ALGORITHM OF FLOWFIELD VISUALIZATION BY VECTOR GRAPHICS

M.S. IVANOV, A.V. KASHKOVSKY
Khristianovich Institute of Theoretical and Applied Mechanics,SB RAS, Novosibirsk, 630090, Russia
Corresponding author: Tel.: +7 383 330 81 63; Fax: +7 383 330 72 68 ; E-mail: sasa@itam.nsc.ru

KEYWORDS:
Main subjects: algorithm, flowfiled visualization
Fluid: scalar flowfield
Visualization method(s): vector graphics

ABSTRACT: The result of numerical modeling of many physical processes is a set of several fields of scalar quantities. Files with field visualization results usually have a large size. An algorithm is proposed, which allows a high-quality scalable image of scalar fields to be stored in a file of a comparatively small size in the PostScript language format or any other format supporting vector graphics.

1 Introduction

If a certain (usually real) number is put into correspondence to each point of a multi-dimensional space, then a scalar field is said to be defined in this domain. In numerical modeling of physical phenomena, scalar values are usually defined for centers or vertices of cells into which the computational domain is divided. A two-dimensional scalar field can be presented with the use of lines of different levels (isolines). Three-dimensional fields are usually presented as isosurfaces or isolines in a given secant plane. The space between the lines is filled with different colors depending on the scalar values, and a correspondence scale is given.

Two basic methods are used for image construction and storage:

Raster method. The image is a fixed-size grid (matrix) composed of raster points (pixels) with different degrees of brightness and different colors.

Vector method. The image is formed by a set of geometric primitives: points, lines, polygons, etc. Each geometric primitive has its own size and color. The primitive characteristics are its type, color, and a set of coordinates that describe the shape of the primitive.

The progress in computer performance and memory promoted an increase in the computational grid size. Modern grids sometimes contain hundreds of millions or even billions of cells. The most important phenomena often occur within several neighboring cells; therefore, it is important to provide a high resolution of the image during field visualization to capture these important phenomena. On the other hand, the resolution of various devices for printing images on paper is permanently improved, which allows detailed images to be published in journals or books. This procedure also requires high-resolution initial images.

If raster graphics is used, the resolution is improved by increasing the number of pixels. If a smooth change in the scalar quantity between two neighboring cells is needed, it is necessary to have several pixels in each cell. Correspondingly, the number of pixels in large-size cells can reach tens of millions. Even if rather effective compression algorithms (PNG, JPG) are used, the file size often reaches several megabytes. Files with books, reports, or papers containing a large number of such illustrations are so huge that it is difficult to send them by e-mail, read on the monitor, and especially to print them, because the velocity of data transmission to printers does not grow so rapidly as the processor speed.

The use of vector graphics allows storage of high-quality images, which can be easily scales without resolution deterioration. In most cases, visualization programs (see, e.g., [1, 2, 3]) consider each cell of the computational domain separately. With the use of vector graphics, the image in each cell is formed by one (in the simplest case) or several primitives. For each primitive, it is necessary to store at least its type and a set of its coordinates. For grids with millions of cells, therefore, huge files multiply exceeding the pixel image size are obtained.

The following questions arise in such a situation. It is possible to avoid considering each cell separately. Instead, is it possible to form the image for rather large domains of the same color, thus substantially decreasing the number of primitives and the file size? This is the idea implemented in the algorithm described below, which allows obtaining files with an uncompressed vector image, which are commensurable in size with compressed raster files.

2 Construction of isolines

To form an image for a sufficiently large one-color domain, it is first necessary to determine the boundaries of the latter. The domain boundaries are lines of an identical level (isolines) constructed for a specified set of values.

Let a scalar field be specified on a grid composed of triangles, with known values at triangle vertices.In practice, it is more convenient to have a list of vertex coordinates and scalar value(s) (there may be several of them) at the vertices, and three reference (indicator, address, or index, depending on implementation) to these vertices are stored for a triangle. If the grid consists of quadrangular cells, there is no particular need to transform this grid to a triangular one. In considering such a cell, it is sufficient to divide it virtually and temporarily into several triangles. For a quadrangular grid, such splitting into two triangles can be easily performed by drawing a diagonal. For each triangle, it is necessary to know triangles having a common boundary with it. As the problem is solved in a plane and there are no intersecting triangles by definition, each edge can have only one neighboring triangle.

