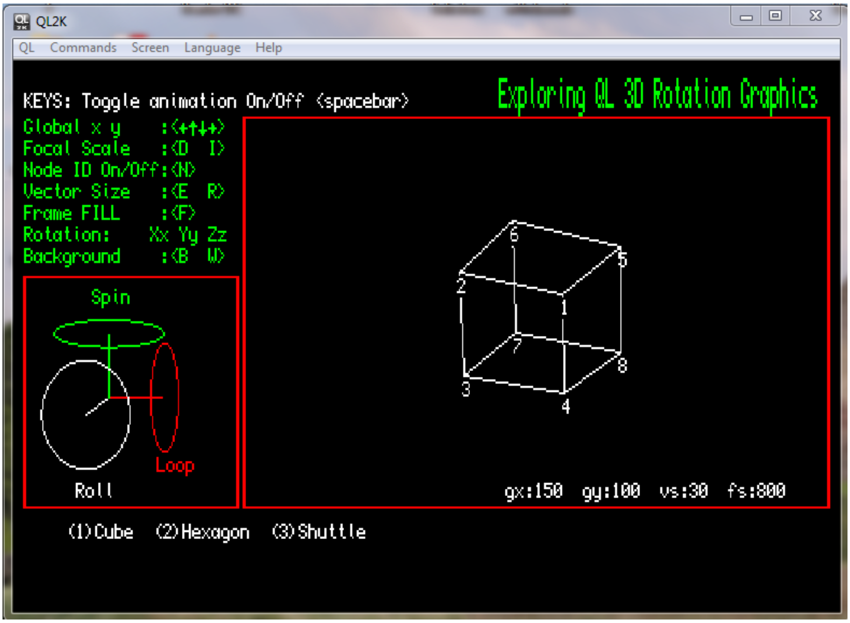
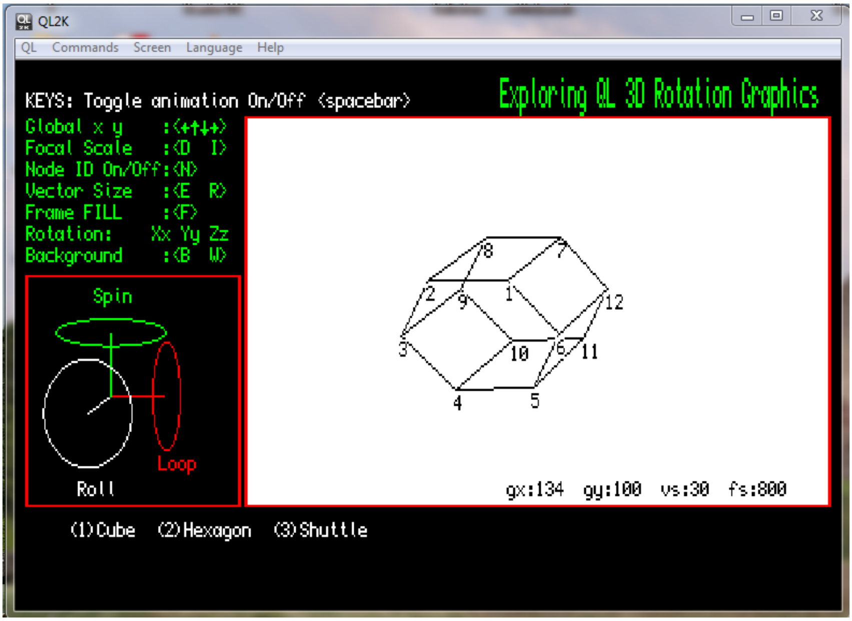
Sinclair QL Exploring 3D Rotation Graphics



