11 Perspective and Stereoscopic Projections

Perspective

We have seen that the orthographic projection has the property that parallel lines in three-dimensional space are projected into parallel lines on the view plane. Although very useful, such views do look odd! Our brains are used to the perspective phenomenon of three-dimensional space, and so they attempt to interpret orthographic figures as if they were perspective views. For example, the cubes of figures 9.1 and 10.1 look distorted.

So it is essential to produce a projection that displays perspective phenomena, that is, parallel lines should meet on the horizon – an object should appear smaller as it moves away from the observer. The drawing-board methods devised by artists over the centuries are of no value to us. Three-dimensional coordinate geometry and the concept of ACTUAL to OBSERVED positions, however, furnish us with a relatively straightforward technique.

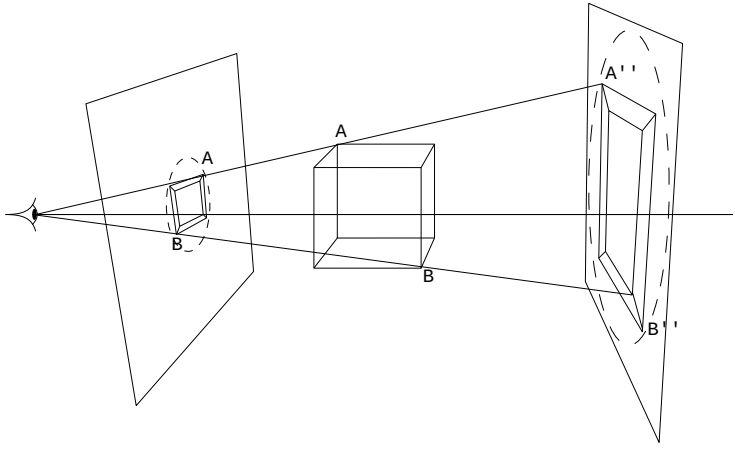

Figure 11.1

What is perspective vision?
To produce a perspective view we introduce a very simple definition of what we mean by vision. We imagine that every visible point in space is sending out a ray which enters the eye. Naturally the eye cannot see all of space, it is limited to the cone of rays that fall on the retina, the so-called *cone of vision*, which is outlined by the dashed lines of figure 11.1. The axis of this cone is called *straight-ahead* ray. We imagine that space has been transformed into the OBSERVED position with the eye at the origin and the straight-ahead ray identified with the positive *z*-axis.

We place the view plane (which we call the *perspective plane* in this special case) perpendicular to the axis of the cone of vision at a distance d from the eye. In order to form the perspective projection we mark the points of intersection of each ray with this plane. Since there are an infinity of such rays this appears to be an impossible task. Actually the problem is not that great because we need consider only the rays that emanate from the important points in the scene, that is the vertices at the ends of line segments or the corners of polygon; facets. The final view is formed by relating the projected points on the perspective plane in exactly the same way as they are related in three-dimensional space and then by identifying the view plane with the graphics screen.

Figure 11.1 shows a cube that is observed by an eye and projected on to two planes: the whole scene is also drawn in perspective! Two example rays are shown: the first from the eye to A, one of the near corners of the cube (relative to the eye), and the second to B, one of the far corners of the cube. The perspective projections of these points on to the near plane are A' and B', and on to the far plane A'' and B''. Note that the projections will have the same shape and orientation, but they will be of different sizes.

Calculation of the perspective projection of a point
We let the perspective plane be a distance *d* from the eye (variable PPD in later programs). Consider a point P = (x, y, z) in space that sends a ray into the eye. We must calculate the point where this line cuts the view plane (the $z = d$ plane) – suppose it is the point $P' \equiv (x', y', d)$. Let us first consider the value of y' by referring to figure 11.2 . By similar triangles we see that $y'/d = y/z$, that is $y' = y \times d/z$. Similarly $x' = x \times d/z$. Hence $P' \equiv (x \times d/z, y \times d/z, d)$. Since the view plane is identified with the *x*/*y* coordinate system of the graphics screen we can ignore the $z = d$ coordinate.