If a structured grid is used, the neighboring triangle can be easily calculated on the basis of grid topology. In the case with an unstructured grid, the neighboring triangles are found in advance, and three references (for each edge of the triangle) to neighbors are stored for each triangle. In the general case, the operation of determining neighbors has a complexity of the order of O(N^2), where N is the number of triangles, but there is also a fairly simple algorithm with a complexity O(N).

1. For each vertex, triangles including it are determined:
   (a) a counter of triangles to which the vertex belongs is created for each vertex;
   (b) All triangles are checked, and the counter of triangles of each vertex is increased by one;
   (c) For each vertex, a certain amount of memory is allocated for the list of triangles including this vertex in accordance with the counter of triangles;
   (d) The triangles are checked again, and the number of the triangle is put on the list of three vertices forming this triangle. It is important that the numbers in the lists are arranged in increasing order, because the triangles are checked in increasing order.
2. The triangles are checked again to determine their neighbors:
   (a) For each side of the next triangle, two vertices connected by this side are taken;
   (b) A search for coincidences is performed in the lists of these vertices. As was mentioned above, the lists are ordered; therefore, they are checked simultaneously: the list of the triangle with the smaller number is checked;
   (c) A coincidence means that the side belongs to the triangle with this number. If the side belongs to two triangles, there are two coincidences. One of the numbers coincides with the number of this triangle, and the second coincidence is the number of the neighboring triangle on this side. If the side belongs to the domain boundary, then there is one coincidence.

If the values of some scalar quantity in the vertices are known, the most natural way of approximation of this quantity inside the triangle is linear interpolation. Then, the isoline inside the triangle is a segment of a straight line and, thus, can cross only two sides of the triangle. It is better to perform isoline interpolation or approximation, if needed, for the entire isoline than at a level of the triangle. The edge intersection points are determined by linear interpolation. To construct the next point, it is necessary to pass to the neighboring triangle that has a common side with the current triangle.

Rather often (especially for grids with very small cells), the isoline remains close to a straight line within several cells. In this case, it is sufficient to store and use the first and last points of such a segment, and all intermediate points (which belong to this segment or are very close to it) can be omitted from considerations.

Isoline construction is continued until the next point coincides with the starting point ("closed" isoline) or until the point is found to be on the grid boundary ("open" isoline). After that, the beginning of a new isoline is sought by means of successive checking of the cells, and the process is repeated. The cells in which the isoline of a given level is already constructed are marked, and such cells are skipped in searching for a new isoline of the same level. For reasons that will be explained below, it is important to known in isoline construction whether the domain with greater values is located on the right (let us call it the "forward" direction of the isoline) or to the left ("backward" direction).

With isolines being constructed, it is possible to create a good-quality image. Figure 1 shows the density field of a supersonic gas flow formed by irregular (Mach) reflection of an oblique shock wave from a solid surface. The calculations were performed on a uniform rectangular grid 1000×500 cells. The image is formed by 50 isolines. The size of the uncompressed vector PostScript file of this image, which was obtained by the method described above, is 25 Kb. The same image converted to a PNG file with a resolution of 2000 × 1000 pixels (four pixels per cell) is 81 Kb. Certainly, we can talk about the degree of compression and smoothing and about the image resolution, but it is clearly seen that an uncompressed text file containing readily scalable vector graphics for a real problem may turn out to be several times smaller than the corresponding pixel image.

In academic papers, however, one color in the field most often corresponds to a certain range of values. The entire image is colored, and the changes in the scalar quantity are estimated by analyzing the changes in color. Actually, the space between two neighboring isolines should be "filled" with an appropriate color (like it shown of Fig. 2). For this purpose, single-color domains should be identified.

