevertheless, substantial map information is now available in machine readable form. Our particular work has used the GBF/DIME files of the U. S. Census Bureau, which represent the street maps of major metropolitan areas. Other digitized maps, however, include the U. S. G. S. 1:2,000,000 series, which are completely digitized, including both topography and surface features; the U. S. G. S. 1:250,000 series, for which the topography is digitized; and occasional maps at larger scales. Some vendors now have a business editing and improving on these government sources; and some industries (including for example the petroleum and communications businesses) create and store their own maps. There is an effort in progress to standardize machine-readable street maps for vehicle routing applications.

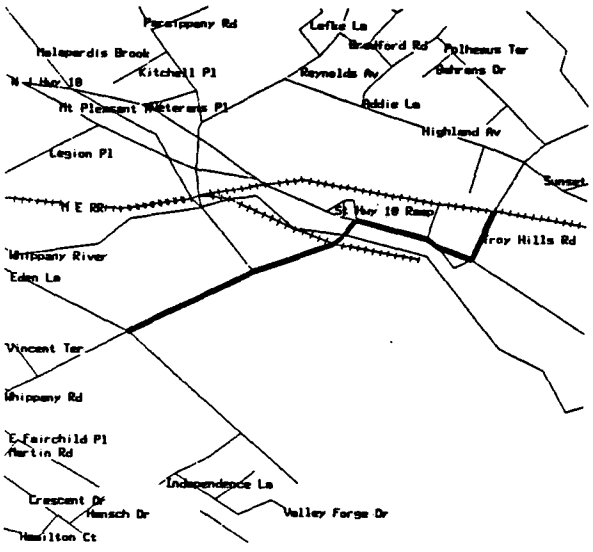
Tied to the standard map information are a variety of other databases which are tied to geographical position and may also have graphical content. These include, for example, weather information; information on towns; information on traffic; and information on businesses and their locations. It sometimes takes additional work to connect these databases to maps, however.[1] The use of place names, for example, typically requires a translation to latitude and longitude. Weather information, for example, is often identified as to geographical location by the name of the observing station, typically an airport. The translation from airports to latitude and longitude is easy (the FAA provides the necessary data) but must be done. News stories are datelined with the name of a city; this time it is the Census Bureau which provides a translation tape. Addresses, on a national scale, are tied to zip codes; the Postal Service offers yet another database for translation. Local addresses require a street map to identify exact location.

Working with R. J. Elliott of Bell Laboratories, a set of programs was developed to print, store, and search maps.[2] The underlying purpose of the system was a way to retrieve locations and find routes to them. Here are some samples of directions given by the system for driving from one place to another. In these examples, what the user types is in *italics* and the computer responds in the normal font. The programs recognize locations given as an intersection, street address, or for those areas with business lists, the name of a business or a category of business. The first example specifies locations by an intersection and then by the "Nearest X" construction.

Where are you? *whippany rd and eden la*
Where do you want to go? *nearest restaurant*

closest place is VICTORIA STATION
From EDEN LA and WHIPPANY RD
travel NE on WHIPPANY RD
go 0.5 miles (2 intersections) and cross M&E RR
go 144 ft and cross WHIPPANY RIVER
go 252 ft turn right on ST HWY 10
go 0.3 miles (2 intersections) turn left on TROY HILLS RD
go 0.1 miles to intersection of M&E RR and TROY HILLS RD

Figure 1.



The route found is shown in Figure 1. Note that all routing, at present, is from intersection to intersection; thus addresses are "rounded off" automatically to the nearest corner. The next example is a straightforward intersection to intersection route:
Where are you? *south st and mountain av, 07974*
Where do you want to go? *295 n maple av, 07920*

Start at Mountain Av and South St (2074552 0675596), end at 295 N Maple Av, 07920 (2035459 0689905)

412 nodes touched in finding path.

