12 A General-Purpose Hidden Surface and Hidden Line Algorithm

There are many different types of hidden line and/or surface algorithin. One variety has a rectangular array that represents the totality of pixels on the screen. We imagine rays of light that leave the eye through each of the pixel on the screen. These rays naturally pass through objects in our scene and we can note the coordinates of these points of intersection. The array will hold the 'z-coordinate' (initially infinity) of the nearest point of intersection. So we build build up a picture by adding new objects, finding where the rays cut the object, and changing the array values (and the pixel colour on the screen) wherenever the latest point of intersection is nearer the eye than the corresponding value stored in the array. This technique (ray tracing) is very useful if we wish to shade-in areas in subtly differing tones of a given colour. It does, however, have enormous storage requirements and needs a very powerful computer, well beyond the capacity of microcomputers. Because we must work with only four colours and have limited storage we give another general algorithm which works on the 'back to front' principle mentioned earlier.

As in previous chapters, we assume that objects are set up by the 'scene3' procedure; however we now insist that the NOV vertices in the scene are stored in the X, Y, and Z arrays. Their perspective projections on to the view plane are stored in arrays XD and YD. The NOF convex facets are stored as a list of vertex indices (a maximum of six) in array FACET, and the number of edges in any facet is placed in array SIZE. Non-convex facets can be constructed out of convex facets.

we assume that all objects are closed. They need not be convex but each must be closed and its surface must be composed of convex facets which are stored in anticlockwise orientation. Thus it is impossible to see the under-side of any facet; that is, when projected on to the view plane we see only facets that maintain their anticlockwise orientation. Strictly speaking, this means that we cannot draw planar objects. If these are required for a particular scene then we avoid the problem by storing each facet of a planar object twice – once clockwise and once anticlockwise – so whatatever the position of the eye, on perspective projection we shall see one and only one occurrence of the facet. This restriction was imposed to speed up the hidden surface algorithm. This is very necessary because we are now approaching the limits of the processing power of the BBC micro.

Nevertheless, we think it is important to study general hidden line/surface algorithtns for educational reasons. It is essential for anyone with more than a passing interest in computer graphics to understand the problems implicit in drawing views of three-dimensional objects with the hidden lines/surfaces suppressed. The procedure given in listings 12.1 and 12.2 is such a hidden surface algorithm, which can be transferred to larger machines where it will run with ease. If you get the opportunity to use more powerful computers it will be very instructive to run our programs on them.

In order to produce a hidden surface picture of a scene that is stored in the OBSERVED position, each facet on the objects in the scene must be compared with every other facet in order to discover whether their projections overlap on the view plane. Because of the abow restrictions we need compare only the visible facets, that is those that when projected keep their anticlockwise orientation. If they do overlap we then need to find which facet lies in front and which behind. Once this information is compiled we can work from the back of the scene to the front to get a correct hidden surface picture. We do have another limitation: we assume that it is impossible for a facet to be simultaneously in front of and behind another facet (that is facets do not intersect one another except at their edges) and we cannot have situations where facet A is in front of (>) facet B > facet C etc. > facet A.

Our algorithm for discovering whether two facets (I% and J% from our database) do overlap when projected on to the view plane is given in procedure 'overlap' in listing 12.1 . The method depends on the concept of inside and outside developed in chapter 3 . We place the *x*-coordinate and *y*-coordinate of the vertices of facet I% in arrays XF(1, . .) and YF(1, . .) respectively. We then take one line from facet J% and cut off all parts of the facet that lie on the negative side of the line: the resulting polygon is placed in arrays XF(2, . .) and YF(2, . .). We then take the next line and compare it with these values and store the resulting polygon in XF(1 . .) and YF(1, . .) etc. After all the lines fiom facet J% have been used then we are left with the polygon that is common to both projected facets. If at any time this polygon becomes empty we know that the projected facets do not overlap and so we leave the procedure setting GVER = 0.

If the facets do overlap then OVER = 1 and we draw a line from the eye to intersect a point inside the common polygon on the view plane and find the intersections with facets I% and J%: the point we choose is the median of the first three points on the polygon. Comparing the *z*-coordinates of the respective intersections enables us to discover which of I% and J% is in FRONT and which is at the BACK.

