THE EVOLUTION OF A GRADUATED SYMBOL SOFTWARE PACKAGE IN A CHANGING GRAPHICS ENVIRONMENT

Wolf-Dieter Rase
Bundesforschungsanstalt für
Landeskunde und Raumordnung (BfLR)
Postfach 20 01 30
5300 Bonn 2
Federal Republic of Germany

ABSTRACT

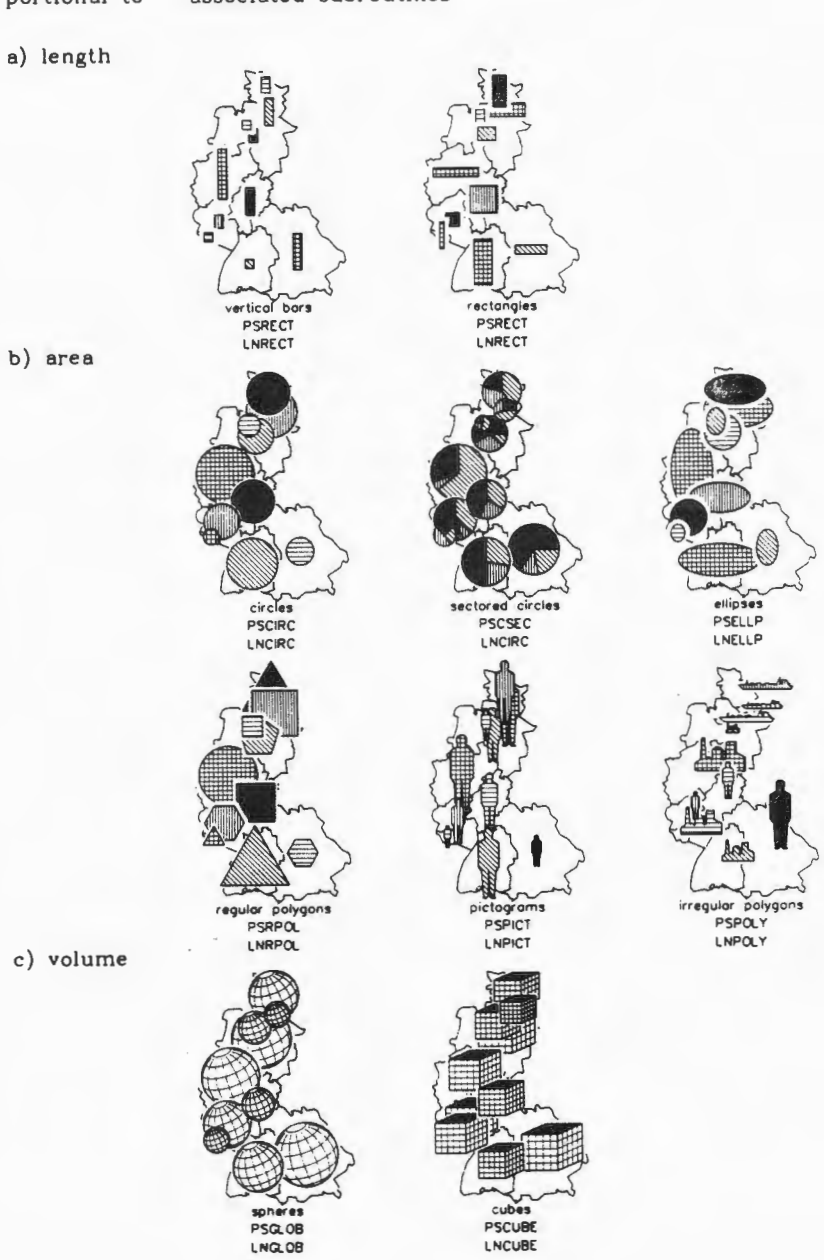
The paper describes the development history of a software package for computer-assisted drafting of graduated symbol maps. According to the prevalent machinery ten years ago the first version was realized on a minicomputer with vector-oriented devices, such as mechanical plotters and storage tube terminals. The programs provide functions for plotting graduate symbols of various forms, legends and line drawing. To improve map legibility the hidden areas in case of overlapping symbols are removed. The visual separation of the symbols is enhanced by a halo effect, a small gap between the symbols, or the symbols and map background. New hardware required reprogramming of the package to exploit the extended capabilities of new computers and graphic devices, including color and hardware area fill. To increase the flexibility, portability and maintainability the graphic standard GKS (Graphical Kernel System) has been used as a device-independent software interface for the third generation of the package.

