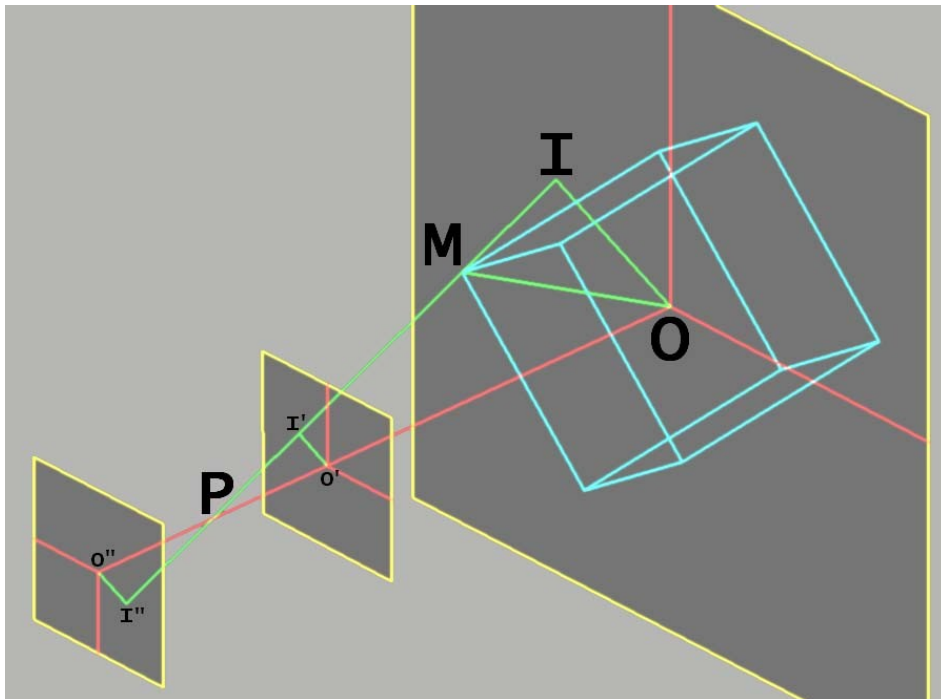


## An (old) new way to program perspective transforms with FORTH

In all the books about algorithms of perspective transforms, the classic method first considers three rotations to move the coordinate system of the object to superpose with the coordinate system of the point of view (or the camera), and then performs an affine transform.

To this end, one has to program matrix multiplications with sines and cosines, that needs floats. Which software libraries were not very fast at times (35 years ago) when the general consumer computers did not often have floating point coprocessors. Moreover, when some nerd insisted to program with languages which (like FORTH) did know only integers.

So, I (the nerd) did “furiously” search an alternative method, that would use only integers. This new method for old computers exists now, I invented it ! Look at the following figure :



The perspective projection from a point of view P will transform all points M of a 3D object into image points I on a plane orthogonal to the OP point of view vector. (or on other parallel planes, as you can see). Anyway, you get this simple vector computation formula with scalar and vector products :

$$\vec{OI} = \vec{OP} \wedge ( \vec{OP} \wedge \vec{MP} ) / ( \vec{OP} * \vec{MP} )$$

that may simply be expressed with integer operations on integers in object 3D coordinate reference :

$$\vec{V}_1 * \vec{V}_2 = x_1 * x_2 + y_1 * y_2 + z_1 * z_2$$

the scalar product (of integer 3D coordinates) gives a scalar, that is a simple (integer) number

$$\vec{V}_1 \wedge \vec{V}_2 = ( y_1 * z_2 - y_2 * z_1 , z_1 * x_2 - z_2 * x_1 , x_1 * y_2 - x_2 * y_1 )$$

the vector product gives a 3D vector (with 3D and finally 2D integer coordinates – scaled to integer screen pixels).

So, to program this in FORTH, I first had to program some primitives for 3D vectors with integer coordinates. Whereas the objects that I finally wanted to describe were kind of 3D “polyedres” with coordinates in the range of 16 bits signed integers.

To be noted that when multiplying 16 bit numbers you may get results that “overflow” up to 32 bits.

In FORTH, to overcome this issue, you have so called “scaled” integer multiplications ; words like  $\ast/$ , that take three parameters  $n1, n2, n3$  and yield to the result  $n1 \cdot n2 / n3$  where  $n1 \cdot n2$  is computed first on 32bits and if  $n3$  is well chosen, the following division gets back to 16 bits without overflow.