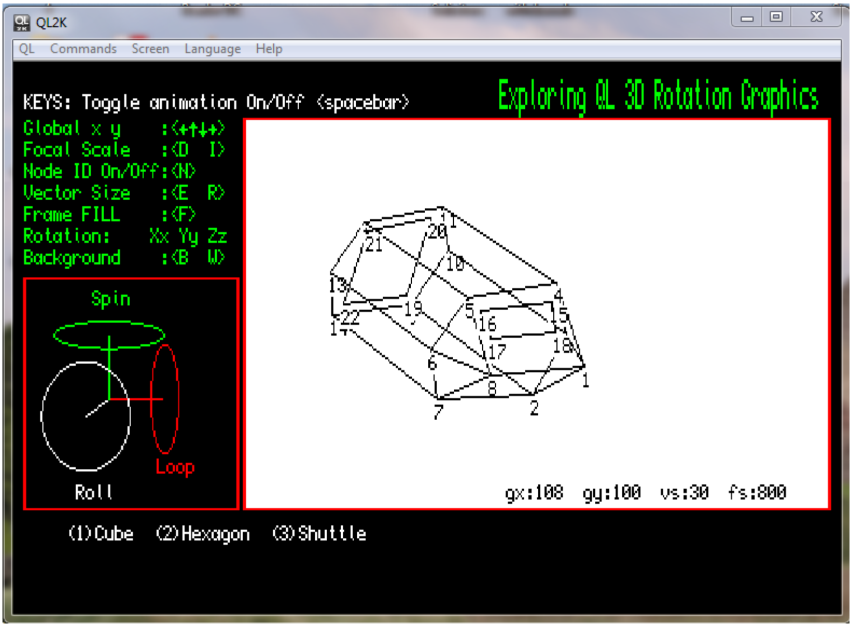
```
KEYS: Toggle animation On/Off <spacebar>          Exploring QL 3D Rotation Graphics
Global x y     :<←↑↓→>
Focal Scale    :<D  I>
Node ID On/Off:<N>
Vector Size    :<E  R>
Frame FILL     :<F>
Rotation:      Xx Yy Zz
Background     :<B  W>

                                         gx:150  gy:100  vs:30  fs:800

     (1)Cube   (2)Hexagon   (3)Shuttle
```

Sinclair QL Exploring 3D Rotation Graphics



Exploring QL 3D Rotation Graphics

```
KEYS: Toggle animation On/Off <spacebar>
Global x y     :<←↑↓→>
Focal Scale    :<D  I>
Node ID On/Off:<N>
Vector Size    :<E  R>
Frame FILL     :<F>
Rotation:      Xx Yy Zz
Background     :<B  W>

gx:134  gy:100  vs:30  fs:800

(1)Cube  (2)Hexagon  (3)Shuttle
```



Exploring QL 3D Rotation Graphics

```
KEYS: Toggle animation On/Off <spacebar>
Global x y     :<←↑↓→>
Focal Scale    :<D  I>
Node ID On/Off:<N>
Vector Size    :<E  R>
Frame FILL     :<F>
Rotation:      Xx Yy Zz
Background     :<B  W>

gx:108  gy:100  vs:30  fs:800

(1)Cube  (2)Hexagon  (3)Shuttle
```

QBITS on Animation

Magic Lanterns with flickering cards, later replaced with photography pictures all were used to create the illusion of movement. The first motion sequence photographed in real-time was created in 1878 by British photographer Eadweard Muybridge. W. K. L. Dickson **a**n Edison employee developed the Kinetoscope (a peep-hole motion picture viewer) introduced at the Chicago World Fair of 1893. The Lumiere brothers made the first public screening of ten short films in 1895 and as they say the motion picture industry was born.

In much the same way the availability of home computing in the 1980’s sparked myself and others interest in creating computer generated moving graphics, especially when it involved the manipulation of 3D images.

QBITS 3D Graphics

Unfortunately back in my early days of programming with QL SuperBASIC, such things as 3D Rotation Graphics was a little out of my league and probably still is. However I did jot down some notes amongst my future aspirations.