Listing 12.1

```
7300 REM overlap
7310 DEF PROCoverlap(I%,J%)
7319 REM check if views of facets I% and J% overlap
7320 P1=1 : SI=SIZE(I%) : SJ=SIZE(J%)
```

```
7329 REM place projected vertices of facet I%
         in arrays XF(1,..) and YF(1,..)
7330 FOR K%=1 TO SI
7340 FI=FACET(K%,I%)
7350 XF(P1,K%)=XD(FI) : YF(P1,K%)=YD(FI)
7360 NEXT K%
7369 REM use each line of the J%th facet to slice off part of
         polygon stored in XF,YF. Line joins (X1,Y1) to (X2,Y2)
7370 VJ1=FACET(SJ,J%) : X1=XD(VJ1) : Y1=YD(VJ1)
7380 FOR K%=1 TO SJ
7390 VJ2=FACET(K%,J%) : X2=XD(VJ2) : Y2=YD(VJ2)
7399 REM line is CA.y+CB.x+CC=0
7400 CA=X2-X1 : CB=Y1-Y2 : CC=-X1*CB-Y1*CA
7409 REM go round SI vertices in XF, YF. If positive relative to
         line, then add to new XF,YF. If negative ignore. If an
         intersection then add this to to new XF,YF arrays.
7410 P2=3-P1 : XI1=XF(P1,SI) : YI1=YF(P1,SI)
7420 V1=CA*YI1+CB*XI1+CC : A1=ABS(V1) : PCN=0
     : IF A1<EPS THEN S1=0 ELSE S1=SGN(V1)
7430 FOR M%=1 TO SI
7440 XI2=XF(P1,M%) : YI2=YF(P1,M%)
7450 V2=CA*YI2+CB*XI2+CC : A2=ABS(V2)
     : IF A2<EPS THEN S2=0 ELSE S2=SGN(V2)
7460 IF S1>=0 THEN PCN=PCN+1 : XF(P2,PCN)=XI1 : YF(P2,PCN)=YI1
     : IF S1=0 THEN 7500
7470 IF S1=S2 OR S2=0 THEN 7500
7479 REM calculate intersection
7480 MU=A1 : UM=A2 : DENOM=A1+A2
7490 PCN=PCN+1 : XF(P2,PCN)=(UM*XI1+MU*XI2)/DENOM
     : YF(P2,PCN)=(UM*YI1+MU*YI2)/DENOM
7500 V1=V2 : S1=S2 : A1=A2 : XI1=XI2 : YI1=YI2
7510 NEXT M%
7519 REM facets do not overlap: OVER=0
7520 IF PCN<3 THEN OVER=0 : ENDPROC
7530 SI=PCN : P1=P2 : X1=X2 : Y1=Y2
7540 NEXT K%
7549 REM find (XMID,YMID) common to both projected facets
7550 XMID=(XF(P1,1)+XF(P1,2)+XF(P1,3))/3
     : YMID=(YF(P1,1)+YF(P1,2)+YF(P1,3))/3
7559 REM MU1 is the distance from the eye of equivalent point
         on the 3D plane containing facet I%
7560 V1=FACET(1,I%) : V2=FACET(2,I%) : V3=FACET(3,I%)
7570 DX1=X(V1)-X(V2) : DX3=X(V3)-X(V2)
7580 DY1=Y(V1)-Y(V2) : DY3=Y(V3)-Y(V2)
7590 DZ1=Z(V1)-Z(V2) : DZ3=Z(V3)-Z(V2)
7600 A=DY1*DZ3-DY3*DZ1 : B=DZ1*DX3-DZ3*DX1 : C=DX1*DY3-DX3*DY1
7610 D=A*X(V1)+B*Y(V1)+C*Z(V1)
7620 MU1=D/(A*XMID+B*YMID+C*PPD)
7629 REM MU2 is the distance from the eye of equivalent point
         on 3D plane containing facet J%
7630 V1=FACET(1,J%) : V2=FACET(2,J%) : V3=FACET(3,J%)
7640 DX1=X(V1)-X(V2) : DX3=X(V3)-X(V2)
7650 DY1=Y(V1)-Y(V2) : DY3=Y(V3)-Y(V2)
7660 DZ1=Z(V1)-Z(V2) : DZ3=Z(V3)-Z(V2)
7670 A=DY1*DZ3-DY3*DZ1 : B=DZ1*DX3-DZ3*DX1 : C=DX1*DY3-DX3*DY1
7680 D=A*X(V1)+B*Y(V1)+C*Z(V1)
7690 MU2=D/(A*XMID+B*YMID+C*PPD)
7699 REM if MU1>MU2 then FRONT=J% else FRONT=I% : OVER set to 1
7700 OVER=1 : IF MU1 > MU2 THEN FRONT=J% : BACK=I%
                           ELSE FRONT=I% : BACK=J%
7710 ENDPROC
```

The next step is to work out how to use this information to produce the final picture. This is achieved in listing 12.2, which contains the routines 'hidden', 'topsort', 'push' and 'pop' . The method is to compare each visible facet with every other and to produce a network of information about the relative positions of the facets (in front or behind). For each visible facet (I% say) the idea is to set up a linked list (LIS(I%)) that contains the indices of all facets that lie in front of it, and the array G(I%) will contain the number of facets that facet I% obscures. Array G is also used initially to denote if the facet is clockwise and invisible (G(I%) = −1), or anticlockwise and visible (G(I%) = 0).