I coded the scalar and vector products similarly, with two 16 bits integer 3D vector operands and a 16 bits scale factor.

Here, the FORTH words for these primitive 3D vector operations :

```
.( 3D VECTORS )

: 3-VECTOR CREATE 6 ALLOT ;

: 3V@ 0 4 DO DUP I + @ SWAP -2 +LOOP DROP ;
: 3V! 6 0 DO DUP I + ROT SWAP ! 2 +LOOP DROP ;
: 3VDUP 2 PICK 2 PICK 2 PICK ;
: 3VOVER 5 PICK 5 PICK 5 PICK ;
: 3VSWAP 5 ROLL 5 ROLL 5 ROLL ;
: 3VROT 8 ROLL 8 ROLL 8 ROLL ;
: 3VDROP DROP DROP DROP ;
: 3V+ 3 ROLL + SWAP 3 ROLL + ROT 3 ROLL + SWAP ROT ;
: 3V- 3VSWAP 3 ROLL - SWAP 3 ROLL - ROT 3 ROLL - SWAP ROT ;
: 3V* >R 3 ROLL M* 2SWAP 4 ROLL M* ROT 5 ROLL M* D+ D+ R>
  M/MOD SWAP DROP ;
: 3V^ ( x,y,z,X,Y,Z,s --- [yZ-Yz]/s, [zX-Zx]/s, [xY-Xy]/s ) >R
  4 PICK OVER M* 3 PICK 6 PICK M* D- R@ M/
  4 ROLL 4 PICK M* 3 ROLL 7 PICK M* D- R@ M/
  5 ROLL 3 ROLL M* 4 ROLL 5 ROLL M* D- R> M/ ;
: MODULUS DUP M* ROT DUP M* D+ ROT DUP M* D+ SQRT SWAP DROP ;
```

Upon this first layer, a second set of words codes the perspective transform, based upon three 3D vectors and two scalar variables :

3-VECTOR P 3-VECTOR H 3-VECTOR V VARIABLE R VARIABLE S

P is the point of view (vector OP), H is the horizontal reference of the image plane, a vector of the same length as P, an V is the vertical reference of the image plane (also of the same length). S is the size of the 3D object, the diameter of a sphere containing it and R the length of vector P. P is given, S is attached to an object and the other parameters are computed with the word POINT-OF-VIEW :

```
: POINT-OF-VIEW 3VDUP P 3V! 3VDUP MODULUS R ! 3VDUP 3VDUP ROT
  ROT OR IF 2DROP 0 MODULUS >R 3VDUP DROP NEGATE R @ R@ */ SWAP
  R @ R@ */ 0 H 3V! NEGATE ROT OVER R@ */ ROT ROT R@ */ R> V
  3V! ELSE DROP ROT H 3V! NEGATE ROT ROT V 3V! THEN ;
```

Before going on, let me also show the word ROTATION that for animations, will rotate the point of view P or any other 3D vector (x,y,z --> xr,yr,zr) around a rotational vector O (ox,oy,oz).

```
: ROTATION ( x,y,z,ox,oy,oz --- xr,yr,zr ) 3VOVER 3VOVER 3VOVER
  MODULUS DUP >R 3V^ 3VOVER R@ 3V^ 2/ ROT 2/ ROT 2/ ROT 3VROT
```

```
3V+ 3VDUP 3VROT R> 3V^ 3V+ ;
```

The perspective transform itself will be computed through three words but first I have to define two variables CRD and CNX and a constant PIX. In an upper layer of code I will define a “defining” words MESH, which will contain a table of CRD 3D vectors and CNX connections between them and do the drawing. So :

```
VARIABLE CRD VARIABLE CNX 190 CONSTANT PIX ( should be >= 128 )
```

Depending on your display, the pixels of the QL are not squares but rectangles with various aspect ratio. One has to give PIX a value depending on that ; on QL Windows emulations, when the pixels are squares, the value is 128. On 4/3 TV sets, the value is about 190.