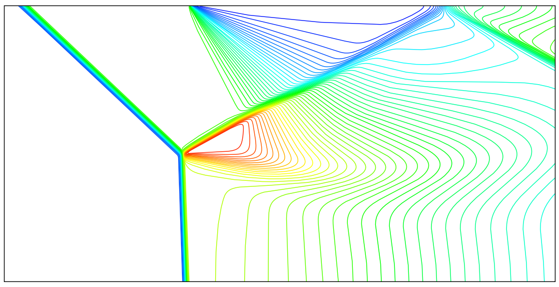

Figure 1: Isolines. 50 levels

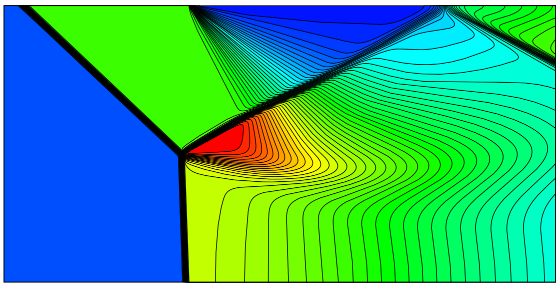

Figure 2: Isolines filled by collors. 50 levels

3 Construction of single-color domains on the basis of isolines

Each two neighboring isolines are boundaries of a single-color domain. As isolines do not cross each other, some part of the boundary has to be completed. If isolines are only open, then the boundary is completed along the domain boundary. In practice, however, there are no fields with open isolines only. There are always closed isolines present. Most probably, there exist some algorithms for connecting two (and more) nested closed isolines with an additional segment. The search for such a segment, however, is a rather complicated task in the general case. It should not cross other isolines and may be absent altogether: if the last isoline of the level is considered, its entire internal domain is filled with one color, and the search for other isolines in this domain is meaningless. Thus, connecting two neighboring isolines requires a fairly complicated analysis of the scalar field and an algorithm that takes into account various possible specific features of this field.

Another approach to image formation is proposed here. In this approach, smaller domains are drawn above larger domains; thus, the image is formed. Indeed, let us consider three neighboring closed isolines L_1, L_2, and L_3, with the levels $L_1 < L_2 < L_3$. Let L_1 have the greatest area. Obviously, L_2 is inside L_1, and L_3 is inside L_2. L_3 cannot be located inside L_1 without being simultaneously located inside L_2: this would mean a jump in the value or several values at one point of the field. As these figures are nested, their boundaries do not intersect, the area of L_2 is smaller than the area of L_1, and the area of L_3 is smaller than the area of L_2. Thus, drawing all three figures in decreasing order in terms of their area (L_2 above L_1, and L_3 above L_2), we obtain a correct pattern of alternation of isolines. Obviously, this approach is also valid in the case with $L_1 > L_2 > L_3$.

An undoubted advantage of this approach over the method of identification of single-color domains is its simplicity: there is no need to construct additional segments, to analyze the mutual location of isolines, etc. Moreover, this approach offers another principal advantage. In identifying single-color domains, each isoline is a boundary between two domains: with a greater value of the examined quantity and with a smaller value, i.e., the coordinates of this isoline are enumerated twice. In the method proposed here, there is one boundary of the figure, and the isoline coordinates are enumerated only once, which halves the image size.

As the boundary of the figure should be closed, open isolines are completed to become closed isolines. The open isoline is supplemented with a segment of the computational domain boundary, which connects this isoline with an isoline of the same level. After that, the boundary of the figure goes along the new isoline up to the next point on the boundary, passes over the boundary, etc., until it becomes closed on the initial isoline. The order of this round pass is chosen so that the domain with greater values is on the right of the boundary ("direct round").

As a method for automatic construction of closed isolines only, we can proposed to add a dummy series of thin cells along the entire domain boundary with the values of the scalar quantity being many times (e.g., million times) smaller than the value of the smallest specified level; in this case, there will be never any open isolines, and the lines along the domain boundary will almost coincide with the latter.

After all domains are formed, it is necessary to calculate their area. It is obtained by integrating the areas of trapezoids whose vertices are the vertices of each segment and projections of these vertices onto the lower boundary of the domain:

$$S = \sum_{i=1}^{N} (X_{p_{i+1}} - X_{p_i}) \cdot (Y_{p_{i+1}} + Y_{p_i})/2$$

(N is the number of segments of the isoline; X_p and Y_p are the coordinates of the point P.

As the domains are described by an ordered set of points, some trapezoids (with an increasing X coordinate) have positive areas, and others have negative areas. With the clockwise round, the area of the figure is positive. As was mentioned above, the isolines are constructed so that the greater values are on the right; therefore, this domain should be filled with a color corresponding to the higher level. If the area is negative, it means that the round is counterclockwise, there is a region of reduced values inside, and the domain is filled with a color corresponding to the previous (lower) level.

The domains are ordered with the help of an arbitrary algorithm of sorting in accordance with the absolute value of the area. Drawing the field begins from filling the entire computational domain with a color that should be "outside" the greatest domain. After that, all domains are drawn in decreasing order in terms of their area.