Graduated Symbols in Thematic Mapping

The graduated symbol, also known as proportional symbol (Dent 1985), is considered to be the appropriate technique to map the spatial distribution of absolute values at discrete points. The point can also serve as a geometric representative for non-point references, e. g. administrative areas. The size of the symbol is proportional to the value of the variable associated with the location, either in length, area, or volume (fig. 1).

Besides the size other graphic variables, such as brightness, color, texture, or orientation, may be used to form the graduated symbol. The graphic variables can either be combined to increase the redundancy of the information, or used separately to transcribe several independent variables in a multivariate map. For the sake of readability, however, the number of variables should be limited; not more than three is a good rule of thumb. A recent discussion of the use of graduated symbols in computer-assisted thematic mapping can be found in Haldrup, 1985.

Fig. 1 Graduated symbols and associated subroutine names



Objectives for Computer-assisted Drafting of Graduated Symbols

The computer-assisted production of graduated symbol maps seems to be a simple task because the geometric construction of the symbols is well-defined. The formulas to calculate the coordinates of circles, ellipses, regular polygons and other forms can be found in most textbooks on elementary Analytical Geometry. To achieve a cartographic product comparable to hand-made maps, however, it is necessary to provide additional features besides the drafting of the symbols:

- Because the placement of symbols on the map is basically fixed overlapping cannot be avoided. The simulation of an arrangement in three dimensions (stacking of the symbols in the z-axis) enhances the legibility. To improve the impression of a perspective scene the hidden parts of the symbols should be eliminated.

- The readability is further enhanced by a small gap between overlapping symbols, or between the symbols and the map background (situation, boundary lines etc.). This small gap is known as a halo, and is standard practice in graphics and cartography.

- Sometimes it is necessary to modify slightly the position and orientation of certain symbols, for example if parts meaningful for the perception of the map are hidden by other symbols.

The cartographic requirements of map readability and legibility are complemented by economic considerations for the development and the application of the programs:

- To keep unit production costs low the programs for drafting graduate symbols should make moderate use of computer resources.

- A user-friendly interface is an important factor for minimizing production costs, both in respect to program development and production.

- The overall development costs including design, programming, testing and maintenance should be distributed over a long program lifetime and a large number of program users, again to minimize unit cost. The average lifetime of software is much longer than the average lifetime of hardware. Portability is not only important for the painless transfer to other computer systems, but also to subsequent generations of hardware within the own organization.

- Therefore the software must be general and flexible enough to cover various application environments, computer systems and graphic devices. No modifications should be necessary to implement the software on a computer system different from that of the developer.

During the last ten years software for computer-assisted drafting of graduate symbol maps has been developed and used at the Federal Research Institute for Regional Planning (BfLR). The development tought us some lessons on software lifetime, portability and standardization recapitulated in the following development history.

General Remarks on Hidden Surface Removal and Halo

Hidden surfaces

Hidden surface elimination is a common problem in computer graphics. Various algorithms have been devised to give efficient and naturalistic displays of complex 3D scenes (Sutherland et al, 1974). The general case of hidden surface removal in user space (maintaining the resolution of the input data) is rather complex and time-consuming. Perspective transformations, shearing and other pecularities have to be observed, let alone shading, illumination and shadowing. Shortcuts are possible by reducing the resolution to picture space, and by exploiting special features of the graphic output devices, for example opaque or transparent overplotting on raster devices, or specialized processors for 3D graphics. Increased efficiency is bought by increased device dependence, and subsequently by reduced portability and flexibility.

The removal of hidden surfaces is much easier with graduated symbols, because some of the special cases causing so much trouble in a general solution can be ruled out from the beginning:

- The symbols have zero thickness and zero distance in the third dimension. Viewing transformations from 3D space to the picture plane (and vice-versa for certain hidden surface algorithms) are not necessary.

- Shearing is impossible, because the plane of the symbol has the same orientation as the picture plane.