From GLENSIDE RD, MOUNTAIN AV and SOUTH ST
travel SW on MOUNTAIN AV
go 4.1 miles (19 intersections)
and cross HILLCREST RD and PASSAIC RIVER
go 0.6 miles (7 intersections) and cross EL RR
go 0.9 miles (2 intersections) turn left on LONG HILL RD
go 2.1 miles (5 intersections)
turn right on BASKING RIDGE RD
go 0.6 miles (4 intersections) continue on S MAPLE AV
go 2.1 miles (12 intersections) turn right on N MAPLE AV
go 1.6 miles (6 intersections) to intersection
of I 287 RAMP, N MAPLE AV and
N MAPLE AV RAMP
total distance 11.9 miles

The next example uses the name of a business as a location:

Where are you? *at the Lackawanna Diner*
Where do you want to go? *14 Oak st, 07960*
39 nodes touched in finding path.

From MORRIS AVENUE WEST and MORRIS ST
travel E on MORRIS ST
go 227 ft turn right on PINE ST
go 0.2 miles (3 intersections) turn right on SOUTH ST
by a business district which includes:
Beauty Salons: David Salon De,
Beauty Salons: Michael's Salon,
Commodity Brokers: Shearson Loeb Rhoades Inc,
Restaurants: Community Coffee Shop and
Thrift Shops: South Street Exchange
go 0.1 miles (2 intersections) turn left on DE HART ST
by Commodity Brokers: Shearson Loeb Rhoades Inc and
Loans: Heritage Bank-North
go 0.2 miles (2 intersections) turn left on
MAC CULLOCH AV
by Lawyers: Schlossberg Michael
go 0.1 miles to intersection of FARRAGUT PL,
MAC CULLOCH AV and OAK ST

This is actually a very short trip. It seems long because it involves several close-together turns in downtown Morristown. It is shown in Figure 2. This may not appear to be the shortest route, but in a trip this short the program is primarily trying to minimize turns. We impose a cost of 0.25 miles on each left turn and 0.125 miles on each right turn, since otherwise the program tends to make a very large number of turns in its effort to save distance. In this case, the attempt to avoid the left turn from Community Place onto Maple Avenue caused the route to go one block further west than might seem obvious. There are options that will give the shortest route regardless of turns, but on long trips this produces very complex instructions.

The length of the description is sometimes considered a disadvantage: the wealth of detail given by the program

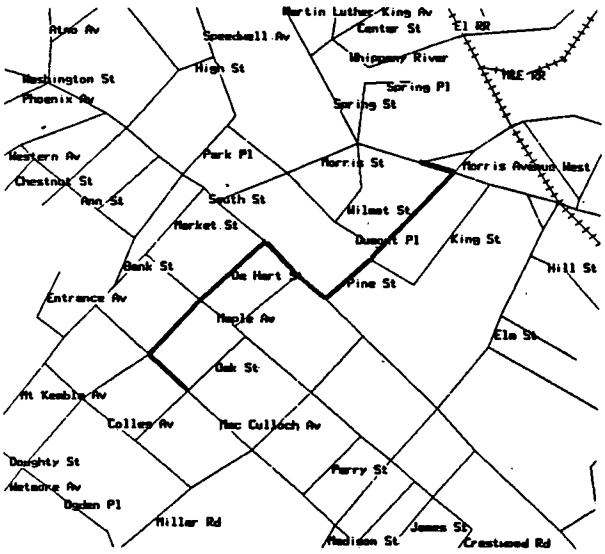


Figure 2.

makes trips seem longer than they are. In addition to specifying all turns by direction and street name, the program details the distance on each street, the number of intersections crossed, and the business establishments at each turn. It also reports each river or railroad crossing. Although much of this might be superfluous in the case of a driver familiar with the area, it is hoped that it would be valuable to a novice driver. Some improvements (e.g. recognizing that some businesses, like lawyer's offices, are not prominent along the road and not useful as landmarks) are obvious.