Figure 3 shows the same field as that in Fig. 1, but with domains filled with various colors. Fifty levels are again used. The size of the file in the PostScript format is 25.4 Kb. As the points of the isolines are enumerated only once, the file size is practically the same (greater by 2%) as the size of the file with the isolines only. The same image converted to a files of the PNG format with a resolution of four pixels per cell (2000 $\times$ 1000) has a size of 72 Kb.

Obviously, the file size increases with increasing number of levels (number of isolines). Thus, the size of the file with the image of the same field constructed with 250 levels (see Fig. 4) in the PostScript format is 120 Kb. In the PNG format with a resolution of 2000 $\times$ 1000 pixels, the file size is 173 Kb.

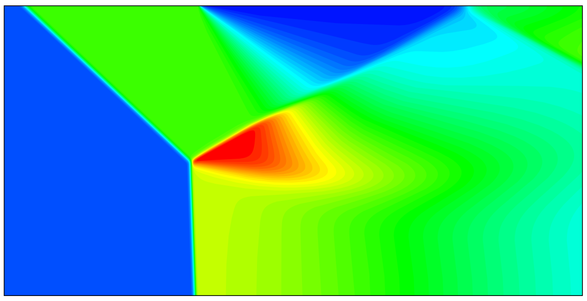
Figure 3: Filling with colors, 50 levels

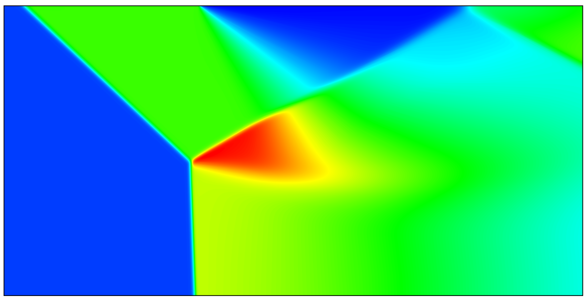
Figure 4: Filling with colors, 250 levels

4 PostScript language

The proposed algorithm is developed for storing images in the PostScript language format, but it can be applied to any other vector graphics format. The file size can be substantially reduced by using the possibilities of the PostScript language appropriately.

The PostScript language [4, 5] was developed by John Warnock at Adobe Systems in 1982 as a simple standard language for describing the text, drawings, and simple images on the printed page. It is important that the description of pages in PostScript is independent of the device on which the page will be reproduced. Files in the format of this language are used as illustrations in printed arts, in particular, in the LaTeX document preparation system (it is in this system that the present paper was prepared). There are numerous programs for displaying these files or for their conversion to the PDF format (also developed by Adobe Systems and actually being a "compressed" version of PostScript).

A PostScript device (printer, monitor) is a device including a PostScript language processor. The language processor takes a **text** file with a code written in the PostScript language from the computer and transforms it to a pixel form in accordance with the resolution of this device. After that, it is printed or displayed on the monitor.

The data in PostScript are numbers, arrays, rows, and symbols. Manipulation with these data is performed through a stack into which data (operands) from the initial file are first fed and an operator, which chooses a necessary number of operands from the stack and urges the language processor to perform these or those actions. The numerical result of operator execution (if any) is also placed into the stack, with the operands already stored in the stack being shifted. For instance, the arithmetic operation $27/(4+5)$ looks as follows:

```
 4 5 add 27 exch div
```

Here, the operator `exch` exchanges the result of addition with the help of the operator `add` and the number 27 before the operation of division `div`. Such style of programming, where the operands are defined earlier than the operator, is called postfix notation or reverse Polish notation. The set of operands or operators can be united into procedures. For instance, the notation

```
/R { 1. 0. 0. setrgbcolor} def
```

creates the procedure **R** whose initiation establishes the red color as the current color.

The image is formed in the coordinate system with the origin in the bottom left corner of the image, the X axis is directed to the right, and the Y axis is directed upward. The set of operators is rather large, and only the key operators affecting image formation within the framework of this algorithm are considered in Table 1.