Another dimension of complexity can be eliminated by reducing the hidden surface problem to a hidden line problem. Using vector-oriented graphic devices area fill for a graduated symbol is simulated by hatching. The intersection points of a hatch line with an overlapping circle or polygon are easier to calculate than the intersection lines of two polygons and the subsequent recomposition of the visible difference polygon.

Halo effect

The halo, the small gap between crossing lines or overlapping surfaces, enhances the visual separation of the elements in a picture. The shadow-like appearance of the halo stimulates the impression of perspective and depth in the z-axis. It is used for example to improve images of wire-frame models in construction, architecture or other applications (Appel 1979, Akman 1981).

Again the special case of proportional symbols is much easier to handle than the general case: a temporary enlargement of the upper symbol during the output of the lower symbol or line is sufficient. Practical experience has shown that a halo with variable width is superior to a halo with fixed width. The variable width is set proportional to the area of the upper symbol.

The First Generation for Vector Devices

The algorithm for hidden line removal

The first generation of software for graduated symbol mapping was developed for a 16 bit minicomputer (PDP-11/45) and vector-oriented devices, such as a mechanical x-y-plotter and storage tube CRTs. As mentioned above the area fill for graduated symbols is simulated by hatching, and the hidden surface removal is reduced to hidden line removal. The algorithm works as follows:

1. Calculate minimum and maximum of symbol in y-direction. Set y-value for first hatching line to the minimum value.

2. Calculate intersections of hatching line with symbol outline. Assign intersection code of "0" to intersection (x-) values.

3. Determine overlapping symbols. Calculate intersections of hatching line with overlapping (upper) symbols. Assign an intersection code of "1" to left intersection (entry), and a code of "-1" to right intersection (exit).

4. Sort intersections (x-values and intersection codes) in ascending order of x-value.

5. Plot intersection line as long as the sum of current intersection codes equals 0. Do not plot intersection line while the sum is greater 0.

6. Increment y-value of hatching line by the distance DIST. Go to 2 until y-value of hatching line is greater YMAX.

7. Go to 1 until last symbol is plotted.

The best way how overlapping symbols are determined depends on the type of symbol. For circles the well-known distance condition is used, and rectangles are even simpler. For polygons the comparison of the enclosing rectangles narrows the potential candidates to a number small enough for an analytical intersection calculation.

To improve the visual appearance, especially if much overlapping occurs, the y-value of the first hatching line should be corrected to modulo DIST in step 1 to ensure a common starting point of the hatching pattern for all symbols.

Some other improvements and shortcuts may be added, for example an additional condition in step 3:

3a. If the left intersection value of the overlapping symbol is smaller than the left intersection value of the overlapped symbol, and the right intersection value of overlapping symbol is greater than the right value of the overlapped symbol, go to 6 (line completely hidden).

If the hatching line is not horizontal the rotation of the coordinates proved to be the most efficient solution. Of course the intersection coordinates must be retransformed before plotting. Whereas the intersection calculations for circles and ellipses are trivial, polygons have to be checked edge by edge. Precautions are necessary for irregular polygons, because more than two intersections per hatching line are possible.

The spheres are visualized by parallels and meridians resulting in circles and rotated ellipses in the picture plane. To intersect these curves an iterative approach, a relative of the bisection method (Gruenberger et al, 1965), proved to be superior to a purely analytical solution.

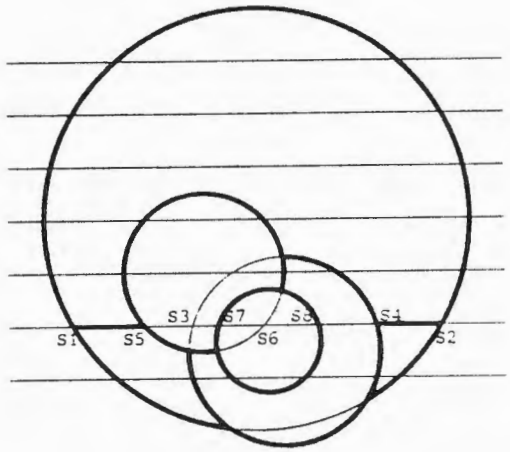
The outline of the symbols is treated in a similiar fashion. Again the intersections of two circles can be calculated directly. The intersection coordinates are sorted in ascending order of angles, with the associated intersection codes. The intersections of polygon outlines have to be calculated by comparing edge by edge. The hidden line removal is done in the same way as with the hatching lines.