The definition of the best route is not obvious. Originally, we assumed the shortest distance route was best; but such routes in our suburban area often involve a staggering number of turns and are very complex. Alternatives include minimum time, if the average travel speed on different streets is known, and minimum length of directions required (roughly, minimum number of turns). Minimizing the number of turns, however, sometimes produces very long routes. Still other metrics (e.g. minimal gas consumption, or minimum number of traffic lights) have been suggested. As a compromise we use minimum distance with a penalty for turns, which produces reasonable-looking routes. The program has options to find minimum time routes if the average speed on different streets is known; for example, Figures 3 and 4 show routes from lower to mid-Manhattan; in Figure 3 the distance is minimized (subject to our penalties for turns) while in Figure 4 the program is trying to minimize time, given a table showing 6th Avenue as very slow and 11th Avenue as somewhat faster (the table does not include some other roads that might be faster, such as the FDR Drive).

In our processing of geographical data, several different operations must be performed, and this affects the choice of data representation. To search for a particular street, there had to be an alphabetical list of streets by name. To draw a map, it is necessary to have fast access to all points within a certain area. To find routes, a list of all points connected to a given point is necessary. Finally, it is important to be able to find the closest intersection to a given x-y position.

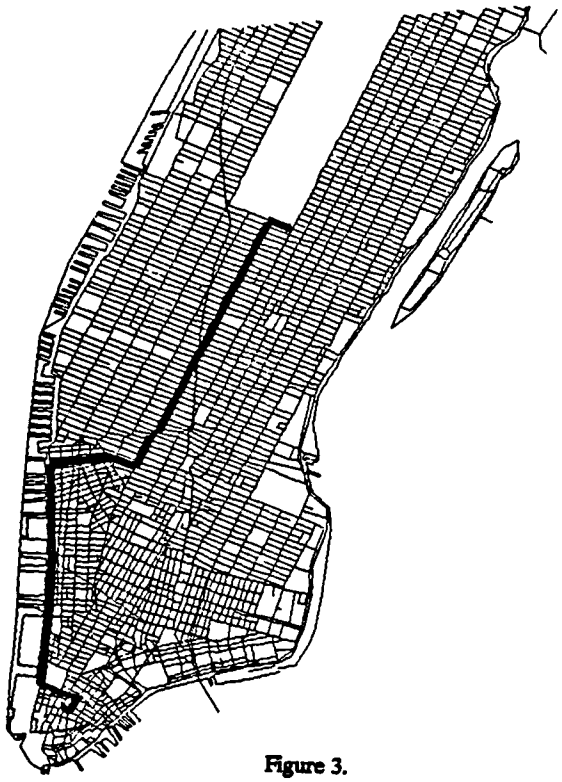


Figure 3.

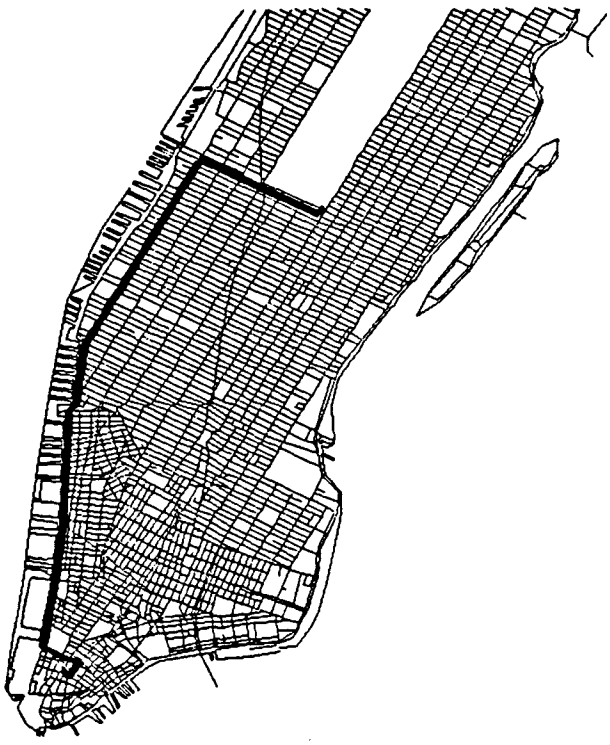


Figure 4.