We then create a stack on to which we initially 'push' any facet that does not obscure any other (that is, whose G value is zero). Then one at a time we 'pop' a facet off the stack and draw it on the screen. Once the facet is drawn, we go down the linked list for that facet and decrement the G counts for each facet in the list. If the G count for any facet becomes zero then the number of that facet is pushed on to the stack. Eventually this method, called topological sorting (the 'topsort' procedure), gives the correct order in which facets may be drawn to give the true back to front hidden surface view.

The linked lists and the stack are implemented by using a method known as a HEAP. HEAP% is an array of integers whose values are divided into two parts, the information section and a pointer. Because we are using relatively small integers, both information and pointer can be stored in one array location: multiply the information by 10 000 and add in the pointer. Initially the heap contains zero information and each location points to the next in the array. There is a variable called FREE which denotes the next available location in thc heap. Whenever store is required for a linked list or a stack (which is itself a linked list) the FREE location in HEAP is made available, and the value of FREE changed. A garbage collector is built into this system so that whenever a location is no longer needed it is reallocated to the FREE list. The stack can only become empty when all the facets have been drawn because of our restriction that facets cannot be shnultaneously in front of and behind one another. See Knuth (1973), or Horowitz and Sahni (1976) for a formal description of linked lists, stacks and topological sorting.

Because we cannot be sure of the size of the HEAP and of the XF and YF arrays, these must be input into the program. As a rough guide the lists rarely exceed 20, and the heap size is best set at about twice the number of facets. If you underestimate these array requirements then the program will fail. You can always run it again with larger values! The execution of such routines are necessarily slow when you consider the number of comparisons that have to be made, so we print out information about the facet being compared, the present value of FREE and the time taken in seconds. You will also find that, used in conjunction with our library of graphics and matrix procedures as well as large data-bases, the program has to be loaded into store with PAGE set to &1100. Even so there will not be enough room if the full MODE 1 screen is used. So