Fig. 2 Example of algorithm applied on overlapping circles

S	x-value	code
1	0.5	0
2	7.5	0
3	2.8	1
4	6.2	-1
5	1.9	1
6	4.1	-1
7	3.2	1
8	5.2	-1

Sorted in ascending order of x:

			sum	action
1	0.5	0	0	line start
5	1.9	1	1	line end
3	2.8	1	2	
7	3.2	1	3	
6	4.1	-1	2	
8	5.2	-1	1	
4	6.2	-1	0	line start
2	7.5	0	0	line end



The graphic interface

At the time of development only device-dependent software was available for interfacing to output devices. The capabilities of the devices were rather restricted. Thus the software interface consisted of only two functions: draw vector, change pen. Although no standard has been used, the modification for other graphic devices than plotter or storage tube terminals was not a major problem. The calls could be easily emulated or replaced in the source code. In this respect the objective of portability was met, but not exactly in the sense we understand it nowadays.

Limitations

The small address space of the minicomputer (64 kbytes) on one hand and an acceptable throughput for floating point operations on the other imposed some bias on the use of computer resources:

- The programs had to be optimized for minimal memory requirements. As a trade-off the execution speed was not optimal, but still sufficiently low for the most common symbols and even interactive use.

- The software was written in Fortran, but language elements not defined in the language standard Fortran-66, such as 16-bit integers, were used to keep memory requirements low.

- The functions were implemented as a set of subroutines, not as a main program.

The implementation as a subroutine library improved the portability, but hampered the unrestricted use. To draw a map the user had to write his own main program reading in the data and calling the symbol and other functional subroutines. A fair amount of programming skills, DP experience and time was required, resources quite scarce among cartographers in a production environment.

The Supermini Generation

The 32-bit superminis with a much larger address space and virtual memory capabilities, in our case a DEC VAX-11/780, opened new perspectives for cost-efficient computer-assisted mapping. A second version of the package emerged. The programs were optimized for execution time, because memory was not longer the limit. Some deficiencies and minor problems were corrected, new features and functions added, which turned out to be necessary for production. The software interface remained unchanged. Only vector devices with their limited capabilities for area fill could be used.

A general main program was planned, but after the completion of the subroutine revision postponed to a later date. One reason was the advent of inexpensive color raster devices, the other the emerging graphic standard GKS. It became obvious that both developments would influence computer graphics to a great extent.

Thus a complete revision of the subroutines was necessary before a main program with an elaborate user interface or even an interactive design system would be feasible.

New Developments in Graphic Hardware and Software

Extended requirements

From the experience with the graduated symbol routines in actual production work it became evident that a new version of the software needed improvement in several respects:

- Some additional cartographic functions, for example for legends, were required.

- The programs should be able to exploit extended graphic capabilities provided by new output devices, such as different line styles, area filling with default or user-defined colors and patterns.

- Portability and maintainability should be enhanced.

The simple two-function graphics interface was not longer sufficient to fullfill the new requirements, and had to be replaced. The coding of the subroutines was not satisfactory following the principles of modern software technology, for example the excessive use of go-to statements. The intermediate version for the supermini had renounced on purpose the use of new language constructs in Fortran-77, because compilers for Fortran-77 were not generally available at that time. This restriction is not longer valid. Some of the "spaghetti code" is to be replaced by if-then-else constructs.

The graphic standard GKS

The only way to provide extended graphic capabilities and maintain portability is to utilize a generally accepted standard for interfacing graphic devices. The Graphic Kernel System (GKS) was chosen because it was the first graphic standard adopted by the important national and international standard organizations, among others ANSI, DIN and ISO. The concept of the "virtual workstation" liberates the user from the constraints of a specific real device. GKS provides the desired additional output functions for area fill and line styles. The elaborate input and picture manipulation functions furnish excellent tools for interactive map design and manipulation.

