Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

I. Herman

2.5 Dimensional graphics systems

Computer Science/Department of Interactive Systems Report CS-R8921 May

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

2.5 Dimensional Graphics Systems

Ivan Herman
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

The outline of an extension of traditional 2D graphics systems is given. This extension is aimed at supporting a three dimensional application program, without incorporating full viewing into the general graphics system itself. The resulting system might be very advantageous for large application programs which have their own three dimensional facilities.

1983 CR Categories: G.0, I.3.2, I.3.5
Keywords & Phrases: computer graphics standards, projective geometry in computer graphics, computer graphics systems.
Note: The present text is to be published in: *Eurographics'89 Conference Proceedings*, eds. F.R.A. Hopgood and W. Strasser, North Holland, 1989.

1. Introduction

General purpose graphics systems and standards, like GKS[1] , GKS-3D[2] , PHIGS[3] , CGI[4] or any of their ancestors like Core, can be classified rigorously as either 2D or 3D systems.

The notion of 2D systems means that function definitions contain graphics output primitives for two dimensions only; lines, markers, polygons etc. defined on a plane. These primitives may be fairly complicated, like stroke precision characters or polygons with pattern fillings. However, the definitions of these primitives reflect the planar nature of graphics: patterns are usually defined as a regular set of small parallelograms, dividing in turn a larger parallelogram regularly etc.

3D systems usually contain the more or less obvious generalisations of the two dimensional output primitives: polygons, text etc. are defined with three dimensional coordinates. Furthermore, these systems contain different forms of viewing facilities; the ultimate goal of these features is to project the three dimensional primitives onto the display screen which is, at present, inherently two dimensional. In the course of this process the system may also perform some kind of hidden surface and/or hidden line removal.

It is relatively straightforward to find application areas which may use a 2 dimensional package; a number of practical problems are basically planar, like e.g. business graphics, some areas of presentation graphics and even some CAD applications; electronic design is a good example.

This not exactly the case for three dimensional systems. There are of course application areas which are inherently 3D, like mechanical design or architecture. However, these systems may find it more natural to use a two dimensional general graphics system for their own purposes rather than a three dimensional one. This contradiction is the result of the fact that a sophisticated application program (e.g. a solid modeller) has to create some kind of internal representation of the objects it wants to manipulate; an internal representation which contains, among other things, a whole range of strictly geometrical data. Consequently, it may happen (and, in fact, it does happen), that the application program has the possibility to perform all viewing, hidden line/surface removal etc. of its own with a significantly higher speed, because it can make use of the sometimes rather detailed additional information which is within the internal representation. Using a full three dimensional graphics system would mean in this case the duplication of e.g. the viewing pipeline; a duplication which might be expensive both in terms of efficiency as well as in price.

The result of this fact is that such systems might find it more straightforward to use a two dimensional system instead of a three dimensional one. However, in this case, one faces a disturbing

problem: a number of complicated but at the same time very useful features of these systems (like GKS) are not usable any more! In fact, primitives like cell array, polygon filling with patterns, stroke precision characters are defined in a way, that it is impossible to use them to generate the *planar projection* of their three dimensional counterpart. As an example, it is not possible to generate directly a picture like the one in Figure 1 with a traditional two dimensional system. In other words the application has to solve e.g. the pattern filling of a polygon by itself, although this is clearly in contradiction with the usual demand of using general purpose graphics systems whenever it is possible to reduce development time and costs.

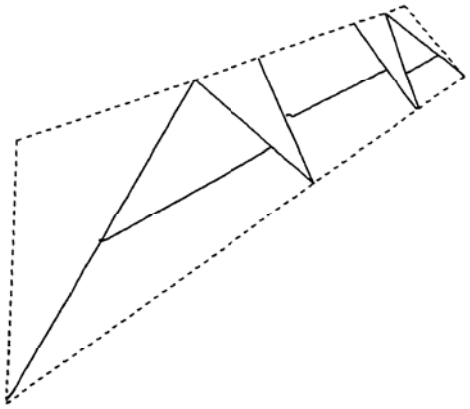

Figure 1

It might be therefore interesting to see whether it is possible to define an extension of the traditional two dimensional systems so as to cover the geometry of the planar projections of the traditional three dimensional output primitives; we might call such a system a 2.5D graphics system. This is what we will attempt to do in the remainder of this paper.