Example 11.1
Calculate the perspective projection of a cube that has eight vertices (0, 0, 4) + (±1, ±1, ±1) on the perspective plane $z = 4$, where the eye is origin and the straight-ahead ray is the positive *z*-axis.

The space is defined so that the scene is in the OBSERVED position. We can calculate the projections of the eight vertices by using the above method. For example (1, 1, 3) is projected to $(1 \times 4/3, 1 \times 4/3, 4) = (4/3, 4/3, 4) \rightarrow (4/3, 4/3)$ on the screen. So we get the eight projections:

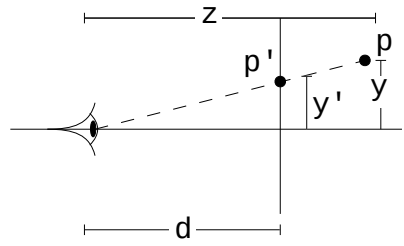

Figure 11.2

$(1, 1, 3) \rightarrow (4/3, 4/3)$ $(1, -1, 3) \rightarrow (4/3, -4/3)$

$(-1, 1, 3) \rightarrow (-4/3, 4/3)$ $(-1, -1, 3) \rightarrow (-4/3, -4/3)$

$(1, 1, 5) \rightarrow (4/5, 4/5)$ $(1, -1, 5) \rightarrow (4/5, -4/5)$

$(-1, 1, 5) \rightarrow (-4/5, 4/5)$ $(-1, -1, 5) \rightarrow (-4/5, -4/5)$

and the resulting diagram is shown in figure 11.3a

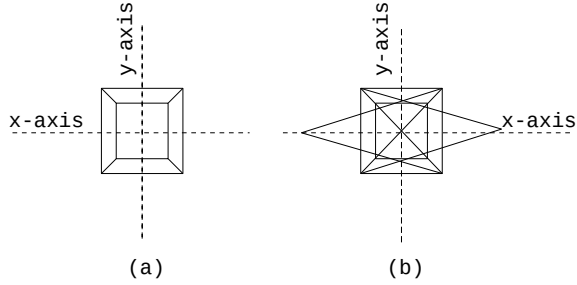

Figure 11.3

Properties of the perspective transformation

(1) The perspective transformation of a straight line (Γ_3, say) is a straight line (Γ_2 say). This is obvious because the origin (the eye) and the line Γ_3 form a plane (Ω say) in three-dimensional space and all the rays emanating from points on Γ_3, lie in this plane. (If the line enters the eye, Ω degenerates into a line.) Naturally Ω cuts the perspective plane in a line Γ_2 (or degenerates to a point) and so the perspective projection of a point on the original line Γ_3 now lies on the new line Γ_2. It is important to realise that a line does not become curved on perspective projection.

(2) The perspective transformation of a facet (a closed sequence of coplanar line segments) is a facet in the perspective plane. If the facet is an aarea bounded by n coplanar line segments then the transformation of this facet is naturally an area in the $z = d$ plane that is bounded by the transforms of the n line segments. Again note that no curves are introduced in this projection: if they were, then the task of producing perspective pictures would be far more complicated.
(3) The projection of a convex facet is also convex. Suppose that facet F_3 is projected on to facet F_2. Since the projection of a closed facet is also closed and lines go into lines, then points inside F_3 are projected into points inside F_2. Suppose F_2 is not convex. Then there exist two points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ inside F_2 such that the line joining them goes outside this facet. Hence there is at least one point p on the line outside F_2. If $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are projections of points $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ from F_3 then $\boldsymbol{p}$ is the projection of some point $\boldsymbol{q}$ on the line joining $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$. Since F_3 is convex then $\boldsymbol{q}$ must be inside F_3 and thus $\boldsymbol{p}$ must be inside F_2 – a contradiction and our proposition is thus proved.
(4) All infinitely long parallel lines appear to meet at one point, their so-called *vanishing point*. If we take a general line (with base vector $\boldsymbol{p}$) from a set of parallel lines with direction vector $\boldsymbol{h}$