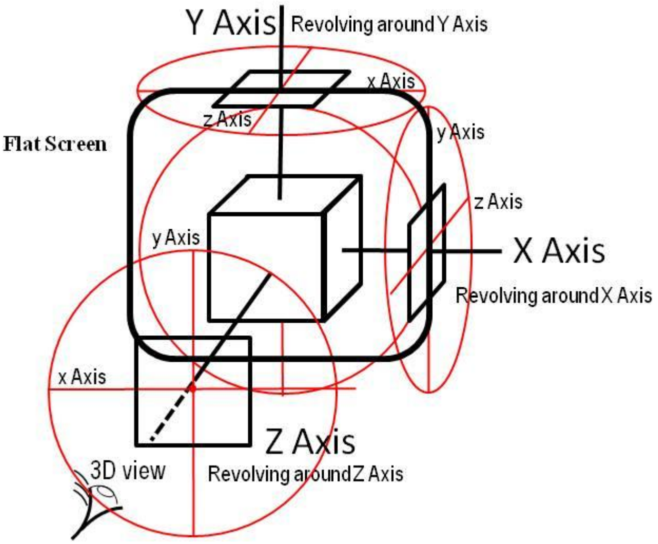
So having involved myself with QL SuperBASIC again I thought it was time to give 3D Rotation Graphics a bit of a spin (sorry for the pun). However what exactly was my goal. One was the basic code for revolving a simple object such as a wireframe Cube in various directions, moving it about the screen, altering its global position and being able to zoom in and out. Create the illusion of perspective perhaps, something to do with focal scale, but more about that later. As an afterthought a finishing touch would be filling in each enclosed area of my wire frame to create a solid object.

Depending on what source you refer to or your own background you might come across a few variation on the terms used for 3D movement. The most common are Roll, Pitch, Yaw associated with flying. However there are others, Rotate, Circulate Orbit, Spin, Loop. For my 3D Rotation Graphics I decided on Loop, Spin & Roll. All just happen to be four letter words, a little conformity in computer programming always a good thing.

Exploring 3D Rotation Graphics

So where to start... with a two dimensional object, its outline points of reference are depicted by xy coordinates. Moving position, alters what is displayed, but is achieved by changing the xy coordinates by an equal number of x points across the screen (left to right) and by the number of y points (up or down) the screen. When an object is moved to a new position, without changing its shape or size, this is a translation.

A three dimensional object requires a third coordinate, usually assigned as z. So first objective was to explore this z coordinate and its association with x and y. 3D rotation changes the orientation within each of its relative planes. The transformation alters the shape and size viewed. So what is involved in sorting out this change in coordinate values on a 2D screen? Just the sort of thing to send many a programmer, myself included, into a mental meltdown.



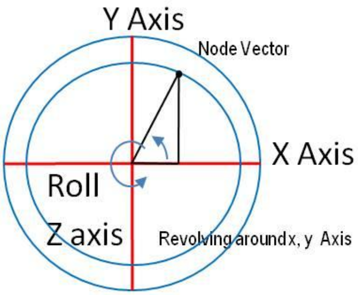
Imaginary Eye View

Looking at a flat screen it is easy enough to imaging the x, y coordinates. For three-dimensional space, we need to look at points that lie in front and behind the screen. To do this our imagine of a cube is half stuck out from the front of the screen surface, the other half lying behind. Looking face on to the screen, you see a square, when you stand over the screen and look straight down you also see a square (half poking out the front, half poking out the back). Looking directly from left or right of the screen, again you see a square half out the front and half out the back

xyz Coordinate Relationship
Understanding the relationship between xyz planes to identify location points of an Object is the key to 3D rotation. The three drawings shown here represent one Node and its position relative to the three planes of Roll, Spin, and Loop. I hope that the diagrams will go some way to revealing the illusion of 3D graphics when viewed on a 2D screen.

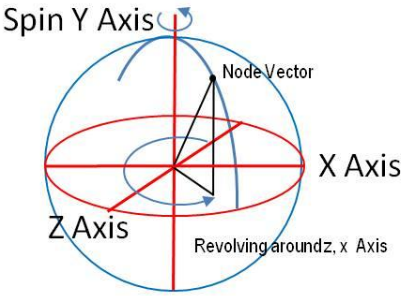
Roll
A Node rotating on the Z Axis is as if looking directly at the screen it moves around the four quadrants of the x y coordinates.

As shown its starting position is in the first quadrant (0 to 90 degrees).