It would be possible to design a different data representation for each of these functions, but that is expensive. Instead, a general storage method involving square patches over the map was used.[3] The map is divided into squares, each 10,000 feet on a side. This size was chosen because it roughly corresponds to the size of the largest area map that can be drawn in full on a typical screen display. Thus relatively few patch fetches, and therefore disk seeks, are required in order to draw a map. If the map is much smaller, than it is very short, and the although an entire patch will be read and only part of the patch used, the map is small enough that it can still be drawn in a few seconds. A map of an area much larger than a single patch can not be displayed in full on a typical 1000x1000 bitmap screen, and so it will be excerpted. We also store an condensed version of the data, showing only the important streets, and so larger maps can be drawn quickly from this abbreviated data file. Sample maps are shown in Figures 5 and 6.

Underneath the problem of giving directions, of course, is the shortest path problem. This is well studied mathematically, but the mathematical work[4, 5, 6] generally deals with arbitrary graphs. The street maps are planar, have a Euclidean metric, and are generally well-behaved. All of the streets are usually in only one partition of the graph (there is some route from every street intersection to every other). However, the maps are very large (perhaps 100,000 nodes in New York City) and have few edges per node, being sparsely connected.

Various routing algorithms are possible. There is a choice between breadth-first and depth-first search, between hierarchical and complete map routing, and between single and double-ended searching. In addition, it is necessary to decide on the right thing to minimize: distance, time, or turns.

As these choices were implemented:

* breadth-first search is the idea that all paths out of one node are explored before any of the paths from the resulting nodes are tried. Thus the set of nodes explored spreads out from the source until it reaches the destination, touching all nodes in the middle on the way.
* depth-first search attempts to follow one path through many nodes, only trying another path after the first is no longer profitable. Profitable is defined as no longer getting closer to the destination.
* complete routing involves looking at every single possible street between the source and the destination;
* hierarchical routing selects a group of important streets, and does not even consider unimportant streets except very close to the beginning and end of the trip (e.g. in planning a trip from New York to San Francisco you don't usually consider whether or not it makes sense to get off I-80 for a few miles in Iowa).
* single-ended searching works from one end to the other;
* double-ended searching works from both ends towards the middle.

In the mathematical literature, breadth first search is typically preferred. But the optimal algorithm, in the mathematical sense, is defined on nonplanar graphs and represents a worst case time. Normally the user cares

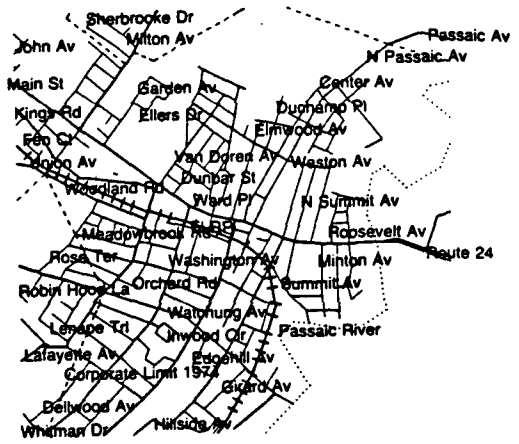


Figure 5.

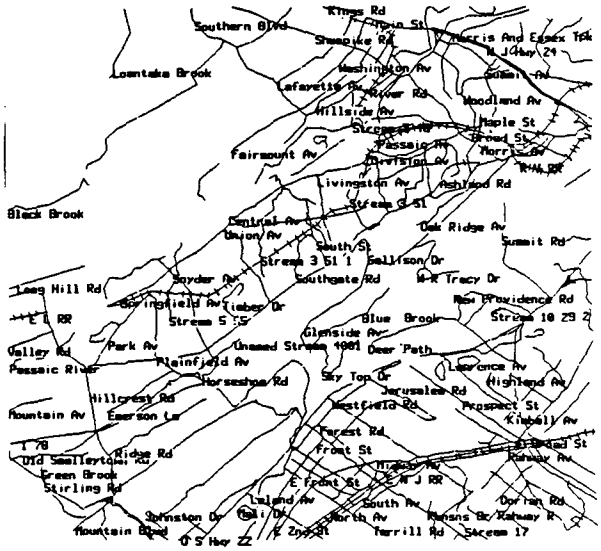


Figure 6.