$$\boldsymbol{p} + \mu\boldsymbol{h} \equiv x_p, y_p, z_p) + \mu(x_h, y_h, z_h)$$

where $zh > 0$, then the perspective transform of a general point on this line is

$$\left(\frac{(xp + \mu x_h) \times d}{(zp + \mu z_h)}, \quad \frac{(yp + \mu y_h) \times d}{(zp + \mu z_h)} \right)$$

which can be rewritten as

$$\left(\frac{(x_h + x_p/\mu) \times d}{(z_h + z_p/\mu)}, \quad \frac{(y_n + y_p/\mu) \times d}{(z_n + z_p/m)} \right)$$

As We move along the line towards large z-coordinates, that is as $\mu \to \infty$ then the line moves towards its vanishing point, which is therefore given by $(d \times x_h/z_h, d \times y_h/z_h)$. This vanishing point is independent of $\boldsymbol{p}$, the base point of the line, and hence all lines parallel to the direction $\boldsymbol{h}$ have the same vanishing point. Of course the case $z_h < 0$ is ignored because the line would disappear outside the cone of vision as $\mu \to \infty$.
(5) The vanishing points of all lines in parallel planes are collinear. Suppose that the set of parallel planes has a common normal direction $\boldsymbol{n} \equiv (x_n, y_n, z_n)$. If a general line in one of these planes has direction $\boldsymbol{h} \equiv (x_h, y_h, z_h)$ then h is perpendicular to $\boldsymbol{n}$ (all lines in these planes are perpendicular to the normal to the plane $\boldsymbol{n}$). Thus $\boldsymbol{n} \bullet \boldsymbol{h} = 0$, which in coordinate form is

$$x_n \times x_h + y_n \times y_h + z_n \times z_h = 0$$

Dividing by z_h gives

$$x_n \times x_h/z_h + y_n \times y_h/z_h + z_n = 0$$

and so the vanishing point ($d \times x_h/z_h$, $d \times y_h/z_h$) lies on the straight line

$$x_n \times x + y_n \times + d \times z_n = 0$$

and the statement is proved.

Example 11.2

Find the vanishing points of the edges of the cube in example 11.1, and of the diagonals of its top and bottom planes.

We divide the twelve edges of the cube into three sets of four edges; each set is parallel to the *x*-axis, *y*-axis and *z*-axis respectively and so has directional vectors (1, 0, 0), (0, 1, 0) and (0, 0, 1). The first two sets have zero *z*-values, and so their extended edges disappear outside the cone ofvision and are ignored, whereas the third direction has the vanishing point ($4 \times 0/1$, $4 \times 0/1$) ≡ (0, 0) on the view plane. On the top and bottom faces the diagonals have directions (1, 0, 1), the major diagonal, and (–1, 0, 1), the minor diagonal. The major diagonal on the top plane is (–1, 1, 3) + μ (1, 0, 1) and so the vanishing point is ($4 \times -1/1$, $4 \times 0/1$) ≡ (4, 0). The minor diagonal on the top plane is (1, 1, 3) + μ(–1, 0, 1) and has the vanishing point ($4 \times -1/1$, $4 \times 0/1$) ≡ (–4, 0). By similar calculations we find that the vanishing points of the major and minor diagonals on the lower face are also (4, 0) and (–4, 0) respectively. The relevant edges are extended to their vanishing points in figure 11.3b. Note that all the lines mentioned lie in the two parallel planes (the top and bottom faces ofthe cube) and so the vanishing points should be collinear: they are because (4, 0), (0, 0) and (–4, 0) all lie on the *x*-axis. By a similar calculation we would find that the vanishing points of the diagonals of the side faces lie on a vertical line through the origin.

Exercise 11.1

Draw a perspective view of a tetrahedron with vertices (1, 1, 5), (1, –1, 3), (–1, 1, 3) and (–1, –1, 5). Find the vanishing points (inside the cone of vision) of lines that join pairs of mid-points of the edges of the tetrahedron.