Methodologically, what we will do is as follows. When implementing a three dimensional graphics system, one has to decide where, that is at which point of the output pipeline a given algorithm is to be performed. As an example, generating stroke precision characters means the generation of a number of small individual line segments; these line segments might be generated prior to the viewing transformation or, eventually, after it. If the line segments are calculated at the "beginning" of the pipeline, real three dimensional output may be generated without additional algorithmic problems (the stroke characters are interpreted just as a set of polylines); however, in this case, a very large number of points has to be generated, which may slow down the throughput of the system. On the other hand, generation of the appropriate line segments at a later step might lead to algorithmic problems when true three dimensional appearence is to be achieved. For our purposes, we will describe some of the internal mathematical algorithms of an imaginary three dimensional system which tries to solve this problem by postponing the actual output generation (of e.g. pattern parallelograms) as far as possible in the three dimensional output pipeline; basically, the aim is that everything should be performed in two dimensions, after viewing. Although this is not necessarily the optimal way to implement a general purpose system (see e.g. Herman [5]), this approach clearly generates an internal interface on the two dimensional level; an internal interface which might be used as a basis for the definition of a 2.5D system.

In the sequel, we will not deal with the problem of singularities eventually arising in the course of a projective transformation. What we have in mind is an application which uses "traditional" viewing

only, with the projection reference point being at a reasonable distance from the objects to be viewed. Clearly, there are applications where this assumption does not hold; however, handling these singularities properly would lead to a whole set of problems from the side of the application program; problems whose proper handling could make it worthwhile to use a general purpose three dimensional system (which has to handle these problems probably by "vocation").

2. Mathematical Preliminaries

It is to be expected that to achieve the goal cited above, we have to make use of some mathematical tools of projective geometry; in fact, this is the branch of mathematics which describes the behaviour of projections. We have to begin with some definitions.

Let A, B and C be three different collinear affine points on the plane. We define the *cross ratio* of the points A, B and C (denoted by (ABC)) as being the following (non-zero) real number:

$$(ABC) = \frac{\overline{AC}}{\overline{CB}} \tag{1}$$

where $\overline{AC}$ and $\overline{CB}$ mean the *directed* Eucledian distance of the two points.

If the point C trends to infinity, the limit of the corresponding aspect ratio (with the points A and B remaining fixed) will be -1; consequently, the definition in (1) may be extended to the case when the point C is a so called ideal point (i.e. the point of the line lying "at infinity"), namely:

$$(ABC) = -1$$

With the help of the cross ratio we may now define the *double ratio* of four collinear points. Let A, B, C and D be four different collinear points on the *projective* plane (one of the points might also be ideal). We define the double ratio of the four points (denoted by $(ABCD)$) as being the following real number:

$$(ABCD) = \frac{(ABC)}{(ABD)} \tag{2}$$

Clearly, when all four points are affine (i.e. none of them is ideal), the double ratio may also be expressed with the help of directed distances, namely:

$$(ABCD) = \frac{\overline{AC}\,\overline{DB}}{\overline{CB}\,\overline{AD}} \tag{3}$$

The double ratio has a number of remarkable properties. We will cite some of them, which are necessary for our purposes; the corresponding proofs may be found for example in Penna et al[6]. , in Fisher[7] in Coxeter[8] or in any other standard textbooks on projective geometry †.