more about average time than worst case, and road maps are well behaved graphs. Thus we found running experiments that depth-first hierarchical search was generally much faster. We do agree with the literature on the advantages of double-ended search, which represents a 25% savings; but this is sometimes wrong in specific cases.

How is the data processing for the route finding actually done? The route crosses some number of map squares. The streets in those squares are read in, since they fit in memory, and then internal memory operations are used to find the points connected to any given point. An alternative would be a large B-tree or other disk representation, but this is less efficient because of the locality of the data (if one node is touched by the route finding, the nodes near it are likely to be accessed also). Another possibility would be storing the connection lists for the graph; this would simplify routefinding but not map drawing. Our representation is chosen to simplify both finding the route and drawing a map of it; it does not, for example, consider

the problem of updating the map structure and would be inappropriate for an application in which the map changed frequently.

An important aspect of the map routing programs are the relations to other databases. For example, given a listing of businesses with categories and street addresses, it is possible to compute position from street address, and thus plot map location for each business. This permits specifying a location as "Nautilus Diner" or other such business name, and it is treated as if it were a street address. It is also possible to find the closest business of a particular category, e.g. "nearest restaurant". This searching is simple, because there are not a great many businesses in any category. Thus a list of businesses is maintained sorted on category name, and all the businesses within the specified category are scanned to find the closest one. Thus a route destination can be specified as "nearest barber" or "nearest bank".

It is also possible to plot maps showing the locations of businesses, even indicating which side of the street they are on. A sample map of this sort is shown in Figure 7. Other information peculiar to street maps that is stored and used in these programs includes which streets are one-way, and which streets are limited-access highways. There is also a town name to zip code translation table to give users flexibility in specifying addresses. Thus considerable special-purpose knowledge is included in the map system.

Because of the special knowledge and the special properties of the problem, however, the techniques used for map routing and drawing are not generally applicable beyond this sphere. For example, routing problems also arise in VLSI design. Many of the algorithms here have analogies in VLSI routing programs, but the interface and code can not be directly transferred. The constraints on VLSI routing, for example, are quite different, depending on the number of layers, etc. VLSI designs also tend to be restricted to routes with only right-angle turns, and the constraints on good VLSI routes are quite different from those on good driving routes. VLSI designers, for example, may be trading total chip area for speed in laying out routes, or trying to minimize the probability of defects.

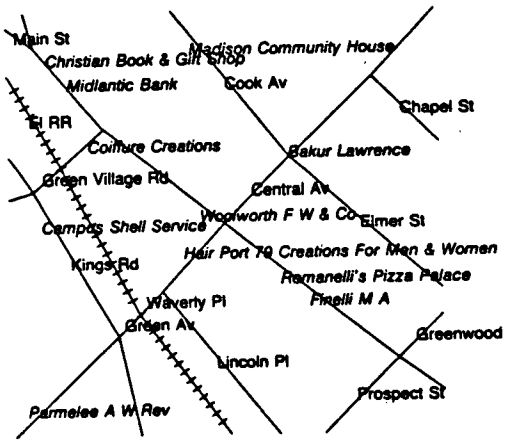


Figure 7.

Even within the domain of street maps and driving directions, the code may not be directly reusable. Suppose one wished to solve the traveling salesman problem instead of the shortest path problem; an new set of heuristics is needed. In theoretical terms, the problems are even more different since shortest path is a polynomial time problem while traveling salesman is NP-complete. Thus for either a different problem in the same domain, or a different domain for the same problem, new methods are appropriate.