The listing of the file given below shows a red trapezoid encircled with a black frame (see Fig. 5). After a necessary title, which is a sign of a PostScript file, the first four lines of this listing are responsible for drawing the trapezoid itself: the color for filling the figure is defined (red), the trapezoid contour is drawing, and the contour is filled with the color. In the next four lines, the contour is drawn with the black color in a similar manner:

Let it be one of the domains to be displayed. Bearing in mind that there are many domains of this kind, the description of the figure can be written more compact, using the following capabilities of the PostScript language:

- As all numbers are given with three decimal digits, it is possible to use the scale of 0.001, which allows passing to integer numbers, avoid the dot with retaining the same accuracy, and reduce the number recording by one byte.

Operator	Operands	Purpose
scale	sx sy	Set x and y scales of the image
setrgbcolor	r g b	Set the current color defined by the intensity of red (r), green (g), and blue (b)
setlinewidth	w	Set the line width equal to w
moveto	x y	Move to the point x,y without drawing
lineto	x y	Draw a line to the point x,y
rlineto	rx ry	Draw a line to the point shifted by rx,ry with respect to the current location
closepath	-	Connect the first and last points, thus, creating a closed figure
fill	-	Fill the closed figure with a color
stroke	-	Show the image of the figure formed
showpage	-	Finalize the page image
gsave	-	Save the current state
grestore	-	Restore the saved state

Table 1: Most important operators of the PostScript language

```
%!PS-Adobe-2.0
2 setlinewidth

1 0 0 setrgbcolor
121.896 286.123 moveto 192.574 286.123 lineto
192.574 457.843 lineto 182.574 457.843 lineto
closepath fill stroke

0 0 0 setrgbcolor
121.896 286.123 moveto 392.574 286.123 lineto
392.574 457.843 lineto 352.574 457.843 lineto
closepath stroke

showpage
```

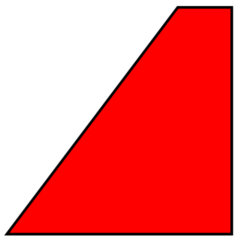

Figure 5: Example of the trapezoid image.

- All repeated operators (or their combinations) can be re-defined with single-letter procedures.
- Drawing from the first point is performed in relative coordinate; in this case, one of the coordinates for horizontal and vertical lines will be 0, which occupies one byte in the file. Moreover, as the isoline usually consists of small (as compared with the domain size) segments, the relative coordinates require a smaller number of symbols for their presentation than the absolute coordinates.
- After drawing the figure, it is possible to save the current state by using the gsave operator, fill the domain with a color, then restore the state by using the grestore command (after that, the image of the red trapezoid is already formed, but the language processor is in the state when the figure is only defined), change the color, and draw the contour. With this procedure of drawing, there is no need to enumerate the coordinates of the line for the contour again.

The listing for such a file is shown below:

```
%!PS-Adobe-2.0
/B {1 0 0 setrgbcolor} def
/C {0 0 0 setrgbcolor} def
/M {moveto} def
/R {rlineto} def
/D {closepath gsave fill stroke grestore C stroke} def
0.001 0.001 scale
2000 setlinewidth

% This is direct plot of the trapeze
B 121896 286123 M 170678 0 R 0 171720 R -41000 0 R D
```

```
showpage
```

It is seen that the text directly responsible for drawing (definitions of the procedures can be ignored because they are used only once in the file) decreased five-fold: from 259 to 53 bytes. A similar procedure allows drawing fields with color filling and with isolines, with the file size being not increased. Figure 2 shows the same field as in Fig. 3, but with the isolines shown by the black color. As the difference in only in the procedure used, the file size is absolutely the same.

5 Comparison of the file size of images obtained by different methods

The size of the file with the vector image obtained by using the above-described algorithm depends more on the distribution of the scalar quantity than on the grid size. For instance, Figs. 6 and 7 show the distributions of pressure and Mach numbers (ratio of flow velocity to velocity of sound) obtained in studying the viscous structure of the flow in the case of the Mach reflection of the shock wave from the plane of symmetry (see [6]) for the flow Mach number M=4. The modeling was performed on the basis of the numerical solution of the Navier–Stokes equations for a compressible gas with the use of a shock-capturing numerical method explicit in time and based on a fifth-order WENO scheme for convective terms and the fourth-order compact central approximation for diffuse terms. The numerical algorithm used was described in [7]. The size of the computational domain was 1600 $\times$800 (1.28 million) cells. The sizes of the files with the image constructed for 250 levels in the PostScript format are 121 Kb and 88 Kb, respectively (the difference is about 30%).