First of all, the double ratio is a one-to-one mapping of the points of the line and the (non-zero) real numbers. Namely, the following is true:

> Given three different collinear points on the projective plane, denoted by A, B and C, if x is an arbitrary non-zero real number, then there exists one and only one point D on the line determined by A, B and C, for which the following equation holds:
>
> $$(ABCD) = x \tag{4}$$

The second property has a particular importance for us (and, in fact, it is one of the most important results in projective geometry). This property is as follows:

> The double ratio is projective invariant. This means that if A,B,C and D are four different collinear points of the plane (or space) and T is an arbitrary projective transformation,

† Care should be taken, however, when consulting the literature; in some cases, following different local traditions, the definitions of the double ratio may slightly differ from the ones given here (e.g. by an additive constant, the order of the directed distances in the formulae etc.).

then

$$(ABCD) = (T(A), T(B), T(C), T(D))$$

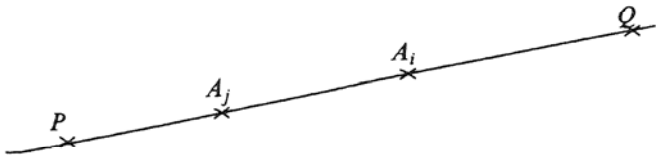

Figure 2

We will also need a more explicit version of equation (4). Namely, let us say that four different collinear affine points are given on a line like in Figure 2; let us denote them by P, Q, A_j and A_i (the reason for this notation will be clear a bit later). Let us also consider that we know from somewhere the value of (PQA_jA_i), which we will denote by α. Using these data, it is possible to express the value of the directed distance $\overline{PA_i}$ with the help of α and the directed distances involving P, Q and A_j. Namely:

$$\alpha = \frac{(PQA_j)}{(PQA_i)}$$

that is

$$\alpha = (PQA_j) \frac{1}{\frac{\overline{PA_i}}{\overline{A_iQ}}}$$

by substituting the value of $\overline{A_iQ}$:

$$\alpha = (PQA_j) \frac{\overline{PQ} - \overline{PA_i}}{\overline{PA_i}}$$

that is:

$$\alpha \, \overline{PA_i} = (PQA_j)\overline{PQ} - (PQA_j)\overline{PA_i}$$

which leads to the final expression:

$$\overline{PA_i} = \frac{(PQA_j)}{\alpha + (PQA_j)} \overline{PQ} \tag{5}$$

This simple equation is the clue to the further results; it shows that if we know the value of the double ratio from some other source, the point A_i may be determined on the line with the help of the other points. We have also to remark that the equation (5) is valid for all possible positions of A_i; it is not required that the point should belong to the line segment determined by P and Q (the substitution we used for $\overline{A_iQ}$ is valid for all possible cases).

With these simple mathematical tools in hand we may now describe the necessary mathematical algorithms to "implement" the following primitives of our imaginary 3D system: cell array, pattern style polygon filling and stroke characters. Our most detailed example will be that of the cell array; in fact, the very same ideas will be re-used for the two other primitives as well.

3. Cell Array

In case of three dimensional systems as well as in case of a traditional two dimensional system, the cell array primitive consists basically of a parallelogram (in a plane), subdivided regularly into a grid of smaller parallelograms, each of which is then assigned a given colour. Regularity means that an $m \times n$ subdivision is defined: the parallel edges are subdivided into m (respectively n) equal subsegments, and these internal points will generate the final grid.

This primitive is clearly one of those which lead to the problems described in the introduction. Indeed, applying a projective transformation against such a planar grid lying in space (that is projecting it onto another plane) the regularity will (eventually) be destroyed; the image of the parallelograms will not be parallelograms any more and the internal dividing points on the edges will not have a regular structure either. However, the internal subdividing points will not be absolutely random; as we have seen, having been generated by a projective transformation, their respective *double ratios* will remain unchanged.

Let us concentrate first on one side of the parallelogram only. Originally, that is prior to the projective transformation, the edge is determined by the two end-points P and Q. By the definition of the cell array primitive, a set of internal points, denoted by $A_1, A_2, \ldots, A_n$, is generated which subdivide regularly the line segment PQ. Applying the projection to these points, we arrive at the points $P', Q', A'_1, \ldots, A'_n$ respectively.