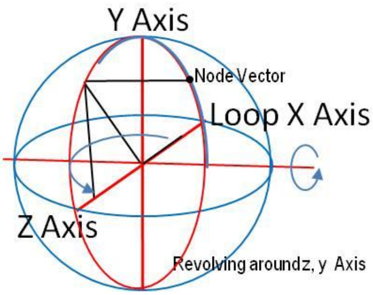
Spin
A Node rotating on the Y Axis is as if looking down from above. It moves around the four quadrants of the zx axis. This also shows its relationship.

The starting position is not perhaps what you were expecting, as it now lies in the fourth quadrant (270 to 360 degrees).



Loop
A Node rotating on the X Axis is as if looking at it from the side as it moves around the quadrants of the zy axis.

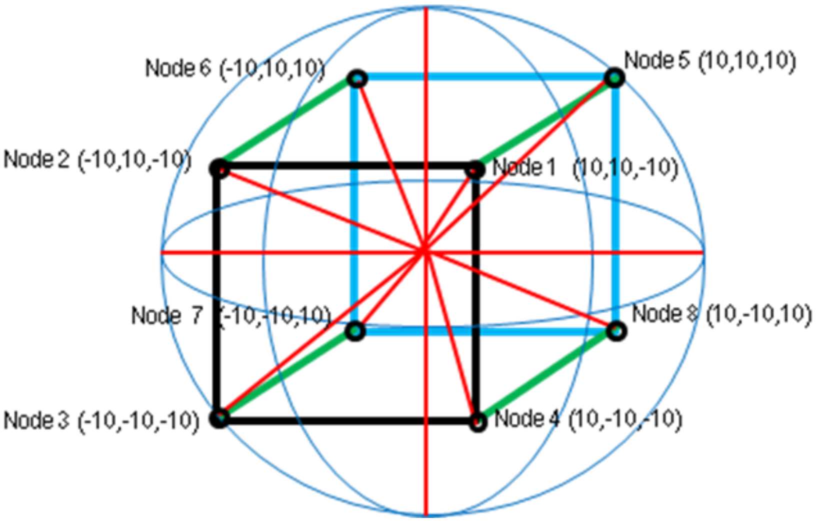
The vector position of the Node now appears to lie in the second quadrant (90 to 180 degrees).



On each plane if we refer to the centre of our object as 0,0 then revolving around it in the first quadrant 0-90 degrees we have +x,+y. In the second 90 to 180 degrees we have –x, +y. In the third 180 to 270 degrees -x, -y and finally in quadrant four 270 to 360 (and back to 0) degrees we have +x,-y.

Initialising Node xyz coordinates

Each point of reference that connects a 3D object be it a simple cube or multisided polyhedron I refer to as a Node. These points of reference (Nodes) create the coordinates to draw a wireframe Cube as they are referenced to its own plane. The centre of the Cube is given as a global x,y position. Node (1) can then be expressed on the X axis as +10 from gy. For the Y axis as it is above gx and an even distance again +10. Looking down from above we can also see it lies in front of the screen on the Z axis, this places the front of the object closer to us so here we can give it a value of -10.



	DATA 8 :REMark Number of Nodes
Node(1) xyz is 10, 10,-10	DATA 10, 10,-10
Node (2) xyz is -10, 10,-10	DATA -10, 10,-10
Node (3) xyz is -10,-10,-10	DATA 10,-10,-10
Node (4) xyz is 10,-10,-10	DATA 10,-10,-10
Node (5) xyz is 10, 10, 10	DATA 10, 10, 10
Node (6) xyz is -10, 10, 10	DATA -10, 10, 10
Node (7) xyz is -10,-10, 10	DATA -10,-10, 10
Node (8) xyz is 10,-10, 10	DATA 10,-10, 10

As a set of DATA lines the above can be fed into an Array to keep the basic configuration information. This will apply to not only our Cube but with any polyhedron constructed with multiple Nodes.

DIM x(n),y(n),z(n) where n is the number of Nodes of our polyhedron.

Vector Calculations
A Vector is described as a distance in a particular direction. For our purposes this represents the lines drawn between Nodes to present our object in a wire framework. Vectors are calculated as an xy screen coordinate derived from a Global xy at the centre of our object and each Node xyz coordinate.