Listing 12.2

```
7000 REM hidden surface / general
7010 DEF PROChidden
7020 LOCAL I%,J%,K%,M%
7030 INPUT"Size of HEAP "HSIZ
7040 INPUT"Size of facet list "FLIST
7049 REM setup data structure arrays
7050 DIM G(NOF),LIS(NOF),HEAP%(HSIZ),XF(2,FLIST),YF(2,FLIST)
7060 EPS=0.000001 : FREE=1 : NVIS=0 : TSTO=TIME
7069 REM initialise the HEAP
7070 FOR I%=1 TO HSIZ
7080 HEAP%(I%)=I%+1
7090 NEXT I%
7099 REM orientate projected facet : G(I(%)=-1 clockwise
                                     G(I(%)=0   anticlockwise
7100 FOR I%=1 TO NOF
7110 I1=FACET(1,I%) : X1=XD(I1) : Y1=YD(I1)
7120 I2=FACET(2,I%) : X2=XD(I2) : Y2=YD(I2)
7130 I3=FACET(3,I%) : X3=XD(I3) : Y3=YD(I3)
7140 DX1=X2-X1 : DY1=Y2-Y1
7150 DX2=X3-X2 : DY2=Y3-Y2
7160 IF DX1*DY2-DX2*DY1 > 0
     THEN G(I%)=0 : NVIS=NVIS+1 : LIS(I%)=0 ELSE G(I%)=-1
7170 NEXT I%
7179 REM compare visible facets G(I%) now holds number of
         facets behind facet I%
7180 FOR I%=1 TO NOF-1
7190 IF G(I%)=-1 THEN 7250
7200 FOR J%=I%+1 TO NOF
7210 IF G(J%)=-1 THEN 7240
7219 REM if facets overlap i.e. facet FRONT in front of facet BACK
         then increment G(FRONT) and add FRONT to linked list for
         BACK. Adjust the HEAP
7220 PROCoverlap(I%,J%)
7230 IF OVER=1 THEN G(FRONT)=G(FRONT)+1 : NFREE=HEAP%(FREE)
     : HEAP%(FREE)=FRONT*10000+LIS(BACK) : LIS(BACK)=FREE : FREE=NFREE
7240 NEXT J%
7250 PRINT"FACET=";I%;"  FREE=";FREE;"  TIME=";(TIME-TSTO)/100;" SECONDS"
     : NEXT I%
7259REM draw the background and then topologically sort
        the network of linked lists.
7260 *SPOOL"PICCY"
7270 GCOL0,131 : CLG : PROCbackground : PROCtopsort
7280 *SPOOL
7290 ENDPROC

7800 REM topological sorting procedure
7810 DEF PROCtopsort
7819 REM create STACK. Push on it all facets that do lie in
         front of other facets
7820 LOCAL I%,J% : STACK=0
7830 FOR I%=1 TO NOF
7840 IF G(I%)=0 THEN PROCpush(I%)
7850 NEXT I%
7860 FOR I%=1 TO NVIS
7869 REM pop facet(=J%) from STACK
7870 J%=FNpop : IF J%=0 THEN PRINT"facet network has a cycle" : STOP
7879 REM draw the facet
7880 GCOL0,COL(J%) : F1=1 : FS=SIZE(J%)
7890 F=FACET(F1,J%) : MOVE FNX(XD(F)),FNY(YD(F))
7900 F=FACET(FS,J%) : MOVE FNX(XD(F)),FNY(YD(F))
```

```
7910 F1=F1+1 : FS=FS-1
7920 F=FACET(F1,J%) : PLOT85,FNX(XD(F)),FNY(YD(F)) : IF F1=FS THEN
7940
7930 F=FACET(FS,J%) : PLOT85,FNX(XD(F)),FNY(YD(F)) : IF FS-1=F1 THEN
7940 ELSE 7910
7939 REM draw the edges of the facet
7940 F=FACET(SIZE(J%),J%) : MOVE FNX(XD(F)),FNY(YD(F)) : GCOL 0,0
7950 FOR K%=1 TO SIZE(J%) : F=FACET(K%,J%)
     : DRAW FNX(XD(F)),FNY(YD(F)) : NEXT K%
7960 PT=LIS(J%)
7970 REPEAT : F2=HEAP%(PT) DIV 10000 : NEX=HEAP%(PT) MOD 10000
7980 G(F2)=G(F2)-1 : IF G(F2)=0 THEN PROCpush(F2)
7990 PT=NEX : UNTIL PT=0
8000 NEXT I%
8010 ENDPROC

8100 REM push I% onto STACK
8110 DEF PROCpush(I%)
8120 NF=FREE : FREE=HEAP%(FREE)
8130 HEAP%(NF)=I%*10000+STACK : STACK=NF
8140 ENDPROC

8150 REM pop value off STACK
8160 DEF FNpop
8170 NF=STACK : NUMB=HEAP%(NF) DIV 10000 : STACK=HEAP%(NF) MOD 10000
8180 HEAP%(NF)=FREE : FREE=NF
8190 =NUMB
```

what we do is to run the program in MODE 7 and SPOOL the picture on to backing store. When the program is complete we type

NEW

MODE 1

*EXEC PICCY

and it will be drawn on the screen. Note that we also include a call to a 'background' procedure. You can include a complex 'background' like that in listing 12.3, or a trivial procedure that simply clears the screen. We can turn a hidden surface procedure into a hidden line procedure by having a plain background and by drawing each facet in the same (background) colour.

Examle 12.1

We can now draw a hidden line, perspective view of the scene that we first saw in figure 9.2: one of the two cubes shown in figure 12.1. Note that the background is similar to one of the pictures produced in chapter 1 . The scene has HORIZ = 4, VERT = 3 and is viewed from (10, 5, 20) to (0, 0, 0).

The complete program uses 'lib1', 'lib3', 'overlap' (listing 12.1) and 'hidden' etc. (listing 12.2) together with the 'scene3', 'cube' and 'background' procedures given in listing 12.3. This last version of 'cube' means that we have considered all the array methods of constructing an object, that is, stored/not stored and lines/facets. We have deliberately used the cube over and over again in our diagram because it is such a simple object and because it is easy to understand its various constructions; therefore it does not complicate our discussion of the