3. Another Example: Typesetting

Another special purpose area of graphics programs is in computer typesetting. The entire job of typesetting can be considered a graphics problem, and yet consider the different solutions, even within the single world of the Unix system and its troff program.[7] Mathematical equations are specified by linearizing them, following the rule that you type what you would say to read the equation aloud. This turns, for example "sum from 0 to inf x sub i" into

$$\sum_{0}^{\infty} x_i$$

with the items *sum*, *from*, *to* and *inf* being reserved words. Their meaning is obviously peculiar to the equation world, and the general concept underlying the *eqn* typesetting program, namely that the world is made up of boxes to be adjoined at different positions, is never explicitly presented to the user.

Tables, by contrast, are specified in a two-dimensional way, in which an outline of the table, showing the relative adjustment of the columns, is presented first, followed by the actual numbers to be included. Thus

```
.TS
c s
l n.
Dow Jones Averages
Industrials 1253.86
Transportation 588.68
Utilities 154.34
.TE
```

produces

Dow Jones Averages	
Industrials	1253.86
Transportation	588.68
Utilities	154.34

This representation uses a two-dimensional rather than a one-dimensional representation (it is legal for the user to input the specification as *c s, l n.* but nobody ever does); it is declarative in nature, rather than imperative; and it is not similar to the procedure for representing matrices in *eqn!* The numbers in the table above could also be printed in *eqn* with

```
.EQ
lpile { Industrials above Transportation above Utilities }
rpile { "1253.86" above "588.68" above "154.34" }
.EN
```

yielding

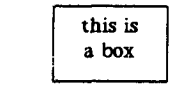
Industrials *1253.86*
Transportation *588.68*
Utilities *154.34*

Note that *eqn* has only one sequence with the commands folded into the data while *tbl* input is divided into a format specification and the data, and that the *tbl* language is declarative while the *eqn* language is imperative in style.

There are also, of course, actual graphics procedures in the *troff* package; in fact there are two of them, again one imperative and one declarative.[8, 9] The *pic* language again uses imperative commands specifying the items to be drawn in sequence; the *ideal* language provides for a representation in complex numbers. Sample utterances in these languages might be:

```
(pic)
box "this is" "a box"
```

which becomes



and

```
(ideal)
label {
  var location;
  text "This is" at location + (0,.15);
  text "a box" at location - (0,.15);
}
main {
  put rectangle {
  botleft = 0;
  width = 1.0;
  topleft - botleft = (0,1)*(botright-botleft);
  }
  put label {
    location = (0.5, 0.5);
  }
}
```

(The definition of *rectangle* has been omitted.)

is an *ideal* statement which draws a similar but square box. Note that *ideal* allows (encourages?) mathematical consistency statements among the constituents of its graphic displays. As with *eqn* vs. *tbl*, the comparison of *ideal* and *pic* discloses radically different ways of accomplishing apparently similar graphics operations. One is declarative, the other imperative; one oriented towards mathematics, the other towards drawing. However, both contain large amounts of typesetting expertise: arrows, drawing, etc. Neither would be appropriate, for example, for three-dimensional scene representation or producing pictures of faces.

To end the discussion of *troff* routines, consider the procedures for writing page layout routines. These are done with interrupt driven coding that has no obvious relation to anything graphical even though its purpose is to produce a two-dimensional arrangement of text blocks. There are many other typesetting and page description languages, but most of them are even more typesetting-specific than these. In some, however, the model of adjoining boxes is more explicit and presented to the user; this is, for example, true of Knuth's TEX language.[10] It has not, however, resulted in the use of TEX for other applications.

Again, there are a variety of styles of data representation used in printing documents. Much special purpose knowledge is used, and yet it is not as much as traditional printers brought to the same job (computer typesetting, for example, rarely interacts with the choice of words for the text, while in high quality printing sometimes the author would be instructed to gain or lose a line by rewriting). Once again, different subject domains demand different graphics data structures, different input languages, and different processing algorithms.