Our first enthusiasm about GKS was damped progressively with its application on our graphic software including the graduated symbol routines. We had to realize that such a general system is much more complex than the basic software for Calcomp plotters or the PLOT-10 subroutines for Tektronix terminals we were used to. There are too many traps that a GKS newcomer cannot avoid to fall at least into one. An application-specific shell would help, but this shell has to be a standard again to guarantee portability.

Defining a standard is one thing, implementing it something else. Using a validated GKS implementation the overhead was so immense that we decided to retreat to our own GKS emulation for production work. Originally the emulation was intended to be an educational tool with very restricted functionality. A selection of functions below what is specified in the "minimal" level of the ANSI standard is interpreted and transformed into calls to the device-specific basic software. This solution is certainly not elegant, but economic. There is hope that more efficient implementations will be available in the future. The GKS suppliers have learned their lessons, for example that Fortran is not the ideal system implementation language. More and more functions now executed in software will be provided by the workstations relieving the host of the elementary work.

Raster devices

As a consequence of the advancement in integrated circuit production, especially for microprocessors and memory chips, graphic devices based on raster technology became less expensive. For economic reasons, but also for superior performance, they replace the vector devices progressively, except in special market segments. With a device-independent interface such as GKS the application programmer can ignore the specifics of new hardware, and leave the support for the new devices to the GKS implementor.

But raster CRTs, inkjet and electrostatic plotters are able to simplify the hidden surface removal for graduated symbols considerably. The symbols are sorted topologically in the z-axis according to cartographic practice. Plotted in the right sequence from background to foreground, the smaller symbols overlay the larger symbols farther in the background. For the aforementioned reasons shearing is not possible. Complicated hidden line algorithms are not longer necessary, overplotting suffices.

On the other hand very many vector devices are still in use which cannot be neglected. Even the new graphic terminals, although internally raster devices, expect data in vector format, for examples all the look-alikes of the Tektronix 4010 series. The objective of portability is not met if the software is restricted to a specific type of output device. Although raster devices would simplify the hidden surface removal and save execution time, device independence has higher priority.

Segments in GKS

The Graphic Kernel System not only provides a standardized device-independent interface for input and output. The more advanced features allow the subdivision of the picture into parts called segments. A segment can be "picked" directly by a pointer device without special bookkeeping in the application program. Segments can be moved, copied or deleted independent of the rest of the picture. Highlighting of segments may be specified to attract the attention of the operator, or as a receipt for a successful pick operation. Priorities can be assigned to the segments for input, and the visibility of the output in case of overlapping segments.

If each graduated symbol is defined as a segment interactive manipulation could be realized quite comfortably. A symbol is located directly by the pick function. The segment can be moved without regeneration of the complete picture, provided the device supports that function. The visibility priority assignment is a restricted but adequate form of hidden surface removal if the segments are moved interactively.

As usual reality looks a little bit different from theory. One problem is the halo generation which cannot be implemented in a general way by GKS functions. Less obvious is the fact that segment priority in GKS does not necessarily include hidden line removal for vector devices. Thus most GKS implementations do not support it, for good reasons. The general case of hidden line removal is costly, both in programming and execution. For some cases of segment overlay singular solutions do not exist.

The Third Generation of the Graduated Symbol Software

Implementation guidelines

The previous discussion has shown that the third generation of the graduate symbol software should exploit the new developments in hardware and software, but has to compromise in several respects:

- The Graphical Kernel System is used as an device-independent software interface for graphic output, including area fill with colors and patterns.

- GKS segmentation should be allowed, but its actual activation left to the user. If the specific GKS implementation does not support segmentation, or the calling main program is restricted to graphic output, the programs should function in the usual way.

- The hidden surface removal can neither rely solely on the use of raster devices, nor on the segment priority feature of GKS. The algorithm for vector devices must still be a part of the software to maintain device independence and portability.

- To enhance portability and flexibility the language standard Fortran-77 is used, including if-then-else clauses.

The use of a programming language other than Fortran was never considered to be a serious alternative. Most of the code of the PDP-11 version needed only minor modification. The conversion to Pascal or ADA would have cost a fortune, besides the fact that GKS implementations for these languages are still a rare breed.

A general application program with an elaborated user interface and all functions necessary to produce maps without the need for writing an own program is the next logical step after completion of the subroutines.