From equation (5) we have an explicit formula describing the directed distance $\overline{P'A'_i}$ for $i \neq 1$. This formula uses the value of the double ratio $(P'Q'A'_1A'_i)$ as well as some other values which are all independent of A'_i. However, as a result of the property concerning the double ratio cited above, this double ratio *has the same value as* (PQA_1A_j).

The important point is, that based on the regularity of the cell array prior to the transformation, this double ratio (that is the only value in (5) depending explicitly on A'_i!) may be calculated easily. Indeed, if there are n internal subsegments, then

$$(PQA_i) = \frac{\overline{PA_i}}{\overline{A_iQ}}$$

that is

$$(PQA_i) = \frac{\frac{i}{n}}{\frac{n-i}{n}} = \frac{i}{n-i}$$

which leads to the equation:

$$\alpha_i = (PQA_1A_i) = \frac{\frac{1}{n-1}}{\frac{i}{n-i}} = \frac{n-i}{i(n-1)} \tag{6}$$

Combining (5) and (6), we may derive the following formulae:

$$\overline{P'A'_i} = \frac{\beta_i}{n-i+\beta_i}\overline{P'Q'} \tag{7}$$

where

$$\beta_i = i(n-1)(P'Q'A'_1)$$

It is easily verifiable that if

$$(P'Q'A'_1) = \frac{1}{n-1}$$

(that is the point A'_1 corresponds to a regular subdivision point), all other points will automatically be regular, that is the formula (7) reduces to:

$$\overline{P'A'_i} = \frac{i}{n}\overline{P'Q'}$$

which was of course to be expected, but it serves as a check on the correctness of the formulae.

Using therefore equation (7), and supposing that the points P', Q' and A'_1 are given, all other A'_i, $i = 2, \ldots, n-1$ may be generated easily.

The formulae themselves are expensive to calculate in that they involve multiplication and division. However, we should not forget that the alternative would be to transform all these internal points with a general projective transformation, which is not a cheap calculation either ‡.

We are now in the position to finalise the cell array problem. Let us consider that our imaginary three dimensional system transforms the four corners of the parallelogram, as well as the first internal dividing point for each edge. Using the notation of Figure 3, we have now the points P', Q', R', S' as well as the points A'_1, B'_1, C'_1, D'_1. With these data in hand, the following steps should be performed.

- Generate all missing intersection points on the edges, that is the points A'_i, B'_i, C'_i and D'_i, using the formulae described in (7).
- Compute the intersection point of the line segments $A'_1C'_1$ and $B'_1D'_1$ (for the sake of simplicity and to make the necessary calculations faster, it also possible to transform the intersection point of A_1C_1 and B_1D_1 to generate this point).
- Again using (7), all internal points of the grid may be calculated, going through the line segments $B'_1D'_1$, $B'_2D'_2$ etc.

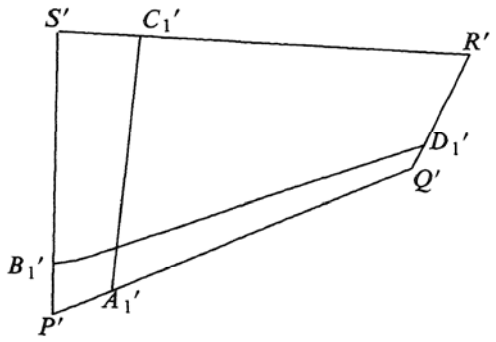

Figure 3

Clearly, the process is easily programmable and gives the required result; in spite of the relative complexity of the formulae in (7), the method is still faster than transforming all internal points of the net via a 3D projection.

It would seem to be more natural to use the intersection point of the edges $P'Q'$ and $S'R'$ (respectively $P'S'$ and $Q'R'$) as third points on the lines to determine the remaining internal subdivision points. This would reduce the number of points to be transformed. However, in a large number of cases, these intersection points tend to be relatively "far away" from the image of the cell array parallelogram; in fact, they might easily be ideal points. Although this would be still mathematically

‡ We have to remark that $\beta_{i+1} = \beta_i + (n-1)(P'Q'A'_1)$, in other words the β_i-s may be generated very quickly.

manageable, using these points would be computationally insecure; an overflow or underflow could easily occur in the course of the calculations. This is the reason why we have chosen the points listed above.