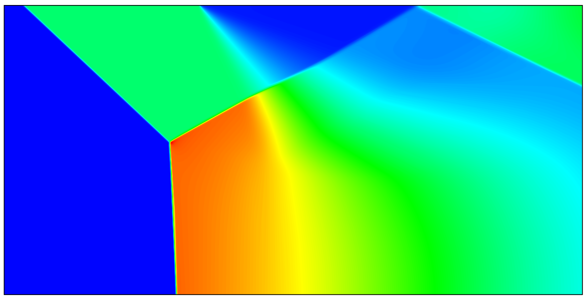
Figure 6: Pressure, 250 levels

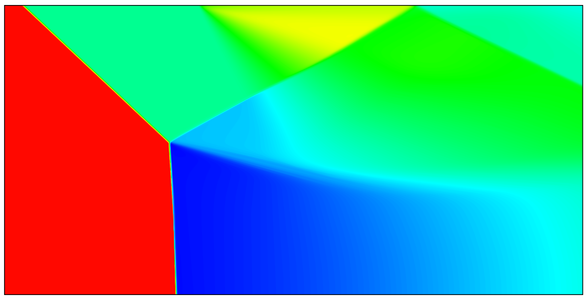
Figure 7: Mach number, 250 levels

The use of other methods of modeling physical processes, however, can lead to a significant increase in the file size. For instance, Figs. 8 and 10 show the results of modeling an axisymmetric flow around a promising reentry vehicle for an altitude of 75 km and velocity of 7.6 k/s, which was performed with a SMILE software system [8] by the Direct Simulation Monte Carlo (DSMC) method [9], [10]. The computational grid consisted of 1490 $\times$1200 (1.79 million) cells

Figure 8 shows the density field constructed for 50 levels. The file size is 199 Kb. Figure 9 shows the temperature distribution, also for 50 levels. The file size is 492 Kb. Though the grid size is only 40 % greater than in the previous example, the file sizes increase more appreciably (despite the use of a smaller number of isolines), and the difference in the sizes of files for different physical quantities in the same calculation is also significant. Such a considerable increase in the file size occurs because of statistical fluctuations, which are always inherent in DSMC computations and because of which the isolines have a “wavy” structure. There are also many small “spots,” i.e., local dips and humps of temperature. If spots whose area is smaller than the area of four cells are not stored in the file, then the image shown in Fig. 10 is obtained. The size of the resultant file is 400 Kb, which is smaller almost by 20 %. The effect of statistical noise is clearly seen in Fig. 11, which shows a fragment of the image in Fig. 10 magnified by a factor of 10 (near one isoline slightly above the trailing edge of the space vehicle). The isoline is shown by the black color, and its complicated configuration, which requires a large number of segments for drawing, is clearly seen. In this case, the file size can only be reduced by smoothing, i.e., replacement of the isolines by smoother curves with a smaller number of segments. This illustration again demonstrates the advantage of the vector presentation of data: even significant magnification does not affect the image quality, which is not always possible in pixel images.

In the last example, for 50 levels of isolines, the file size is still acceptable for a paper. With increasing number of levels, however, the file size rapidly increases to undesirable values. Thus, the file size is 833 Kb for 100 levels and 2.1 Mb for 250 levels. The pixel versions of these illustrations have much smaller sizes, but the difference is still visible, as was demonstrated with pilot printing of the vector image and of the pixel image obtained from it (with a resolution of 1490 $\times$1200) on a printer with a resolution of 600 dpi.

As the algorithm presented in this paper is always contrasted with the pixel image, Table 2 gives the file sizes for all illustrations in this paper, which were obtained by the algorithm described above, converted to the PNG format (JPG files have practically the same size), and, for some fields, obtained by the traditional algorithm (visualization of each cell independent of other cells).