Programming the perspective transformation

The main program for drawing a perspective view of any scene is the same as that for the orthographic view, namely listing 9.2. Again the overall scene is created by a call to a procedure ‘scene3’, which is similar to those discussed in chapter 9. We shall often need to calculate explicitly the ACTUAL to OBSERVED matrix, so that the eye is in the OBSERVED position at the origin and looking along the positive *z*-axis. This is achieved by procedure ‘look3’ given in chapter 9 (listing 9.1). Calls are made to construction procedures, each having a matrix R as parameter. Finally the figure must be drawn, inside the construction procedures or in a ‘drawit’ procedure.

Note that the only difference between the program that draws a perspective view and that of the orthographic view of chapter 9 is in the calculation of the

coordinates of the projected image on the view plane. Unlike the orthographic in the perspective projection the coordinates on the view plane cannot be identified with the *x*-value and the *y*-value of the point in the OBSERVED position. We need to store the perspective transformation of the vertices in the arrays XD and YD: the Ith vertex (X(1), Y(1), Z(1)) in the OBSERVED position is projected to (XD(I), YD(I)). The values in arrays XD and YD are given by

(XD(I) = X(I)*PPD/Z(I) and YD(I) = Y(I)*PPD/Z(I) for I= 1, 2, . . . , NOV

The value of PPD is set to 3*VERT in 'scene3' – the reason for this is given in the next section. The calculation of XD and YD can be ade in the construction procedure, or in the 'scene3' or 'drawit' procedures: it simply depends on the scene that is being considered.

Example 11.3

We draw a fixed scene (the two cubes that are described in example 9.2) in perspective from a variety of observation points, setting HORIZ = 4 and VERT = 3. Rerun the program with HORIZ = 8 and VERT = 6 (is there any difference?). The necessary 'scene3' procedure will be almost the same as listing 9.6 except that it calculates PPD. It has the one addition:

6040 NOV = 0: NOL = 0: PPD = 3*VERT

it places the group of cubes in their ACTUAL position by using the 'cube' procedure of listing 9.7, and then loops through a number of different OBSERVED positions. For each time through the loop we call 'look3' which requires (EX, EY, EZ) and (DX, DY, DZ) to calculate the ACTUAL to OBSERVED matrix. Then the perspective 'drawit' procedure (listing 11.1) is called. This uses the matrix to transform the vertices from their (stored) ACTUAL position to the OBSERVED position, and places the projected vertex coordinates in arrays XD and YD, according to the above equations. The procedure can then finally draw the edges of the cubes in perspective.

Figure 11.4 was drawn by using (EX, EY, EZ) ≡ (15, 10, 5) and (DX, DY, DZ) ≡ (0, 0, 0). Compare this with the orthographic view of the same scene given in figure 9.2.

Exercise 11.2

Draw various perspective views of a wire tetrahedron and a pyramid.

The choice of perspective plane

The only value required for the perspective transformation that we have not yet discussed is that of PPD, the distance of the perspective plane from the eye. We can see from figure 11.1 that different values of PPD produce pictures of different sizes. Which one do we choose? Is there a correct value?

If we consider the practical situation, we note that the observer is sitting

Listing 11.1

```
7000 REM drawit / perspective cube not stored
7010 DEF PROCdrawit
7020 LOCAL I%,XX,YY,ZZ,L1,L2 : CLG
7030 FOR I%=1 TO NOV
7039 REM move ACTUAL points to their OBSERVED position (XX,YY,ZZ)

7040 XX=R(1,1)*X(I%)+R(1,2)*Y(I%)+R(1,3)*Z(I%)+R(1,4)
7050 YY=R(2,1)*X(I%)+R(2,2)*Y(I%)+R(2,3)*Z(I%)+R(2,4)
7060 ZZ=R(3,1)*X(I%)+R(3,2)*Y(I%)+R(3,3)*Z(I%)+R(3,4)
7069 REM store perspective projections in arrays XD and YD
7070 XD(I%)=XX*PPD/ZZ
7080 YD(I%)=YY*PPD/ZZ
7090 NEXT I%
7099 REM draw the lines
7100 FOR I%=1 TO NOL
7110 L1=LIN(1,I%) : L2=LIN(2,I%)
7120 PROCmoveto(XD(L1),YD(L1))
7130 PROClineto(XD(L2),YD(L2))
7140 NEXT I%
7150 ENDPROC
```