4. Pattern filling

The problem of fill area interior style pattern is very similar to cell array. The same regular structure is defined (which is usually called a pattern parallelogram in this case), which has to be duplicated linearly along the two edges to fill a given polygon.

In fact, the process of generating the images of the pattern parallelograms is exactly the same as in the case of cell array (using, therefore, the same defining data on the 2.5D level as for the cell array). As we have already mentioned, the formulae in (7) are not dependent on the position of A'_i; if the point is outside the original parallelogram, the same formulae may be used. The eventual singularity in the formulae (dividing by zero) corresponds to the singularity appearing during the projective transformation, which we have deliberately excluded. This means, that the points belonging to the "duplicated" pattern parallelograms may be calculated similarly, by extending the value of i to larger as well as to negative numbers.

One additional problem should be addressed here, however, and this is the lower and upper "limits" of the whole filling process. Namely, small monochrome four sided polygons may be generated, following the same procedure as for cell arrays; these polygons should be then clipped against the polygon to be filled and the result (if not empty), displayed. However, which is the first and which is the last four sided polygon to be investigated? Figure 4 shows the problem in a regular case: the edges of the monochrome parallelograms (which are, in this case, rectangles) generate a regular coordinate-system like grid on the plane; by calculating the minimal and maximal value of the polygon, the lower and upper limits of the monochrome subparallelograms may be easily established.

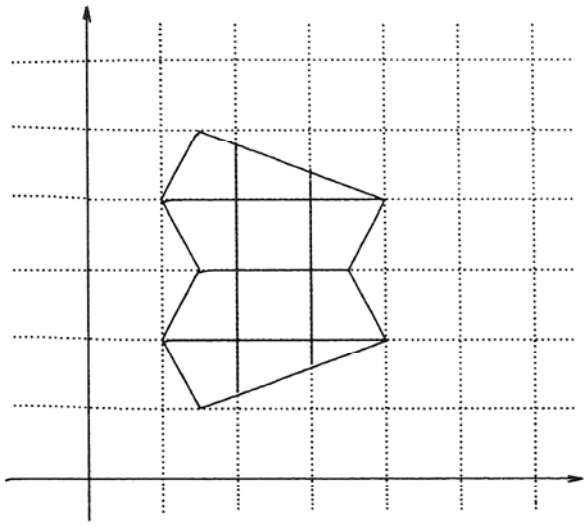

Figure 4

In the more general, 2.5D case, the situation is not really different. As may be seen in Figure 5, the edges of the monochrome subpolygons do generate a regular grid again; by going through these lines stepwise (using the formulae in (7) to determine the line equations) a clip may be performed against

the half planes to find the "minimal" and "maximal" vertex (in fact, this "grid" does correspond to a kind of coordinate system as well; the difference is that the corresponding coordinate system is a projective one and not a Cartesian one).

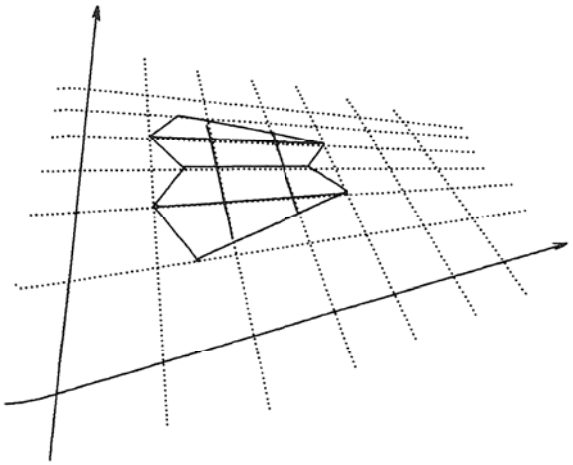

Figure 5

5. Stroke precision text

The last problem falling into the same category is the generation of stroke characters.