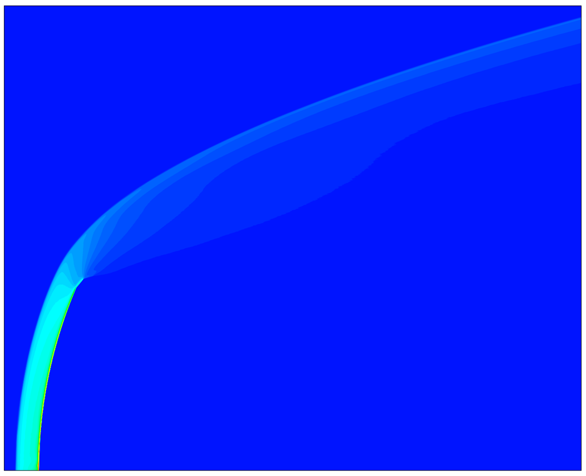
Figure 8: Density, 50 levels

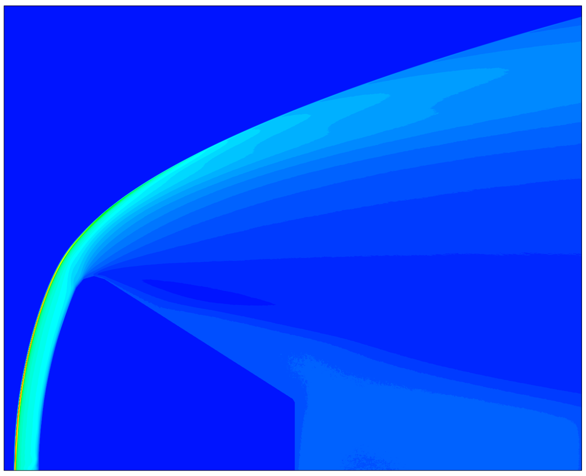
Figure 9: Temperature, 50 levels

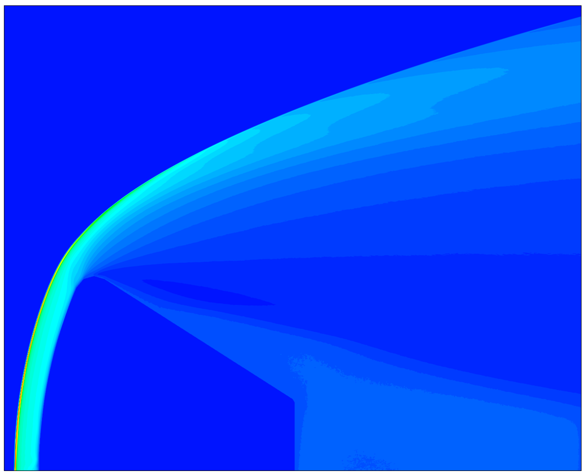
Figure 10: Temperature, 50 levels, filter

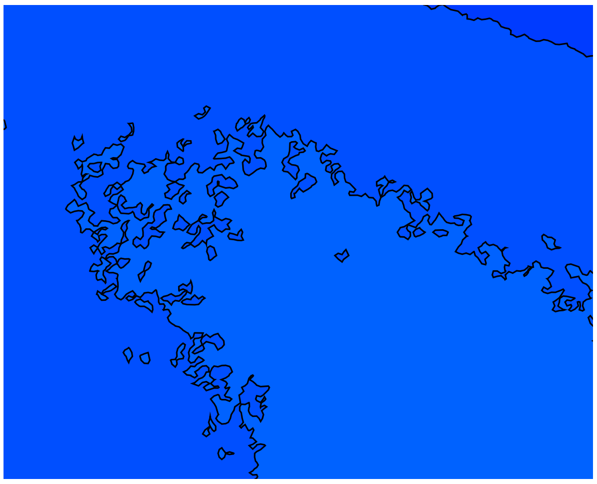
Figure 11: Temperature, 10-fold magnification

The files in the column entitled "Traditional algorithm" were obtained by three different codes indicated by the corresponding index in brackets: tecplot (t), gnuplot (g), and kapic (k); the latter code was developed by the author. As each code uses its own notation of Postscript operators, the file sizes can substantially differ from each other.

The file sizes for Figs. 10 and 11 obtained by the proposed algorithm are identical, because these are the same files, with the only difference that the second file contains PostScript operators of ten-fold magnification, shifting to a necessary point, and truncation of the image going outside the limits of the "window," and the procedure with drawing isoline contours is used (as they are both written in the file, they were simply renamed).

A comparison of the sizes of files obtained by the proposed algorithm and the traditional algorithm is impressive. In particular, for fields on the 1600 $\times$800 grid, the file obtained by the proposed algorithm is smaller by a factor of 700! It is seen that the traditional algorithm is unsuitable for large grids altogether. Illustrations greater than 60 Mb cannot be used in papers. Such a large size can be easily explained. In the traditional algorithm, each cell should be drawn. Let it be a quadrangular cell of one color. For drawing this cell, it is necessary to enumerate the coordinates of its vertices (4 $\times$2 = 8) and invoke the drawing procedure. Even if each coordinate and procedure name is presented by one symbol, there are 18 bytes together with separating spaces. If the cell contains isolines, the size increases in proportion to the number of the resultant domains. Therefore, the file size for a field presented by a million of cells is several times greater than 18 Mb, which is actually seen from Table 2.