Listing 12.3

```
4000 REM background
4010 DEF PROCbackground
4020 VDU23,1,0;0;0;0; : VDU 19,0,4,0,0,0
4030 PROCcircle(640,750,1000,0,1) : PROCcircle(640,750,150,2,2)
4040 PROCcircle(740,650,70,3,3) : PROCcircle(865,680,100,3,3)
     : PROCcircle(950,660,80,3,3)
4050 MOVE 740,580 : MOVE 740,720 : PLOT 85,950,580
4060 ENDPROC

4100 REM circle
4110 DEF PROCcircle(X,Y,R,C1,C2)
4120 MOVE X,Y
4130 F=1 : GCOL 0,C1
4140 FOR I=0 TO 2*PI STEP PI/50 : MOVE X,Y : PLOT
81,R*COS(I),R*SIN(I)
4150 F=1-F : IF F THEN GCOL0,C1 ELSE GCOL0,C2
4160 NEXT I
4170 ENDPROC

6000 REM scene3 / figure 12.1
6010 DEF PROCscene3
6020 LOCAL I%,J%
6030 DIM
X(16),Y(16),Z(16),XD(16),YD(16),FACET(4,12),SIZE(12),COL(12)
6040 DIM A(4,4),B(4,4),R(4,4),Q(4,4)
6050 NOV=0 : NOF=0 : PPD=3*VERT
6060 PROCidR3 : PROClook3 : PROCcube
6070 FOR I%=1 TO 4 : FOR J%=1 TO 4
6080 Q(I%,J%)=R(I%,J%) : NEXT J% : NEXT I%
6090 PROCidR3 : PROCtran3(3,1.5,2)
6100 PROCmult3
6110 FOR I%=1 TO 4 : FOR J%=1 TO 4
6120 A(I%,J%)=Q(I%,J%) : NEXT J% : NEXT I%
6130 PROCmult3 : PROCcube
6140 PROChidden
6150 ENDPROC

6500 REM cube / perspective, stored
6510 DEF PROCcube
6520 LOCAL I%
6530 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
6540DATA 1,1,1, 1,1,-1, 1,-1,-1, 1,-1,1,
         -1,1,1, -1,1,-1, -1,-1,-1, -1,-1,1
6550 RESTORE 6530
6560 FOR I%=1 TO 6
6570 READ F1,F2,F3,F4 : NOF=NOF+1
6580 FACET(1,NOF)=F1+NOV : FACET(2,NOF)=F2+NOV
6590 FACET(3,NOF)=F3+NOV : FACET(4,NOF)=F4+NOV
6600 SIZE(NOF)=4 : COL(NOF)=3
6610 NEXT I%
6620 FOR I%=1 TO 8
6630 READ XX,YY,ZZ : NOV=NOV+1
6640 X(NOV)=R(1,1)*XX+R(1,2)*YY+R(1,3)*ZZ+R(1,4)
6650 Y(NOV)=R(2,1)*XX+R(2,2)*YY+R(2,3)*ZZ+R(2,4)
6660 Z(NOV)=R(3,1)*XX+R(3,2)*YY+R(3,3)*ZZ+R(3,4)
6670 XD(NOV)=PPD*X(NOV)/Z(NOV)
6680 YD(NOV)=PPD*Y(NOV)/Z(NOV)
6690 NEXT I%
6700 ENDPROC
```

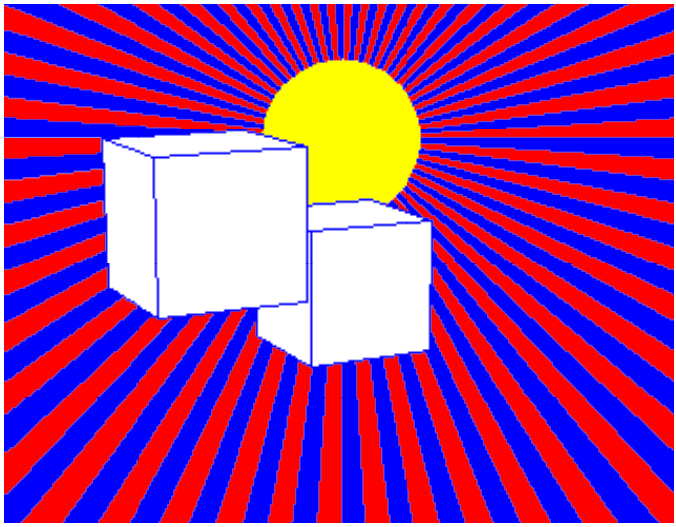

Figure 12.1

general principles of three-dimensional graphics. Now is the time to introduce complexity into our objects: provided that you understand the limitations of the algoritbms, you will find that the ideas we have discussed are equally valid.