When implementing stroke characters in a general graphics system, the characters themselves are usually described internally on some kind of grid, or integer valued Cartesian coordinate system. The exact resolution of this grid is dependent on the environment and is usually hidden from the user of the graphics system. However, this number exists (it has typically a value around 100). Let us consider for the time being that this value is k and, for the sake of simplicity, we consider it to be an even number.

Furthermore, a text drawn in stroke precision is defined as a succession of planar character boxes (all boxes lying on the same plane). For the sake of simplicity again, we consider only the case when these boxes are of the same size. These two regular grids (the succession of character boxes and the character description grid) form a larger grid on the whole text extent parallelogram, which has a resolution of k in the "vertical" direction (as a result of the character description) and a resolution km in the "horizontal" direction, where m is the number of characters in the text. The grid generated for the string of Figure 1 is presented in Figure 6; to avoid confusion, however, a much coarser grid is shown, with a value of $k = 8$ instead of the order of 100.

Let us now define text for our 2.5D system as follows. The corner points of the whole text extentsion as well as the middle point of the edges belonging to the first character box should be transformed and handed over; out of these data the 2.5D system is able to reproduce the $m \times km$ grid which is necessary to generate the characters; once this grid is available, the (orthogonal) character description may be mapped onto this grid easily.

To reproduce the grid, almost the same formulae as in (7) may be used. The reason why the formulae are not applicable directly comes from the fact that the transformed points do not correspond to the *first* internal point. We should, therefore, generalise the formulae of (7) for the case when, say, an A_l is given instead of A_1 (as we have already remarked, the user should not be aware of the value of

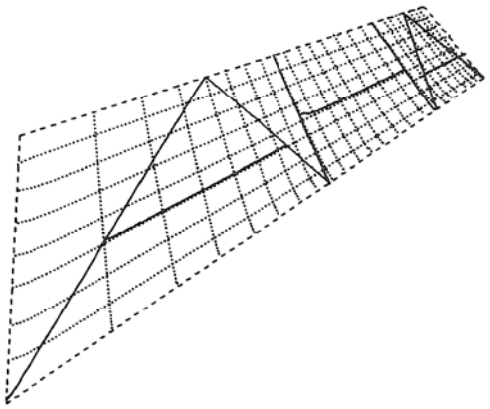

Figure 6

k, and therefore we cannot require the transformed value of the first point on the grid). However, this generalisation is not particularly complicated. In fact, only the value of α_i will be slightly different; indeed, instead of (6), we get

$$\alpha_i = \frac{\frac{l}{n-l}}{\frac{i}{n-i}} \tag{8}$$

which leads (instead of (7)) to

$$\overline{P'A'_i} = \frac{\beta_i}{l(k-i) + \beta_i}\overline{P'Q'} \tag{9}$$

where

$$\beta_i = i(k-l)(P'Q'A'_l) \tag{10}$$

Formulae (9) and (10) are the generalisation of (7). For the generation of stroke characters, we have to apply them independently for the "vertical" and "horizontal" directions. Namely, for the "vertical" case, $n = k$ and $l = k/2$, which leads to:

$$\overline{P'A'_i} = \frac{\beta_i}{k(k-i) + \beta_i}\overline{P'Q'}$$

where

$$\beta_i = ik(P'Q'A'_{k/2})$$

For "horizontal" direction, $n = km$ and $l = k/2$, which leads on its turn to:

$$\overline{P'A'_i} = \frac{\beta_i}{k(km-i) + \beta_i}\overline{P'Q'}$$

where

$$\beta_i = ik(m-1)(P'Q'A'_{k/2})$$

6. Ellipses

Although ellipses as output primitives do not appear in packages like GKS and PHIGS, they do appear in CGI [4] and they may also be introduced to a future revised version of GKS; that is why we deal with them here as well.

Unfortunately, ellipses are also distorted by a projective transformation. In fact, the following may happen (see Herman[5] for further details).