The first word, PERSPECTIVE will transform an OM vector to an OI vector according to the above formula :

```
: PERSPECTIVE ( xp,yp,zp,x,y,z --- xi,yi,zi ) 3VOVER 3VDUP 3VROT
3V- 3VOVER 3VOVER 3VOVER MODULUS DUP >R 3V* 3V^ R> 3V^ ;
```

The second word, PIXEL, given the number of a 3D vector in the connection table, will execute PERSPECTIVE on this vector and two scalar products with H and V followed by some scaling to address pixel coordinates in the QL screen display.

```
: PIXEL ( p --- h,v ) 6 * CRD @ + 3V@ P 3V@ 3VSWAP PERSPECTIVE
3VDUP V 3V@ R @ 3V* 128 S @ */ 128 + >R H 3V@ R @ 3V* PIX S @
*/ 256 + R> ;
```

The last word SEG will either draw a segment between the last point and this one if the connection parameter was positive, or go to this point “pen up”, if it was negative.

```
( SEG: c, hp, vp, ap+1 - goes to - c, hp+1, vp+1 )
: SEG DUP ABS PIXEL ROT 0> IF ADRAW ELSE 2SWAP 2DROP THEN ;
```

And now, the last layer will define words of type MESH and do the DRAWING :

```
: DRAWING ( mesh ) DUP CRD ! DUP 4 + @ S ! DUP @ 1+ 6 * OVER +
DUP ROT 2+ @ 2* + SWAP 0 0 2SWAP DO I @ SEG 2 +LOOP 3VDROP ;

: MESH ( s ) CREATE 0 , 0 , , 0 CRD ! 0 CNX ! DOES> DRAWING ;

: C ( x,y,z ) , , , CRD @ 1+ CRD ! ;
: X ( n ) DUP ABS CRD @ <= IF , CNX @ 1+ CNX ! THEN ;

: END-MESH LAST @ NAME> >BODY CRD @ OVER ! CNX @ SWAP 2+ ! ;
```

Here, using these four words, I defined two MESH, the first one named SQUARE and the second one named CUBE :

```
1000 MESH SQUARE 500 500 0 C -500 500 0 C
-500 -500 0 C 500 -500 0 C -1 X 2 X 3 X 4 X 1 X END-MESH

1000 MESH CUBE 500 500 -500 C -500 500 -500 C -500
-500 -500 C 500 -500 -500 C 500 500 500 C -500 500
500 C -500 -500 500 C 500 -500 500 C -1 X 2 X 3 X 4 X
1 X 5 X 6 X 7 X 8 X 5 X -2 X 6 X -3 X 7 X -4 X 8 X END-MESH
```

And with the following sentence, they will be seen as viewed down the Z object axis :

```
0 0 4000 POINT-OF-VIEW CLS RED SQUARE GREEN CUBE
```

This way, one may control if the PIX value is correct for the display : the squares should be square. So, what about animations ? I did program in the next words a rotation of the point of view AROUND the three object reference axis X Y and Z.

```
.( ANIMATIONS )
VARIABLE T 3-VECTOR 0 VARIABLE A_MESH

: RX T @ NEGATE DUP T ! 0 0 0 3V! ;
: RY T @ NEGATE DUP T ! 0 0 ROT ROT 0 3V! ;
: RZ T @ NEGATE DUP T ! 0 0 ROT 0 3V! ;

: STEP P 3V@ 0 3V@ ROTATION 3VDUP P 3V! ;
: figure red SQUARE green CUBE ;

: AROUND
  ' A_MESH !
  BEGIN CASE KEY 120 OF RX ENDOF 121 OF RY ENDOF 122 OF RZ ENDOF
    88 OF RX ENDOF 89 OF RY ENDOF 90 OF RZ ENDOF 27 OF ABORT ENDOF
    ENDCASE STEP POINT-OF-VIEW CLS A_MESH @ EXECUTE AGAIN ;

  2000 2500 1000 P 3V! 666 T ! RZ AROUND figure

.( AVION ) : XA 16 2 DO I X LOOP ;
5000 MESH AVION
  2600 -350 300 C 1300 -600 0 C 1000 -3500 300 C
    0 -3500 300 C 0 -600 0 C -2600 0 1000 C
-2600 -1800 1050 C -1600 -1800 1050 C -1300 0 1000 C
  400 -400 1000 C 900 -400 1000 C 1200 -550 600 C
  1200 550 600 C 900 400 1000 C 400 400 1000 C
-1600 1800 1050 C -2600 1800 1050 C -1600 0 2000 C
-2600 0 2000 C 0 600 0 C 0 3500 300 C
  1000 3500 300 C 1300 600 0 C 2600 350 300 C
  2700 -350 500 C 2700 350 500 C 600 -500 600 C
    600 500 600 C
-1 X XA 9 X 16 X 17 X 6 X 9 X 18 X 19 X 6 X 20 X 21 X 22 X 23 X
20 X 5 X 2 X 23 X 24 X -25 X 26 X 24 X 1 X 25 X 12 X 27 X 10 X
15 X 28 X 13 X 26 X -11 X 14 X END-MESH

: AIRPLANE GREEN AVION ;

6000 6000 4000 P 3V! T 500 ! RZ AROUND AIRPLANE
```