Except for DSMC computations, the size of files obtained by the proposed algorithm is smaller than or commensurable with the size of files in the PNG format. For presentation of DSMC results, it is possible to use both vector graphics and high-resolution PNG files. In each particular case, however, the most important parameter should be chosen: quality of the image or file size. Availability of a vector image (even in a large file), however, substantially facilitates the procedure of creating pixel files with a required resolution. Nevertheless, the use of smoothing of DSMC results allows considerably reduction of the file size and is preferable.

As it was mentioned above, development of the proposed method of compact visualization of scalar fields was driven by the desire to have illustrations in vector graphics in small-size files to be used in publications. This paper was prepared in the document preparation system LaTeXand contains ten illustrations of sufficiently large scalar fields. The size of the

Number of the figure	Grid size	Number of levels	File size (in kB):			
			Proposed algorithm	Traditional algorithm	PNG	
					resolution	size
1	1000 ×500	50	25		500 ×250 2000 ×1000	15 84
2	1000 ×500	50	25.4		2000 ×1000	103
3	1000 ×500	50	25.4	17793(g),21093(t)	2000 ×1000	72.3
4	1000 ×500	250	120	29202(t)	2000 ×1000	173
6	1600 ×800	250	121	68409(t),89975(g)	1500 ×756 3000 ×1512	100 267
7	1600 ×800	250	88	67947(t)	1500 ×756 3000 ×1512	87 196
8	1490 ×1200	50	199	57714(k)	1490 ×1200 3000 ×2416	34.5 98.6
9	1490 ×1200	50	492	60010(k)	1490 ×1200 3000 ×2416	59.2 162
10	1490 ×1200	50	399		1490 ×1200 3000 ×2416	52.8 148
11	1490 ×1200	50	399		1490 ×1200 3000 ×2416	47 117
	1490 ×1200	100	833		1490 ×1200 3000 ×2416	76.5 211
	1490 ×1200	250	2110		1490 ×1200 3000 ×2416	132 364

Table 2: File size for different fields and algorithms

file with this paper is 2.2 Mb in the PostScript format and 1.7 Mb in the PDF format.

Conclusions

The proposed algorithm of visualization of scalar fields with the use of vector graphics allows high-quality scalable images of fields to be stored in moderate-size files. This algorithm is recommended for programs of visualization of the calculated results and for preparing illustrations for publications.

Acknowledgement

The authors is grateful to P. V. Vashchenkov and D. V. Khotyanovskii (researchers at the Khristianovich Institute of Theoretical and Applied Mechanics of the Siberian Branch of the Russian Academy of Sciences, Novosibirsk) for the results of calculations.

References

[1] Tecplot, Official website [Electronic resource], http://www.tecplot.com/

[2] Gnuplot homepage, Official website [Electronic resource], http://www.gnuplot.info/

[3] GL2PS: an OpenGL to PostScript printing library, Official website [Electronic resource], http://geuz.org/gl2ps/

[4] F. Deumling and D. Sillescu, *PostScript programming language* (translation from german), Fizmatlit, Moscow, 1993.

[5] PostScript language reference manual, Adobe Systems, 2nd ed.

[6] D.V. Khotyanovsky, Y.A. Bondar, A.N. Kudryavtsev, G. V. Shoev, and M. S. Ivanov, *Viscous effects in steady reflection of strong shock waves.* AIAA J., 2009, Vol. 47, No. 5, pp. 1263– 1269.

[7] A.N. Kudryavtsev and D.V. Khotyanovsky, *Numerical investigation of high speed free shear flow instability and Mach wave radiation.* Int. J. Aeroacoustics, 2005, Vol. 4, Nos. 3-4, pp. 325-344.

[8] M.S. Ivanov, G.N. Markelov, and S.F. Gimelshein, *Statistical simulation of reactive rarefied flows: numerical approach and applications.* AIAA Paper No. 98–2669, 1998.

[9] G.A. Bird, *Molecular Gas Dynamics.* Clarendon Press, Oxford, 1976.

[10] G.A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows.* Clarendon Press, Oxford, 1994.