An ellipse is defined by a pair of *conjugate diameters* (or a pair of conjugate radii, as in CGI). This form of definition is preferred, because in this case the following is true:

> Let C be the centre of the ellipse, and P and Q be two respective endpoints of the conjugate radii. Then, the following vector equation will describe the points of the ellipse:

$$P(t) = \vec{CQ}\ sin(t) + \vec{CP}\ cos(t) + C \qquad (0 \leqslant t \leqslant 2\pi) \tag{11}$$

Equation (11) has the particular advantage of being affine invariant and, at the same time, gives an easy way to generate an approximating set of lines for the curve.

Unfortunately, this formula is not projective invariant. As a result of a projective transformation, the pair of conjugate diameters will be transformed into a pair of conjugate *chords* only; in other words, the image of C will not necessarily be the center of the curve.

It is however possible to reproduce the curve after the projective transformation as well. Our 2.5D system should get from the higher levels the images of both endpoints of the conjugate diameters: P' and R' for one line segment and Q' and S' for the other one (see Figure 7). The image of the center (C') may be calculated as an intersection point or, as in the case of cell array, it may also be given by transforming its original counterpart. The algorithm then the following form.

First of all we have to make the following remark. If D denotes the ideal point belonging to the line QS (that is prior to the transformation!) then:

$$(QSCD) = -1$$

This fact may be proved easily, using the fact that C is the middle point of QS. In other words, the image of D, denoted by D', may be generated using (5). We may consider that D' is not an ideal point; if both D' and the corresponding point on $P'R'$ were both ideal, the point C' would be the centre, and we could use directly formula (11).

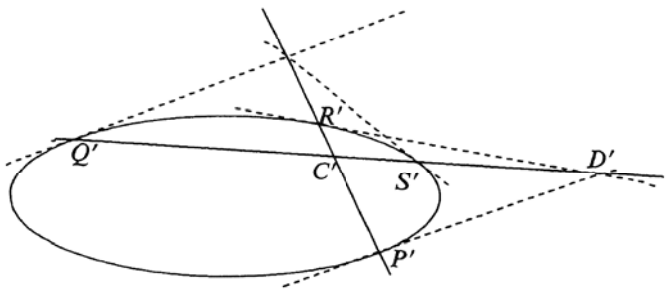

Figure 7

It is provable mathematically (the interested reader may find the proof in the usual textbooks of

projective geometry like the ones already cited) that the point D' will be the so-called *pole* of the line $P'R'$, which means in our case that the lines $D'P'$ and $D'R'$ will be tangents to the ellipse. In other words, by calculating the point D', we have "reduced" our problem to a so called conic-fitting problem (see e.g. Penna et al[6].), which is as follows

> Find the equation (i.e. the matrix) of the conic that passes through three given points (in our case P', S' and R') and is tangent to two given lines at two of these points (in our case $D'P'$ and $D'R'$).

The solution to this problem is roughly as follows (see again Penna et al[6]. for details).

If l_1 denotes the homogeneous representation of the line $D'P'$, l_2 denotes $D'R'$ and, finally, l_3 denotes the line $P'R'$ (see Figure 7), then we may define the following two matrices:

$$M_1 = \frac{1}{2}(l_1 l_2^T + l_2 l_1^T)$$

and

$$M_2 = l_3 l_3^T$$

By blending M_1 and M_2 we may define a set of matrices:

$$M_\lambda = \lambda M_1 + (1-\lambda)M_2 \tag{12}$$

(In fact, M_1 and M_2 are the matrices of two "degenerate" conics and (12) defines a set of curves between these two extremes.) The equation (12) may be then solved for S' to determine λ.

As a third step, making use of the matrix of the curve, we may define a parametrisation of the curve. It is a well known theorem of linear algebra, that if M is a symmetric matrix, then an additional Y matrix may be constructed, which has the following property: the matrix YMY^T may have non-zero values in the main diagonal only, and these values may be 1, -1 or 0 (this is called sometimes the transformation of the matrix against the main axes). Additionally, by constructing this matrix Y for our matrix M_λ, the following property also holds: the formula

$$P(t) = Y(\cos t, \sin t, 1)^T \quad (0 \leqslant t \leqslant 2\pi) \tag{13}$$

describes the points of the conic. In fact, this equation corresponds to (11).