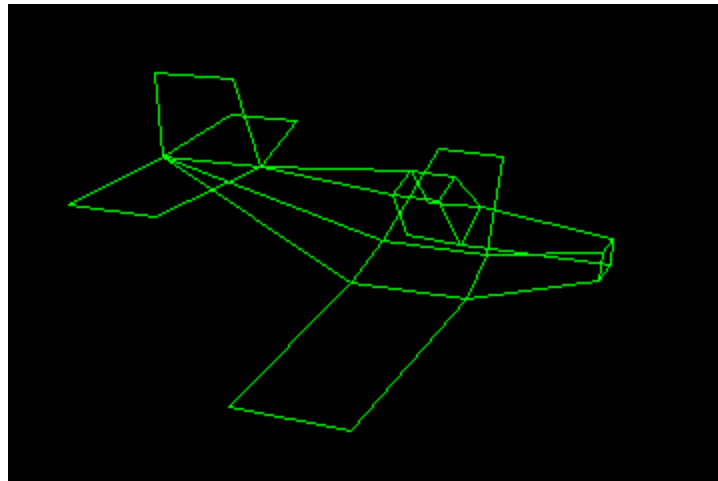
When the AROUND word is started, each time you hit the keyboard, the point of view is rotated and the OBJECT redrawn. If you hit the X or Y or Z key the rotation axis changes and if you hit twice the same the rotation reverses. If you hit the escape key the program stops.

Of course, on a plain old QL, the drawing is not very fast, but with Qemulator running at full speed, it is quasi instanteneous, and I think that I could program continuous animations, without any flicker like those shown as GIF images.

Actually for the show I cheated, I composed the GIF pictures image by image.

Now, for you to test this application, I will have to convert it for SuperForth and it will not be a

piece of cake, because it rests upon two low level words ( List #1 and #2 ) because I coded them in assembler ; and I cannot port them from the ComputerOne Forth machine to SuperForth. I guess I will have to use the floating point graphics of SuperForth that use the QDOS graphics.  
 For the moment, below : all the ComputerOne source file screens, and at the end, a surprise :  
 A cross-eyed stereoscopic GIF animation. For me, cross-eyed viewing is the most easy and pleasing.



( # 1 )

.( APLLOT ADRAW en assembleur 68000 P.KOPFF 14/11/1988 ) cr HEX

```
CREATE ADRAW HERE DUP 2- ! ( c,x1,y1,x2,y2 --- c,x2,y2 )
1C39 , 0002 , 8034 , E60E , 321E , 7401 , 926E , 0002 , 6C04 ,
4441 , 4442 , 3D42 , FFFE , 301E , 7401 , 906E , 0002 , 6C04 ,
4440 , 4442 , ED6A , 3D42 , FFFE , EC68 , 7600 , D640 , D641 ,
6772 , B041 , 6D38 , 3800 , 3A00 , 5345 , E348 , 3D40 , FFFA ,
E349 , 3D41 , FFF8 , 302E , 0002 , 3216 , D06E , FFFE , 986E ,
FFF8 , 6C08 , D26E , FFFC , D86E , FFFA , 3D40 , 0002 , 3C81 ,
617A , 51CD , FFDE , 6036 , 3801 , 3A01 , 5345 , E349 , 3D41 ,
FFFA , E348 , 3D40 , FFF8 , 302E , 0002 , 3216 , D26E , FFFC ,
986E , FFF8 , 6C08 , D06E , FFFE , D86E , FFFA , 3D40 , 0002 ,
3C81 , 6142 , 51CD , FFDE , 3C1C , 3E35 , 6800 , 4EF5 , 7800 ,
UNSMUDGE CREATE APLLOT HERE DUP 2- ! ( c,x,y --- c,x,y )
1C39 , 0002 , 8034 , E60E , 302E , 0002 , 3216 , 611A , 3C1C ,
3E35 , 6800 , 4EF5 , 7800 , UNSMUDGE -->
```