To create our 3D Object we use trigonometry to find the position of a rotating point (x y) set around a central origin at a distance (r) and by degrees (a).

$x = r \times COS(a)$
$y = r \times SIN(a)$

If we then rotate further the angle to b:

$x' = r \times COS(\alpha + b)$
$y' = r \times SIN(\alpha + b)$

By using trigonometric addition of each equation:

$x' = r \times COS(a)\ COS(b) - r \times SIN(a)\ SIN(b)$
$y' = r \times SIN(a)\ COS(b) + r \times COS(a)\ SIN(b)$

Then substituting in the values for x and y above, we get an equation for the new coordinates as a function of the old coordinates and the angle of rotation:

$x' = x \times COS(b) - y \times SIN(b)$
$y' = y \times COS(b) + x \times SIN(b)$

The above describes one plane we have three XYZ. For now we can combine the required function for COS and SIN of the angle to be used with each plane:

ra=+.5 : c = COS(ra) : s = SIN(ra)

Then the code for position in each plane is as follows:

yt = y : y = c x yt – s x z : z = s x yt + c x z   X axis (y, z planes)
xt = x : x = c x xt + s x z : z = s x xt + c x z   Y axis (x, z planes)
xt = x : x = c x xt – s x y : y = s x xt + c x y   Z axis (x, y planes)

Where yt, xt hold the previous x, y coordinate values. The x y z are updated with new values. Following this the 3D coordinates are transposed into 2D screen positions with the following:

vx = gx + (x(n) x fs) / (z(n) + fs)
vy = gy + (y(n) x fs) / (z(n) + fs)

Where gx,gy are the global coordinates and fs is a scale factor that determines how much we have zoomed in or out from an imaginary focal point.

The above calculations for each Nodes vx(n) and vy(n) screen coordination can be set within a FOR loop and stored in a Dimentioned Array.

DIM vx(n),vy(n) where n is the same as the number of Nodes

QB3D Nodes, Vectors & Frames

Displaying our Cube we might begin by reviewing its components. A Cube has eight coordinate points (Nodes) and six sides (Frames). As with any polyhedron we will need to identify the number of Nodes, their xyz values from which we can then calculate (Vector) values vx,xy for the 2D screen coordinates. Once we have these we can go on to work out each Frames group of coordinates. As a Frame is by definition a closed area we also have the option to leave unfilled as a wireframe or colour fill it to create a solid Object.

QB3D Screen Display

As already stated a Frame is a surface area contained within a set of linked Nodes. What is needed is a Data set to identify which group of linked Nodes encloses each area. Then the SuperBASIC LINE function can be used to draw the shape of each to construct a wireframe of the Object.

	vres	DATA 6
Frame (1)	Vector 1 to 2 to 3 to 4 ink	DATA 1,2,3,4, 4
Frame (2)	Vector 1 to 2 to 3 to 4 ink	DATA 1,4,8,5, 240
Frame (3)	Vector 1 to 2 to 3 to 4 ink	DATA 4,3,7,8, 240
Frame (4)	Vector 1 to 2 to 3 to 4 ink	DATA 3,2,6,7, 240
Frame (5)	Vector 1 to 2 to 3 to 4 ink	DATA 2,1,5,6, 240
Frame (6)	Vector 1 to 2 to 3 to 4 ink	DATA 5,6,7,8, 2

```
RESTORE vres :READ vn :INK bg1
FOR lp=1 TO vn
   READ a,b,c,d,i:
   LINE vx(a),vy(a) TO vx(b),vy(b) TO vx(c),vy(c) TO vx(d),vy(d) TO vx(a),vy(a)
END FOR lp
```

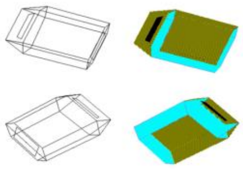
A FOR loop with READ function to call upon the lines of DATA provides the instruction set to build our wireframe Cube. At the end of each line of DATA I have added a fifth value this I use to set the ink colour for each Frame. By adding a FILL command to our FOR loop the area within a Frame can be colour filled to make our 3D wireframe look like a solid Object.