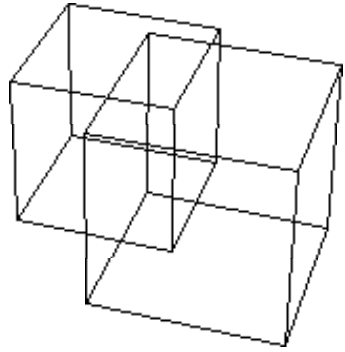

Figure 11.4

in front of a television screen and that the perspective view plane is identified with the plane of the television screen. Normally the observer is sitting at a distance that is about three times the height of the screen from the terminal. In the scale of our mapping from the real-world to the graphics area of pixels, this is a distance 3*VERT (the value we used above). If we choose PPD to be greater than this value it is as though we are creating a close-up, and if PPD is less than 3*VERT we get the smaller image of a long shot. You will have noticed that perspective pictures are independent ofthe screen size, that is the absolute values

of HORIZ and VERT are irrelevant, only their relative values matter, therefore in perspective pictures we shall always take HORIZ = 4 and VERT = 3: you can change the main pro@ram accordingly.

Example 11.4

We now draw a perspective hidden surface view of a stack of cubes (listing 11.2). Note how the vertical edges appear jagged. This is always the case in true perspective views because of the concept of vanishing points: compare this with the cubes drawn by listing 1.15, where we cheat! Figure 11.5 was drawn with mode 1 , HORIZ = 4, VERT = 3, (EX, EY, EZ) = (20, 10, 40) and (DX, DY, DZ) = (4, 1, 0). To demonstrate that the picture is independent of the screen size try the same picture with HORIZ = 40 and VERT = 30. Then try HORIZ = VERT = 1: now there is a difference.

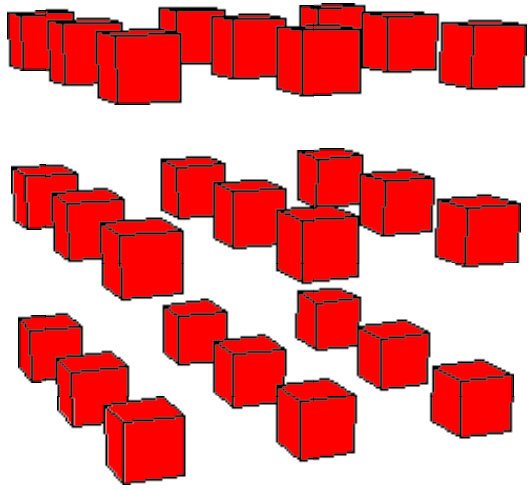

Figure 11.5

Clipping

Theoretically, objects may be positioned throughout space, even behind the eye, although we consider only points with positive *z*-coordinates in the OBSERVED position. Even so, some of these points go outside the cone of vision and become invisible. In fact, part of the cone of vision is outside the screen area (we can after all see the outside of the graphics area). We are left with a subset of the