4. Other Domains.

Other graphical domains present similar problems. Success in processing faces has depended on the identification of key points on the face, defining the nose, eyes, mouth, and so forth, and the measurement or positioning of those points. The experiments by Leon Harmon and co-workers in face identification, for example, used a table of sizes of different facial features to drive an interactive program that retrieved faces from a file of 256 pictures.[11] Spotting points on faces is a special-purpose recognition task which is not the same as recognizing problems with printed circuit boards or distinguishing printed characters.[12] More recently Susan Brennan has programs which draw faces, including caricatures of faces, using about 150 located points on the faces.[13]

Another recent, very impressive set of graphics routines are the procedures for imitating natural shapes using fractals that have been built at IBM and Lucasfilm. The picture of "Road to Point Reyes" at the title page of the SIGGRAPH '83 conference proceedings is a remarkable example of computer scene production. However, the complexity of the work required to produce this picture almost destroys any hope of retrieving it in response to a request for pictures of roadside posts or rainbows.[14]

Another large, major class of graphics projects use raster scanned bitmap displays. In these, a picture does not even have a display list of vectors and it is even harder to decide from an examination of the picture what it might be. The advantage, of course, is that the software to handle bitmaps is somewhat more general; *bitblt*, the operation of copying a rectangle from one place to another, underlies many different applications. Ingenious techniques have been programmed to pan and zoom on bitmaps, as well as rotate and reflect, but this is still a very limited set of operations. The availability of display systems that quickly move pictures and display them, however, means that bitmaps can be used for picture browsing in a way that slower displays rarely can be. Unfortunately the intellectual processing of the pictures represented by bitmaps is almost impossible.

Other impressive graphics programs include the displays of protein molecule structures, the file of chemical structure diagrams at Chemical Abstracts, the processing of airplane wing designs at Boeing, the many VLSI design tools, and the many circuit design systems. In general, all of these systems include detailed, expert level information about their subject domains. Chemical programs, for example, enforce rules about linkages between atoms and legal molecular conformations. In VLSI design other rules enforce the number of layers, the connections between them, and the allowable distances between and sizes of objects. There is nothing common between these programs, and none of them can be applied outside their own domain.

The contrast with string processing programs is quite remarkable. There exist many on-line interactive retrieval programs that will process text in any subject area and retrieve it by keyword searching. Some of these programs (those without morphological analysis) may not even care what language the text is written in, much less its subject

domain. Storage procedures such as B-trees or extensible hashing are known to be efficient ways of storing indexes regardless of the material stored. Boolean queries or coordinate index queries also apply over any subject domain. By contrast, in graphics all we have are the basic operations that correspond to string compare, string move, and so forth. Effectively we have the graphics programs that correspond to font design and printer drivers; we don't really even have word processing equivalents, much less indexing and retrieval.

The absence of graphical tools of comparable scope to standard string matching procedures make it difficult to combine graphical data from different applications. For example, computer typesetting software rarely can be applied directly to chemical structure matching. As a result, nearly all graphical applications require special-purpose routines that are individual, and it is difficult to combine graphical procedures in the way that ordinary procedures are defined. There is almost a complete separation, for example, between procedures that deal with displaylists and those that deal with bitmaps.

There have been some indications of progress in producing general graphics routines. Fractals, for example, seem to be effective in representing a wide variety of natural scenes, whether mountains or plants.[15] Earlier there was work on grammars for pictures, and recently additional work on parallel grammars has added to this. Rosenfeld has published a recent review.[16] David McKeown has demonstrated an ability to translate between map representations and pictorial images.[17] His work explores matching up pictures from aerial photography and representations of the same area as a map. But this still does not make a retrieval system in any way as general as the commercial text retrieval programs now in common use.

How should graphics be stored and retrieved? Aside from the present technique of writing English descriptions of the pictures, and doing text retrieval on the descriptions, it would be better to keep track of the highest level description of the graphics, and writing programs to search for the elements of these descriptions. Given the speed with which scaling and translation can now be performed by hardware, it might be possible to take quick sketches as input to graphics systems, extract the features in the sketch, and look for those in the pictures. This would still be a low level representation, but it would be comparable to looking for identifiers in programs as opposed to high-level structures, which is the normal procedure today.