Realized functions

To draw a graduated symbol map at least four groups of functions are necessary to obtain a satisfying result:

- graduate symbol plotting,
- line drawing with removal of parts hidden by symbols,
- plotting of legends for symbol form, size and filling,
- lettering.

The latter group is not specific for graduated symbol maps, and is not considered further in this context, as well as other graphic and geometric information on the map, such as situation, histograms or scattergrams. The following table gives an overview of the realized functions with their subroutine names. Each form has its associated line and legend drawing routine (fig. 3).

Fig. 3 a) Symbol b) Line drawing c) Size legend

Table of realized functions

Symbol type	subroutine name for plotting symbols	lines	legend
circles	PSCIRC	LNCIRC	LGCIRC
sectored circles	PSCIRS	LNCIRC	LGCIRC
sectored circles, ext.	PSCSEC	LNCSEC	LGCIRC
ellipses	PSELLP	LNELLP	LGELLP
rectangles	PSRECT	LNRECT	LGRECT
regular polygons	PSRPOL	LNRPOL	LGRPOL
irregular polygons	PSPOLY	LNPOLY	
pictograms	PSPICT	LNPICT	LGPICT
spheres	PSGLOB	LNCIRC	LGGLOB
cubes	PSCUBE	LNCUBE	LGCUBE
area fill legend			TYPLEG

In the practical application the circles, full and sectored, are the most used symbol forms. Although other forms proportional to area convey the same information circles are preferred. Obviously the human visual sense perceives smooth forms as more charming than cornered ones, a wide field for psychological speculation. Two different subroutines for sectored circles are provided, a simple one with all sectors having equal radius, and an extended one with individual sector radii, sector displacements, and radial subdivisions of the sectors.

Pictograms are symbols of arbitrary shape scaled proportionally to the variable to be mapped. The caller supplies the outline coordinates of the pictogram and the area the symbols should cover on the map. The subroutine scales the coordinates to the requested size, and positions the symbols at the points on the map specificied by the user. The coordinates of irregular polygons, however, must be specified in map space, including proper scale and position.

The volume symbols should only be applied on data with an unusually extended range. The visual transcription works best with areas, therefore the use of (pseudo-) volume symbols may cause misleading results in the interpretation of the map. As one may expect from the experience with the circles, spheres are preferred to cubes.

A function depicting pillars with a rectangular base will be one of the next extensions. The realization of the pillars, as well as other new symbols, is relatively easy. The subroutines in the table call a number of common functions for area fill and outline plotting, which can be used for symbols not yet contained in the library.

Usually one or two variables are transcribed on a map by size and filling of the symbols. Line style and color of the symbol outline may be used for a third (typed) variable. The number of variables can be extended by utilizing additional graphic variables, for example by varying

- the axis ratios of ellipses,
- the number of edges for regular polygons,
- the outlines of irregular polygons.

The variation in form and orientation by changing the axis ratio of ellipses is an unusual visualization method in mapping, because manual drafting of ellipses is too costly to be practical. But the unusual attracts attention, and attention is the first step to get the addressee to have a closer look at the map. Thus in this case computer-assistance is not only an economic advantage, but also provides a novel tool for transmitting the message more efficiently (an example can be found at Rase, 1980). Comparing the visual acceptance of ellipses and rectangles, smooth forms are preferred as expected.

Again a warning is issued not to overdo it with multivariate maps. Too many variables confuse or even embarass the reader. Subsequently the perception is hampered, or prevented completely. In most cases less is more, of course dependent on the message and the potential recipients.

Options

A set of options allows the user to control the output and the visual appearance of the graduated symbols. If applicable the options are common for all symbol types.

Topological sort. The original sequence of the symbols as supplied by the user is kept. The sequence is only changed if symbols overlap. Then the larger symbols are plotted prior to the smaller symbols. If the user insists on a specific sequence a field can be preset with sequence numbers. The same field contains the sequence numbers after execution of the program. These numbers, either preset or generated by the program, can be used to assign visibility priorities in case of segmentation.

Symbol filling. Hatching is the most general area filling option. It can be used with vector or raster output devices. The hatching angle, distance and offset can be specified by the user. Besides line styles and colors provided by the GKS implementation the user may select two arrangements of dashed lines (with variable dash lengths), or up to 14 small symbols (markers) placed at the intersection of a rectangular grid. The grid and the small symbols can be rotated with any angle. Area fill with colors or user-defined patterns is restricted to raster devices and by the specifics of the GKS implementation.