Listing 11.2

```
6000 REM scene3 / stacking 27 cubes hidden surfaces
6010 DEF PROCscene3
6020 LOCAL I%,J%,XV%,YV%,ZV%
6030 DIM X(8),Y(8),Z(8),XD(8),YD(8)
6040 DIM A(4,4),B(4,4),R(4,4),Q(4,4)
6050 CLS : CLG : PPD=3*VERT
6059 REM Q : ACTUAL to OBSERVED matrix
6060 PROCidR3 : PROClook3
6070 FOR I%=1 TO 4 : FOR J%=1 TO 4
6080 Q(I%,J%)=R(I%,J%)
6090 NEXT J% : NEXT I%
6099 REM loop thru different placings of the cubes
6100 FOR ZV%=-16 TO -4 STEP 6
6110 FOR YV%=-6 TO 6 STEP 6
6120 FOR XV%=-6 TO 6 STEP 6
6129 REM move cube to ACTUAL position
6130 PROCidR3 : PROCtran3(XV%,YV%,ZV%) : PROCmult3
6139 REM then to OBSERVED position
6140 FOR I%=1 TO 4 : FOR J%=1 TO 4
6150 A(I%,J%)=Q(I%,J%)
6160 NEXT J% : NEXT I%
6169 REM draw visible faces of cube
6170 PROCmult3 : PROCcube
6180 NEXT XV% : NEXT YV% : NEXT ZV%
6190 ENDPROC
6500 REM cube / perspective hidden surfaces
6510 DEF PROCcube
6520 LOCAL I%,XX,YY,ZZ,F1,F2,F3,F4,DX1,DY1,DX2,DY2
6530 DATA 1,1,1,  1,1,-1,  1,-1,-1,  1,-1,1,
         -1,1,1,  -1,1,-1,  -1,-1,-1,  -1,-1,1
6540 DATA 1,2,3,4,  5,8,7,6,  1,5,6,2,  2,6,7,3,  3,7,8,4,  4,8,5,1
6550 RESTORE
6559 REM position vertices
6560 FOR I%=1 TO 8
6570 READ XX,YY,ZZ
6580 X(I%)=R(1,1)*XX+R(1,2)*YY+R(1,3)*ZZ+R(1,4)
6590 Y(I%)=R(2,1)*XX+R(2,2)*YY+R(2,3)*ZZ+R(2,4)
6600 Z(I%)=R(3,1)*XX+R(3,2)*YY+R(3,3)*ZZ+R(3,4)
6609 REM put perspective projection of Vertices in arrays XD and YD
6610 XD(I%)=X(I%)*PPD/Z(I%)
6620 YD(I%)=Y(I%)*PPD/Z(I%)
6630 NEXT I%
6639 REM loop thru facets
6640 FOR I%=1 TO 6
6650 READ F1,F2,F3,F4
6660 DX1=XD(F2)-XD(F1) : DY1=YD(F2)-YD(F1)
6670 DX2=XD(F3)-XD(F2) : DY2=YD(F3)-YD(F2)
6679 REM if visible draw facets
6680 IF DX1*DY2-DX2*DY1 < 0 THEN 6770
6690 PROCtriangle(XD(F2),YD(F2),XD(F1),YD(F1),XD(F3),YD(F3),1,-2)
6700 PROCtriangle(XD(F4),YD(F4),XD(F1),YD(F1),XD(F3),YD(F3),1,-2)
6710 GCOL 0,0
6720 PROCmoveto(XD(F1),YD(F1))
6730 PROClineto(XD(F2),YD(F2))
6740 PROClineto(XD(F3),YD(F3))
6750 PROClineto(XD(F4),YD(F4))
6760 PROClineto(XD(F1),YD(F1))
6770 NEXT I%
6780 ENDPROC
```

cone of vision – the *pyramid of vision*. Thus all points outside this pyramid, that is those whose perspective transformation take them off the screen, must be ignored. This is conveniently done for us on the BBC Model B; however you should note that this is not necessarily true on other computers. In fact we further limit scenes so that all vertices in the OBSERVED position will have positive *z*-values, that is all objects must lie in front of the eye (although not necessarily inside the cone of vision). This will avoid a peculiar property of our perspective projection, namely points that lie behind the eye appear on the screen. If you run the above program (the stack of cubes) and vary the values (EX, EY, EZ), for example (0.9, 0, 0), while all the other values stay t]qe same, then the eye may be in among the cubes and the picture will go haywire or even fail. Also see project XVII in chapter 16.