Exercise 12.1

Construct hidden surface scenes that are composed of cubes, tetrahedra, pyramids, octahedra and icosahedra. Introduce new backgrounds and write your own procedures for an octahedron, icosahedron, rhombic dodecahedron etc. (see Coxeter, 1974).

Example 12.2

By now you will have realised that hidden surface algorithms are very slow prograins – we have to make a large number of comparisons. This means that we are rather limited in the scope of objects we can draw. Nevertheless it is very good practice, and if you have the opportunity to use larger machines you will see that the above algorithm will work on these also, but much faster. We give examples of two three-dimensional star-shaped objects in listing 12.4 (both require a parameter A which changes the elongation of the spikes) as well as a 'scene3' procedure. These two 'star' procedures are based on the tetrahedron and cube. Figure 12.2 was drawn with HORIZ = 4, VERT = 3 , and viewed from (35, 20, 25) towards (0, 0, 0).

Listing 12.4

```
6000 REM scene3 / two stars
6010 DEF PROCscene3
6020 LOCAL I%,J%
6030 DIM
X(22),Y(22),Z(22),XD(22),YD(22),FACET(3,36),SIZE(36),COL(36)
6040 DIM A(4,4),B(4,4),R(4,4),Q(4,4)
6050 NOV=0 : NOF=0 : PPD=3*VERT
6060 PROCidR3 : PROClook3 : A=6 : PROCstar1
6070 FOR I%=1 TO 4 : FOR J%=1 TO 4
6080 Q(I%,J%)=R(I%,J%) : NEXT J% : NEXT I%
6090 PROCidR3 : PROCtran3(5,5,5)
6100 PROCmult3
6110 FOR I%=1 TO 4 : FOR J%=1 TO 4
6120 A(I%,J%)=Q(I%,J%) : NEXT J% : NEXT I%
6130 PROCmult3 : A=4 : PROCstar2
6140 PROChidden
6150 ENDPROC

6500 REM star1 /based on a cube
6510 DEF PROCstar1
6520 LOCAL I%
6530 DATA 1,2,9, 2,3,9, 3,4,9, 4,1,9,   6,5,10, 5,8,10, 8,7,10,
          7,6,10, 2,1,11, 1,5,11, 5,6,11, 6,2,11, 4,3,12, 3,7,12,
          7,8,12, 8,4,12, 1,4,13, 4,8,13, 8,5,13, 5,1,13, 3,2,14,
          2,6,14, 6,7,14, 7,3,14
6540 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,
          A,0,0, -A,0,0, 0,A,0, 0,-A,0, 0,0,A, 0,0,-A
6550 RESTORE 6530
6560 FOR I%=1 TO 24
6570 READ F1,F2,F3 : NOF=NOF+1
6580 FACET(1,NOF)=F1+NOV : FACET(2,NOF)=F2+NOV
6590 FACET(3,NOF)=F3+NOV : SIZE(NOF)=3 : COL(NOF)=1
6600 NEXT I%
6610 FOR I%=1 TO 14
6620 READ XX,YY,ZZ : NOV=NOV+1
6630 X(NOV)=R(1,1)*XX+R(1,2)*YY+R(1,3)*ZZ+R(1,4)
6640 Y(NOV)=R(2,1)*XX+R(2,2)*YY+R(2,3)*ZZ+R(2,4)
6650 Z(NOV)=R(3,1)*XX+R(3,2)*YY+R(3,3)*ZZ+R(3,4)
6660 XD(NOV)=PPD*X(NOV)/Z(NOV)
6670 YD(NOV)=PPD*Y(NOV)/Z(NOV)
6680 NEXT I%
6690 ENDPROC
6700 REM star2 /based on a tetrahedron
6710 DEF PROCstar2
6720 LOCAL I%
6730 DATA 2,1,8, 3,2,8, 1,3,8, 1,2,7, 4,1,7, 2,4,7, 2,3,5,
          4,2,5, 3,4,5, 3,1,6, 4,3,6, 1,4,6
6740 DATA 1,1,1, 1,-1,-1, -1,1,-1, -1,-1,1,
          -A,-A,-A, -A,A,A, A,-A,A, A,A,-A
6750 RESTORE 6730
6760 FOR I%=1 TO 12
6770 READ F1,F2,F3 : NOF=NOF+1
6780 FACET(1,NOF)=F1+NOV : FACET(2,NOF)=F2+NOV
6790 FACET(3,NOF)=F3+NOV : SIZE(NOF)=3 : COL(NOF)=2
6800 NEXT I%
6810 FOR I%=1 TO 8
6820 READ XX,YY,ZZ : NOV=NOV+1
6830 X(NOV)=R(1,1)*XX+R(1,2)*YY+R(1,3)*ZZ+R(1,4)
```

```
6840 Y(NOV)=R(2,1)*XX+R(2,2)*YY+R(2,3)*ZZ+R(2,4)
6850 Z(NOV)=R(3,1)*XX+R(3,2)*YY+R(3,3)*ZZ+R(3,4)
6860 XD(NOV)=PPD*X(NOV)/Z(NOV)
6870 YD(NOV)=PPD*Y(NOV)/Z(NOV)
6880 NEXT I%
6890 ENDPROC
```