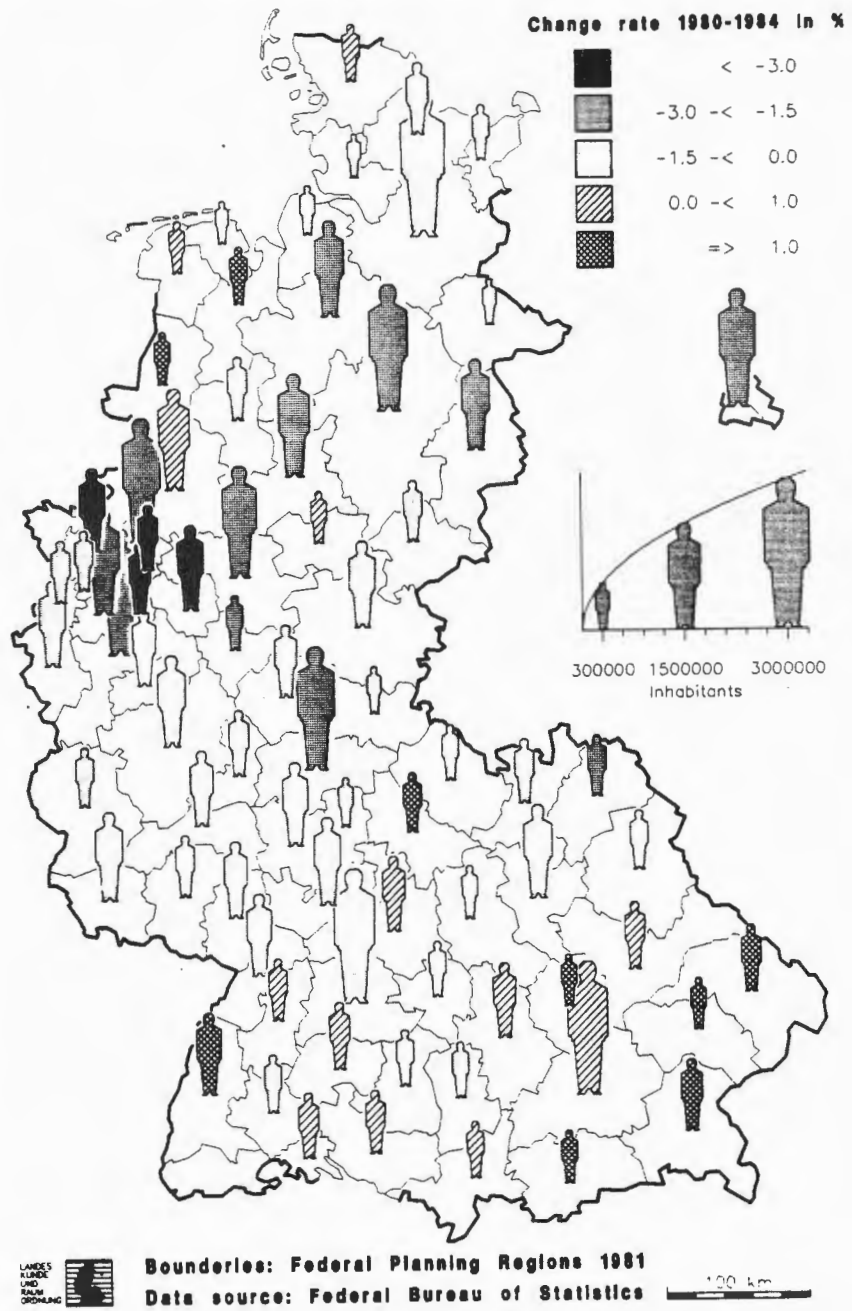
Outline plotting. Following GKS conventions color, line style and line width of the symbol outline is set by the caller prior to subroutine execution. A line index is transferred to the subroutine, either the same index for all symbols, or an individual index for each symbol. The latter feature allows the additional transcription of a type variable.