( # 2 )

.( APLLOT ADRAW en assembleur 68000 P.KOPFF 14/11/1988 suite ) cr

```
0000 , 0040 , 0080 , 00C0 , 8000 , 8040 , 8080 , 80C0 , 207C ,
0002 , 0000 , 4681 , 0280 , 0000 , 01FF , 0281 , 0000 , 00FF ,
EF89 , D1C1 , 2200 , E689 , E389 , D1C1 , 0280 , 0000 , 0007 ,
322E , 0004 , 0281 , 0000 , 0007 , 4A06 , 6606 , E289 , E589 ,
6006 , E288 , E388 , E389 , 7600 , 363A , 0038 , 4A43 , 6612 ,
7400 , 343C , 8080 , ECAA , 847C , 8080 , E0AA , 4682 , C550 ,
4A43 , 6A0A , 363B , 108E , E0AB , B750 , 4E75 , 363B , 1084 ,
E0AB , 8750 , 4E75 ,          DECIMAL      VARIABLE AGM      0 AGM !
```

```

: sqrt ( d --- n,p | d = p*p + n ) 2dup or if -32768 begin
  dup >r dup m* 2over d- r@ um/mod swap drop 2/ r> swap
  ?dup while - repeat 1- >r r@ r@ m* d- drop r> then ;

: M/ ( d,n --- p = d/n ) m/mod swap drop ;
( # 91 )

```

```

.( 3D VECTORS ) ( 1 LOAD )
: 3-VECTOR CREATE 6 ALLOT ;
: 3V@ 0 4 DO DUP I + @ SWAP -2 +LOOP DROP ;
: 3V! 6 0 DO DUP I + ROT SWAP ! 2 +LOOP DROP ;
: 3VDUP 2 PICK 2 PICK 2 PICK ;
: 3VOVER 5 PICK 5 PICK 5 PICK ;
: 3VSWAP 5 ROLL 5 ROLL 5 ROLL ;
: 3VROT 8 ROLL 8 ROLL 8 ROLL ;
: 3V+ 3 ROLL + SWAP 3 ROLL + ROT 3 ROLL + SWAP ROT ;
: 3V- 3VSWAP 3 ROLL - SWAP 3 ROLL - ROT 3 ROLL - SWAP ROT ;
: 3V* >R 3 ROLL M* 2SWAP 4 ROLL M* ROT 5 ROLL M* D+ D+ R>
  M/MOD SWAP DROP ; : 3VDROP DROP DROP DROP ;
: 3V^ ( x,y,z,X,Y,Z,s --- [yZ-Yz]/s,[zX-Zx]/s,[xY-Xy]/s ) >R
  4 PICK OVER M* 3 PICK 6 PICK M* D- R@ M/
  4 ROLL 4 PICK M* 3 ROLL 7 PICK M* D- R@ M/
  5 ROLL 3 ROLL M* 4 ROLL 5 ROLL M* D- R> M/ ;

```

( # 92 )

```

.( PERSPECTIVE ) VARIABLE CRD 128 CONSTANT PIX ( is >= 128 )
  3-VECTOR P 3-VECTOR H 3-VECTOR V VARIABLE R VARIABLE S
: MODULUS DUP M* ROT DUP M* D+ ROT DUP M* D+ SQRT SWAP DROP ;
: PERSPECTIVE ( xp,yp,zp,x,y,z --- xi,yi,zi ) 3VOVER 3VDUP 3VROT
  3V- 3VOVER 3VOVER 3VOVER MODULUS DUP >R 3V* 3V^ R> 3V^ ;
: ROTATION ( x,y,z,ox,oy,oz --- xr,yr,zr ) 3VOVER 3VOVER 3VOVER
  MODULUS DUP >R 3V^ 3VOVER R@ 3V^ 2/ ROT 2/ ROT 2/ ROT 3VROT
  3V+ 3VDUP 3VROT R> 3V^ 3V+ ;
: POINT-OF-VIEW 3VDUP P 3V! 3VDUP MODULUS R ! 3VDUP 3VDUP ROT
  ROT OR IF 2DROP 0 MODULUS >R 3VDUP DROP NEGATE R @ R@ */ SWAP
  R @ R@ */ 0 H 3V! NEGATE ROT OVER R@ */ ROT ROT R@ */ R> V
  3V! ELSE DROP ROT H 3V! NEGATE ROT ROT V 3V! THEN ;
: PIXEL ( p --- h,v ) 6 * CRD @ + 3V@ P 3V@ 3VSWAP PERSPECTIVE
  3VDUP V 3V@ R @ 3V* 128 S @ */ 128 + >R H 3V@ R @ 3V* PIX S @
  */ 256 + R> ; ( SEG: c,hp,vp,ap+1 - goes to - c,hp+1,vp+1 )
: SEG DUP ABS PIXEL ROT 0> IF ADRAW ELSE 2SWAP 2DROP THEN ; -->