Example 11.3
Experiment with perspective views of all types of wire figures, such as bodies of revolution or regular solids. Consider cases where an object is drawn inside the construction routine, that is the values of XD and YD must now be calculated here and not in the 'drawit' routine. Change the program that drew the jet of figure 9.3 so that you get a perspective view, and note that the further the eye is from the plane the smaller it appears – a phenomenon that does not occur with the orthographic projection.

Exercise 11.4
Write a hidden line algorithm for a single convex body by using the ideas of listing 11.2.

Exercise 11.5
Write a program that draws a perspective view of a mathematical surface that is similar to the one given in chapter 10. The method will be exactly equivalent to listing 10.6, with the exception that you must work with the XD/YD values rather than the X/Y arrays.

These hidden surface and line algorithms are perfectly adequate for specially defined single objects. We extend these ideas in chapter 12 where we consider the more general case of a number of objects that are scattered arbitrarily about space. But first we look at stereoscopic projections which enable us to get true three-dimensional images from the BBC micro.

Stereoscopic Views

Perspective views are all very well but unfortunately (or fortunately!) we have two eyes. Each eye should have its own perspective, which will differ slightly from that of the other eye. This is the means by which we appreciate the three-

dimensional quality of our world. We use this concept to produce a stereoscopic view of space, namely we produce a perspective view for each eye. This leads to a problem. We cannot simply draw two such projections because the left eye will see not only the view created for it, but also that made for the right eye, and vice versa. To stop this confusion we must ensure that each eye sees its own view, but only its view. This is achieved by using a pair of stereoscopic spectacles: a pair of transparent plastic sheets, one red (left eye) and one cyan or light blue (right eye). In this way the left eye cannot see red lines because they appear to be the same colour as the white background (that is, both are tinted red) but cyan lines appear black. Similarly for the right eye which cannot see cyan lines, but red lines look black. So the computer must make two line drawings of a scene: one in cyan for the left eye, and one in red for the right eye. The brain will merge the two black images into one and the cyan and red background into white, to give a three-dimensional effect.

So we wish to devise a method of producing the stereoscopic projection of a general point P ≡ (x, y, z), that is two points PL ≡ (x_1, y_1) for the left eye and PR ≡ (x_2, y_2) for the right eye, in the coordinate system of the perspective view plane (see figure 11.6). We sensibly choose the same view plane for both eyes. We shall assume that the origin is between the eyes, that space is in the OBSERVED position, and that the direction of view for each eye (the straight-ahead ray) is parallel to the z-axis. The eyes have coordinates (–e, 0, 0), left, and (e, 0, 0), right: in the program that follows, e is given by variable ED, which is normally about 0.2*VERT. Again the perspective view plane is a distance d (variable PPD) from the origin. In order to find PL we move space by (e, 0, 0) so that P becomes ($x + e$, y, z) and the perspective transform of this point for the left eye is (($x + e$) × d/z –e, $y \times d/z$) which when we return space to its original position becomes ((x + e) × $d/z - e$, $y \times d/z$). Similarly, the right–εye transformation produces PR ≡ ((x – e) × $d/z + e$, $y \times d/z$). Listing 11.3 is a 'drawit' routine which draws a stereoscopic view of a wire object with NGV vertices and NOL lines stored in the usual way. Figure 11.7 shows a grey-scale picture of such a stereoscopic view of the two cubes of figure 9.2 by using 'lib1', '1ib3' and listings 9.6 (line 6040 adjusted for PPD) and 9.7. It has mode 1, HORIZ = 16, VERT = 12, (EX, EY, EZ) = (10, 20, 30) and (DX, DY, DZ) = (0, 0, 0).

For the best stereoscopic views it is best to make the view plane cut the object being viewed, that is make $\sqrt{(EX^2 + EY^2 + EZ^2)}$ = PPD (= 3*VERT). Therefore in the case of stereoscopic views we cannot keep HORIZ and VERT fixed, since for the best projections VERT (and hence HORIZ) depends on (EX, EY, EZ).

Exercise 11.6

Draw stereoscopic views of all the objects drawn previously, including the jet and bodies of revolution.