Halo. For vector devices the halo generation is performed during hidden line removal by a slight enlargement of the upper symbol(s). On raster devices (hidden surface removal by overplotting) a white surface with the same but enlarged outline is plotted prior to the plotting of the symbol. The halo width can be either fixed or proportional to the area of the symbol.

Symbol positioning. The symbols may be either centered on the reference points, or "standing", that means, the minimum y-coordinate of the symbol is equivalent to the y-value of the reference point.

Segmentation. In the Fortran binding of GKS segments are identified by numbers. If the caller wants segmentation he supplies the number for the first segment. The routines increment that number for each segment (symbol), and return the last value. The range of segment numbers can be used for picking and other segment-related operations in the calling program.

Fig. 4 Example for the use of pictograms



Finally the user has to specify if he uses a vector or raster device for output. The inquiry could also be done by a GKS function, but to enable special effects the parameter has to be set explicitly.

User interface at the BfLR

The subroutine package is the most general interface for the application of graduated symbol mapping. The individual user environment, for example in respect to the specifics in

- access of geometric and statistical data (data banks),
- the computer, its operating system and the attached graphic devices,
- the installed GKS implementation,
- established conventions and procedures for program interaction, control, and graphical output,

has not to be taken into consideration. The subroutines can be used in ad-hoc main programs, or incorporated into existing mapping systems. Both earlier and current versions have been implemented successfully at other institutions.

As mentioned above it cannot be expected from the average user to supply its own individual main program to plot the symbols, legends, boundary lines and other features on the map. A general way must be offered to produce a map with some form of control facility. In our case we took the command language of an existing choropleth mapping program and extended it for the graduated symbols.

The first version of this program called PROKAR (Schmalenbach, 1986) is in general use for several months now. As a first step it provides the most common symbols, together with functions for plotting legends, histograms, boundary networks, lettering, and map situation consisting of points, lines and areas. The program will be gradually extendend to cover the symbols not yet available. An interactive version is planned, but will certainly not be realized in the near future.

Conclusion

The described subroutine package provides an efficient set of tools for the computer-assisted production of graduated symbol maps. To distribute the development cost over a long software lifetime and a large number of applications special attention has been paid to standardization. The language standard Fortran-77 and the graphic standard GKS are intended to provide the flexibility, portability and maintainability necessary to satisfy both cartographic and economic requirements.

References

Akman, Varol, 1981, Halo - A computer graphics program for efficiently obtaining the haloed line drawings of computer aided design models of wire-frame objects, User's manual and program logic manual. Rensselaer Polytechnic Institute Image Processing Lab.

Appel, A., Rohlf, F. J., Stein, J. A., 1979, The haloed line effect for hidden line elimination. Computer Graphics, Vol. 13, No. 2, August 1979, 151-157

Dent, B. D., 1985, Principles of Thematic Map Design. Addison-Wesley Publishing Co.

Enderle, G., Kansy, K., Pfaff, G., 1984, Computer Graphics Programming. GKS - The Graphic Standard. Springer Verlag (2nd edition in preparation)

Gruenberger, F., Jaffray, G., 1965, Problems for computer solution. John Wiley & Sons

Haldrup, K., 1985, Generation and editing of proportional point symbols on a microcomputer (DEC 'Rainbow' 100). Thesis submitted to the International Institute for Aerospace Surveys and Earth Sciences (ITC), Enschede, The Netherlands, June 1985

Rase, W.-D., 1980, A Family of Subroutines for Plotting Graduated Symbol Maps. Geo-Processing, 1(1980), 231-242

Rase, W.-D., Steffens, L. J., 1986, PROSYM. Unterprogramme für die Zeichnung von proportionalen Größensymbolen, Version 3.0, Benutzerhandbuch. Bundesforschungsanstalt für Landeskunde und Raumordnung, Bonn, Federal Republic of Germany.

Schmalenbach, I., 1986, PROKAR V1.0, ein Programmsystem für die computerunterstützte Zeichnung von Karten mit Proportionalsymbolen, Benutzerhandbuch. Bundesforschungsanstalt für Landeskunde und Raumordnung, Bonn, Federal Republic of Germany.

Sutherland, I. E., Sproull, R. F., Schumacker, R. A., 1974, A characterization of ten hidden-surface algorithms. ACM Computing Surveys, Vol. 6, No. 1, March 1974, 1-55