By using these steps, what we have achieved is to reduce by one dimension the generation of the ellipse. Instead of using a spatial projective transformation, a planar one (namely $YM_\lambda Y^T$) is used for a set of simple points. However, the determination of this transformation is not so simple and, mainly, it is a *projective* (that is not necessarily affine) transformation, which means that not only the matrix-vector multiplication but also the projective division has to be performed. However, taking into account the fact that the faithful approximation of an ellipse requires a fairly large number of points, reducing the problem by one dimension may still lead to a valuable gain in generation speed.

7. Conclusions

Let us recapitulate the main points again. We have seen that by using some simple results of projective geometry we have a possibility to define what we have called a 2.5D graphics system. The outlines of the necessary extensions are as follows.

> *Cell array* is defined by determining the four corner vertices on the image of the cell array parallelograms, as well as the first internal subdivision points for each edge. To simplify the calculations, the image of the fourth vertex belonging to the lower-left-hand subparallelogram may also be defined.
>
> *Pattern parallelogram* may be defined similarly.
>
> *Stroke precision text* is defined by giving the image of the text extent parallelogram, as well as the images belonging to the mid-points on the edges of the first character box.
>
> *Ellipse* may be defined by giving the image of the conjugate diameter endpoints and (to simplify calculations) the image of the center of the ellipse.

We do not claim that these definitions are optimal; as usual, final functional specifications should be the result of a general consensus. However, it should be clear by now that there is a possibility to define such a 2.5D system, which may serve as a basis of some kind of "application profile"[9] definition put on the top of a "classical" 2D system.

8. Acknowledgements
The idea of an eventual extension of a 2D graphics system in the way I have presented here was raised by Jim Almond at a joint DIN/CGI-DIN/3D meeting held in Berlin in 1986, in which I took part as a member of the DIN/CGI delegation. However, only the problems were raised at that time but the group of experts attending did not really spend too much time on them. Although later I did try to find a solution, I just did not succeed. The real impulse came two years later, during the EG'88 conference in Nice, where I had long discussions with Gergely Krammer and I also had the opportunity to read a draft of his paper[10] which turned my attention to the possibility of using the notion of double ratio in computer graphics. I would like therefore to thank him for the inspiration I have got from him. I am also grateful to Paul ten Hagen, who had read the first draft of the paper and helped me with his comments to finalise it.

REFERENCES

1. ISO (1985). *Information processing systems - Computer graphics - Graphical Kernel System (GKS) functional description,* ISO 7942.
2. ISO (1988). *Information processing systems - Computer graphics - Graphical Kernel System for Three Dimensions (GKS-3D) functional description,* ISO 8805 .
3. ISO (1988). *Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) - Part 1, functional description,* ISO/IEC 9592-1 .
4. ISO (1988). *Information processing systems - Computer graphics - Interfacing techniques for dialogues with graphical devices (CGI) functional description,* ISO DP 9636/1-6.
5. I. HERMAN (1989). Projective Geometry and Computer Graphics, in *Advances in Computer Graphics IV*, ed. M. Grave & M. Roch, EurographicSeminar Series, Springer Verlag.
6. M.A. PENNA AND R.R. PATTERSON (1986). *Projective Geometry and Its Application to Computer Graphics*, Prentice-Hall.
7. G. FISHER (1985). *Analytische Geometrie*, Vieweg Studium, Grundkurs Mathematik, Vieweg & Sohn.
8. H.S.M. COXETER (1974). *Projective Geometry*, University of Toronto Press.
9. A.M. MUMFORD (1988). Application Profiles for Computer Graphics Standards - A Touch of Realism?, in *Eurographics'88 Conference Proceedings*, ed. D.A. Duce & P. Jancène, North-Holland.
10. G. KRAMMER (1989). On the Mathematics of the PHIGS Output Pipeline, *Computer Graphics Forum*, 8.