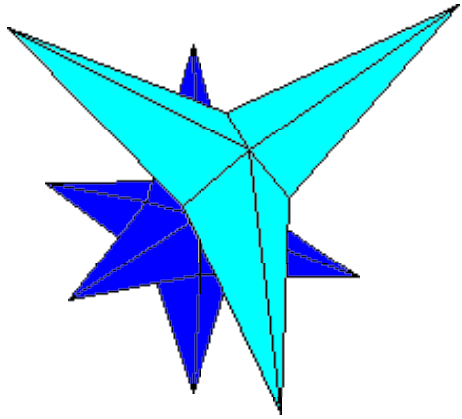

Figure 12.2

Exercise 12.2

The program in listing 10.1 checks that the order of the vertices of a triangular facet are anticlockwise. The program was devised for use with convex bodies that contain the origin. Extend it so that it can cope with the most general case. that is specify the position of the observer and the coordinates of a point inside the object (not necessarily the origin) so that this point and the observer lie on opposite sides of the infinite plane that contains the facet. Use this program to check the above star-shaped objects (in fact for these figures the origin could act as the inside point).

Then produce your own star-shaped objects that are based on an octahedron, cuboctahedron, icosahedron or dodecahedron. Always check the order of the vertices in your facets. You can produce stars that are based on very simple bodies of revolution, and we need not use only symmetrical objects! It is for these non-symmetrical shapes that you really need the extended version of listing 10.1. Provided that you stay within the restrictions mentioned, then listings 12.1 and 12.2 will draw any shape.

Example 12.3

We now give the procedures (listing 12.5) needed to set up and draw a much more complex picture (figure 12.3) which contains 120 facets (about 60 will be visible at any one time). It is viewed from (10, 20, 30) towards (0, 0, 0) and w take about a quarter ofan hour to draw. So be patient!

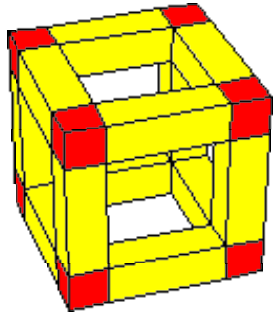

Figure 12.3

Now you have read (and understood) chapters 7 to 12 you will have found that we have reached the limits of three-dimensional graphics on the BBC Mode] B microcomputer. You must get the use of larger computers if you wish to go further in your study of this type of computer graphics. Then you must study the techniques of using data structures and extend the very simple applications given in this chapter. If you use certain structured languages, Pascal for example you will find that it is not essential to build up your own HEAP, such structures are iinplicit to the language. This should enable you to use complex data structures for storing useful information about scenes. For example, a complete scene can be regarded as a linked list of pointers, each of which refers to a linked list of information about the facets on a particular type of object. The facets themselves can be stored as lists ofvertices! A seemingly complex idea, but one that makes the use offixed-size arrays obsolete. Certain context relationships between facets may be stored implicitly in the lists. When you have grasped these ideas you can go on to the complicated graphics algorithms which include methods for animating, colouring, patching and shading. You will find the books by HoroWitz and Sahni (1976) and Knuth (1973) invaluable fbr the study of data structures. You should read Newman and Sproull (1979) and Foley and van Dam (1982) for the really complex three-dimensional graphics methods. In the next chapter we take a more advanced look at character graphics and introduce one method of producing animated three-dimensional drawings.