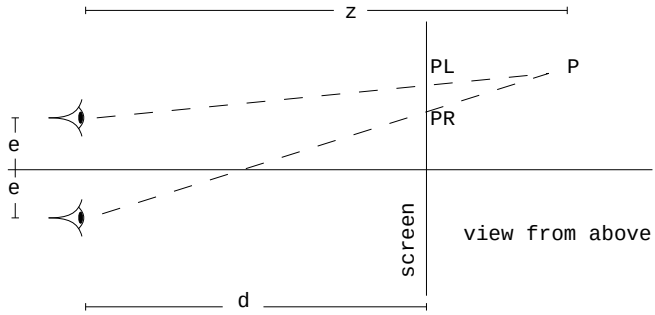

Figure 11.6

Listing 11.3

```
7000 REM drawit / stereoscopic
7010 DEF PROCdrawit
7020 LOCAL I%,L1,L2 : CLG
7028 REM use stereoscopic glasses with red over left eye
       and cyan over right eye
7029REM our background is white so the image for right eye must
       be drawn in colour 1 (red) and for left eye in colour 2
       redefined to be cyan, in order that lines are seen as
black
7030 VDU19,2,6,0,0,0
7039 REM first left eye then right
7040 ED=VERT*0.2 : GCOL 2,2
7050 FOR J%=1 TO 2
7060 FOR I%=1 TO NOV
7069 REM vertices in OBSERVED position
7070 XX=R(1,1)*X(I%)+R(1,2)*Y(I%)+R(1,3)*Z(I%)+R(1,4)
7080 YY=R(2,1)*X(I%)+R(2,2)*Y(I%)+R(2,3)*Z(I%)+R(2,4)
7090 ZZ=R(3,1)*X(I%)+R(3,2)*Y(I%)+R(3,3)*Z(I%)+R(3,4)
7099 REM stereoscopic projection
7100 XD(I%)=PPD*(XX+ED)/ZZ-ED
7110 YD(I%)=PPD*YY/ZZ
7120 NEXT I%
7129 REM draw object
7130 FOR I%=1 TO NOL
7140 L1=LIN(1,I%) : L2=LIN(2,I%)
7150 PROCmoveto(XD(L1),YD(L1))
7160 PROClineto(XD(L2),YD(L2))
7170 NEXT I%
7179 REM now use right eye
7180 ED=-ED : GCOL 2,1
7190 NEXT J%
7200 ENDPROC
```

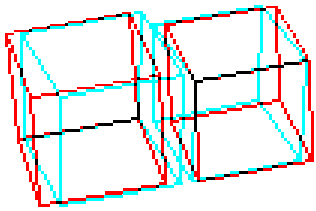

Figure 11.7

Exercise 11.7
Produce stereoscopic hidden line pictures of convex bodies. Now you must not colour in the facets, just draw the visible edges of the object, once in cyan for the left eye, and once in red for the right eye.

Complete Programs

I ‘lib1’, ‘lib3’ and listings 9.6 (‘scene3’), 9.7 (‘cube’) and 11.1 (‘drawit’). The ‘scene3’ routine must be adjusted for perspective thus:

6040 NOV=0: NGL=0: PPD=3*VERT

Data required: mode, HORIZ, VERT, and repeated values for (EX, EY, EZ) and (DX, DY, DZ). Try 1, 4, 3 (5, 15, 10) and (0, 0, 0); (1, 2, 20) and (0, 0, 1).

II ‘lib1’, ‘lib3’ and listings 11.2 (‘scene3’ etc. for the stack of cubes). Data required: mode, HORIZ, VERT, (EX, EY, EZ) and (DX, DY, DZ). Try 1, 4, 3, (20, 30, 40) and (0, 0, –6).

III ‘lib1’, ‘lib3’ and listings 9.6 (‘scene3’), 9.7 (‘cube’) and 11.3 (‘drawit' : stereoscopic). Adjust line 6040 as in above. Data required: mode, HORIZ, VERT, (EX, EY, EZ) and (DX, DY, DZ). Try 1, 16, 12, (10, 20, 30) and (0, 0, 0).