```

( # 93 )

```

.( 3D MESH ) VARIABLE CNX
: DRAWING ( mesh ) DUP CRD ! DUP 4 + @ S ! DUP @ 1+ 6 * OVER +
  DUP ROT 2+ @ 2* + SWAP 0 0 2SWAP DO I @ SEG 2 +LOOP 3VDROP ;
: MESH ( s ) CREATE 0 , 0 , , 0 CRD ! 0 CNX ! DOES> DRAWING ;
: C ( x,y,z ) , , , 1 CRD +! ;
: X ( n ) DUP ABS CRD @ <= IF , 1 CNX +! THEN ;
: END-MESH LAST @ NAME> >BODY CRD @ OVER ! CNX @ SWAP 2+ ! ;

```

```

1000 MESH SQUARE 500 500 0 C -500 500 0 C
-500 -500 0 C 500 -500 0 C -1 X 2 X 3 X 4 X 1 X END-MESH
1000 MESH CUBE 500 500 -500 C -500 500 -500 C -500
-500 -500 C 500 -500 -500 C 500 500 500 C -500 500
500 C -500 -500 500 C 500 -500 500 C -1 X 2 X 3 X 4 X
1 X 5 X 6 X 7 X 8 X 5 X -2 X 6 X -3 X 7 X -4 X 8 X END-MESH

```

```

( 0 0 4000 POINT-OF-VIEW CLS RED SQUARE GREEN CUBE )      -->
( # 94 )

```

```

.( ANIMATIONS ) VARIABLE T 3-VECTOR 0 VARIABLE A_MESH
: RX T @ NEGATE DUP T ! 0 0 0 3V! ;
: RY T @ NEGATE DUP T ! 0 0 ROT ROT 0 3V! ;
: RZ T @ NEGATE DUP T ! 0 0 ROT 0 3V! ;
: STEP P 3V@ 0 3V@ ROTATION 3VDUP P 3V! ;

```

```

: figure red SQUARE green CUBE ;

```

```

: AROUND
' A_MESH !
BEGIN CASE KEY 120 OF RX ENDOF 121 OF RY ENDOF 122 OF RZ ENDOF
88 OF RX ENDOF 89 OF RY ENDOF 90 OF RZ ENDOF 27 OF ABORT ENDOF
ENDCASE STEP POINT-OF-VIEW CLS A_MESH @ EXECUTE AGAIN ;

```

```

6000 6000 4000 POINT-OF-VIEW 600 T ! RZ AROUND FIGURE

```

```

( # 95 )      -->

```

```

.( AVION ) : XA 16 2 DO I X LOOP ;

```

```

3000 MESH AVION
2600 -350 300 C 1300 -600 0 C 1000 -3500 300 C
0 -3500 300 C 0 -600 0 C -2600 0 1000 C
-2600 -1800 1050 C -1600 -1800 1050 C -1300 0 1000 C
400 -400 1000 C 900 -400 1000 C 1200 -550 600 C
1200 550 600 C 900 400 1000 C 400 400 1000 C
-1600 1800 1050 C -2600 1800 1050 C -1600 0 2000 C
-2600 0 2000 C 0 600 0 C 0 3500 300 C
1000 3500 300 C 1300 600 0 C 2600 350 300 C
2700 -350 500 C 2700 350 500 C 600 -500 600 C
600 500 600 C

```

```

-1 X XA 9 X 16 X 17 X 6 X 9 X 18 X 19 X 6 X 20 X 21 X 22 X 23 X
20 X 5 X 2 X 23 X 24 X -25 X 26 X 24 X 1 X 25 X 12 X 27 X 10 X
15 X 28 X 13 X 26 X -11 X 14 X END-MESH

```

```

: AIRPLANE GREEN AVION ;

```

```

6000 6000 3000 POINT-OF-VIEW 600 T ! RZ AROUND AIRPLANE

```