Listing 12.5

```
6000 REM scene3 / hollow cube
6010 DEF PROCscene3
6020 LOCAL I%,J%
6030 DIM X(160),Y(160),Z(160),XD(160),YD(160),
     FACET(4,120),SIZE(120),COL(120)
6040 DIM A(4,4),B(4,4),R(4,4),Q(4,4)
6050 DIM XQ(8),YQ(8),ZQ(8)
6060 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
6070 RESTORE 6060
6080 NOV=0 : NOF=0 : PPD=3*VERT
6090 FOR I%=1 TO 8
6100 READ XQ(I%),YQ(I%),ZQ(I%)
6110 NEXT I%
6120 FOR I%=1 TO 8
6130 PROCidR3 : PROCtran3(4.0*XQ(I%),4.0*YQ(I%),4.0*ZQ(I%)) :
     PROCmult3 : PROCrect(1,1,1,1)
6140 NEXT I%
6150 FOR I%=1 TO 4
6160 PROCidR3 : PROCtran3(0.0,4.0*YQ(I%),4.0*ZQ(I%))
     : PROCmult3 : PROCrect(3,1,1,2)
6170 PROCidR3 : PROCtran3(4.0*ZQ(I%),0.0,4.0*YQ(I%))
     : PROCmult3 : PROCrect(1,3,1,2)
6180 PROCidR3 : PROCtran3(4.0*YQ(I%),4.0*ZQ(I%),0.0)
     : PROCmult3 : PROCrect(1,1,3,2)
6190 NEXT I%
6200 PROCidR3 : PROClook3 : A=6 : PROCmult3
6210 FOR I%=1 TO NOV
6220 XX=X(I%) : YY=Y(I%) : ZZ=Z(I%)
6230 X(I%)=R(1,1)*XX+R(1,2)*YY+R(1,3)*ZZ+R(1,4)
6240 Y(I%)=R(2,1)*XX+R(2,2)*YY+R(2,3)*ZZ+R(2,4)
6250 Z(I%)=R(3,1)*XX+R(3,2)*YY+R(3,3)*ZZ+R(3,4)
6260 XD(I%)=PPD*X(I%)/Z(I%)
6270 YD(I%)=PPD*Y(I%)/Z(I%)
6280 NEXT I%
6290 PROChidden
6300 ENDPROC

6500 REM rectangular block
6510 DEF PROCrect(LH,HT,WH,IC)
6520 LOCAL I%
6530 DATA 1,2,3,4, 2,1,5,6, 6,5,8,7, 7,8,4,3, 2,6,7,3, 5,1,4,8
6540 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
6550 RESTORE 6530
6560 FOR I%=1 TO 6
6570 READ F1,F2,F3,F4 : NOF=NOF+1
6580 FACET(1,NOF)=F1+NOV : FACET(2,NOF)=F2+NOV
6590 FACET(3,NOF)=F
6600 SIZE(NOF)=4 : COL(NOF)=IC
6610 NEXT I%
6620 FOR I%=1 TO 8
6630 READ XX,YY,ZZ : NOV=NOV+1
6640 XX=XX*LH : YY=YY*HT : ZZ=ZZ*WH
6650 X(NOV)=R(1,1)*XX+R(1,2)*YY+R(1,3)*ZZ+R(1,4)
6660 Y(NOV)=R(2,1)*XX+R(2,2)*YY+R(2,3)*ZZ+R(2,4)
6670 Z(NOV)=R(3,1)*XX+R(3,2)*YY+R(3,3)*ZZ+R(3,4)
6680 NEXT I%
6690 ENDPROC
```

Complete Programs

All these programs should be loaded wi;h PAGE = &1100 and the REMs must be stripped away.

I ‘lib1’, ‘lib3’, listings 12.1 (‘overlap’), 12.2 (‘hidden’, ‘topsort’, ‘push’ and ‘pop’) and 12.3 (‘scene3’, ‘cube’, ‘background’). Data required: mode, HORIZ, VERT, (EX, EY, EZ), (DX, DY, DZ), size of HEAP and licet list. Try 7, 4, 3, (20, 5, 10), (0, 0, 0), 10 and 10. When the picture has been SPOOLed on to backing store as file PICCY you must then then type

```
NEW

MODE 1

*EXEC PICCY
```

II ‘lib1’, ‘lib3’, listings 12.1, 12.2, 12.4 (‘scene3’, ‘starl’ etc.) and the ‘background’ from 12.3. Data required: mode, HORIZ, VERT, (EX, EY, EZ) and (DX, DY, DZ), HEAP and facet counts. Try 7, 4, 3, (40, 30, 20), (0, 0, 0), 50, 20; and EXEC picture on to the screen from PICCY.

III ‘lib1’, ‘lib3’, listings 12.1, 12.2, 12.5 (‘scene3’, ‘box’ etc.) and the ‘background’ from 12.3. Data required: mode, HORIZ, VERT, (EX, EY, EZ) and (DX, DY, DZ), HEAP and facet counts. Try 7, 4, 3, (30, 20, 10), (0, 0, 0), 200, 20; and EXEC picture on to the screen from PICCY.