5. Conclusions.

Graphics programs are typically limited to a single application area. A comparable problem exists trying to put linguistic knowledge beyond simple word matching into retrieval. The design of a thesaurus, or the use of natural language understanding software, has so far meant restriction to a tiny subject domain. Extending language understanding programs to a large subject domain appears to involve problems of knowledge representation we do not know how to solve. Can it be that to have truly general graphics programs, we will again need general world understanding? I hope not; but I've not yet seen and don't know how to build a truly general high level graphics system.

References

1. M. E. Lesk, "Combining Data Bases: National and Cartographic Files," *Proc. AFIPS Office Automation Conference*, pp. 415-426, San Francisco, 1982.
2. R. J. Elliott and M. E. Lesk, "Route Finding in Street Maps by Computers and People," *Proc. Nat'l. Conf. on Artificial Intelligence*, pp. 258-261, Pittsburgh, Penn., 1982.
3. R. J. Elliott and M. E. Lesk, "Data Structures for Street Maps," *Proc. Nat'l Comm. Forum*, vol. 38, pp. 625-629, 1984.
4. E. F. Moore, "Shortest Path Through a Maze," *Proc. Int'l. Symp. on Theory of Switching*, vol. 2, pp. 285-292, 1957.
5. D. B. Johnson, "Efficient Algorithms for Shortest Paths in Sparse Networks," *J. Assoc. Comp. Mach.*, vol. 24, no. 1, pp. 1-13, 1977.
6. C. Witzgall, J. F. Gilsinn, and D. R. Shier, "Shortest Paths in Networks," in *Case Studies in Mathematical Modeling*, ed. W. E. Boyce, pp. 171-255, Pitman, Boston, 1981.
7. B. W. Kernighan, M. E. Lesk, and J. F. Ossanna, "Document Preparation," *Bell Sys. Tech. J.*, vol. 57, no. 6, pp. 2115-2136, 1978.
8. B. W. Kernighan, "PIC — A Language for Typesetting Graphics," *Proc. SIGPLAN/SIGOA Symp. on Text Manipulation*, pp. 92-98, Portland, Oregon, 1981.
9. C. J. Van Wyk, "A Graphics Typesetting Language," *Proc. SIGPLAN/SIGOA Symp. on Text Manipulation*, pp. 98-107, Portland, Oregon, 1981.
10. D. E. Knuth, *TEX — A System for Technical Text*, American Mathematical Society, Providence, R.I., 1979.
11. A. J. Goldstein, L. D. Harmon, and A. B. Lesk, "Man-Machine Interaction in Human Face Identification," *Bell Sys. Tech. J.*, vol. 51, no. 2, pp. 399-427, 1972.
12. T. Sakai, M. Nagao, and T. Kanade, "Computer Analysis and Classification of Photographs of Human Faces," *Proc. 1st USA-Japan Computer Conference*, pp. 55-62, 1972.
13. S. Brennan, "Caricatures from Images," *ACM Conference*, San Francisco, Cal., 1984.
14. R. L. Cook, "Shade Trees," *Computer Graphics*, vol. 18, no. 3, pp. 223-231, July 1984. SIGGRAPH '84 Conference Proceedings; the picture is printed in both the 1984 and 1983 SIGGRAPH meetings.
15. Alvy Ray Smith, "Plants, Fractals and Formal Languages," *Computer Graphics*, vol. 18, no. 3, pp. 1-10, 1984. SIGGRAPH '84 Conference Proceedings.
16. Azriel Rosenfeld, "Image Analysis: Problems, Progress and Prospects," *Pattern Recognition*, vol. 17, no. 1, pp. 3-12, 1984.
17. D. M. McKeown and J. L. Denlinger, "Graphical Tools for Interactive Image Interpretation," *Computer Graphics*, vol. 16, no. 3, pp. 189-198, 1982.