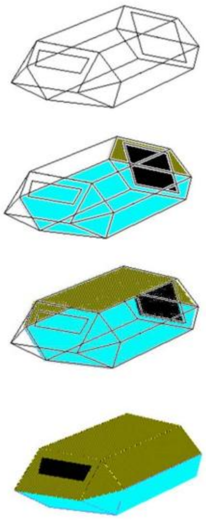
```
RESTORE vres :READ vn
FOR lp=1 TO vn
   READ a,b,c,d,i
   IF cset=1 :INK bg2 :FILL 0 :ELSE INK i :FILL 1:PAUSE 5
   LINE vx(a),vy(a) TO vx(b),vy(b) TO vx(c),vy(c) TO vx(d),vy(d) TO vx(a),vy(a):FILL 0
END FOR lp
```

The **If** statement here allows the colour setting **cset** to be turned On/Off switching between wireframe and solid Object. **bg2** is the default ink colour.

QB3D Frame FILL Drawbacks

Manipulating solid objects in three dimensions can take a lot of computing power. While in static mode my FILL command fills each frame in sequence from an instruction set, in this case the same ordered list that applies to write the wireframes. The sequence was chosen so those frames out of view are filled first with the later ones covering the viewed surfaces last. The problem is as the Object is rotated away from this initial settings in any of its three planes then the Frames hidden and those in front will change.





The examples of Frame FILLs on the right are taken from my listed sequence. Those above are Fills after random rotational changes have taken place. As you can see the problem is matching sequence with image displayed. To identify Frames surfaces to be filled as the object rotates in any of the three planes is another ball game and beyond the scope of this my entry level of Exploration into QL 3D Rotation Graphics.

Note: Although I have chosen to use quadrilaterals as my standard Frame I'm led to believe the gaming industry reduces all their Frames to triangles.

QBITS 3D Programs

This code has been developed using **QL2K** and **SMSQ** emulators running under Windows 7. There are four programs, the first is initial trial of xyz rotation code. The second and third are the Wireframe programs with commands that allow manipulation of parameters controlling various aspects of movement, size and perspective. The fourth provides Node and Frame Data to configure three Objects, a Cube, a Hexagon shape and my attempt at a Space a Shuttle.

QB3D_Cube	Basic code to rotate a Cube.
QB3D_Wire512	Program for standard QL 512x256 screens.
QB3D_Wire768	for the SMSQ 768x512 screen and 16 bit colour.
QB3D_Data1	Data coding for Cube, Hexagon, Shuttle

QBIT 3D Movement

Movement is accomplished in a number of ways; movement across the screen is repositioning the Objects global xy coordinates. This is achieved by use of the cursor keys changing the gx,gy values. Rotary movement is a change of angle in one of the three planes xy, zy, zx Roll/Spin/Loop. Pressing any of the **zZxXyY** keys alters the angle for its corresponding plane. These are processed by the Procedure **Obj_Calc.**

```
280 DEFine PROCedure Obj_Calc
282 cx=COS(rx):sx=SIN(rx)
284 cy=COS(ry):sy=SIN(ry)
286 cz=COS(rz):sz=SIN(rz)
288 FOR np=1 TO no
290     yt=y(np):y(np)=cx*yt-sx*z(np):z(np)=sx*yt+cx*z(np)
292     xt=x(np):x(np)=cy*xt+sy*z(np):z(np)=sy*xt+cy*z(np)
294     xt=x(np):x(np)=cz*xt-sz*y(np):y(np)=sz*xt+cz*y(np)
296     vx(np)=gx+(x(np)*fs)/(z(np)+fs)
298     vy(np)=gy+(y(np)*fs)/(z(np)+fs)
300 END FOR np
302 END DEFine
```

Part of this calculation is the Focal scale (**fs**). Imagine a large building from a distance its shape is fairly uniform. Standing at one corner, the height above us as opposed to the height of the building further down the street appears out of proportion to is true measurement. This is what we understand as perspective, the appearance of things relative to one another as determined by their distance from the viewer and is the technique of representing three-dimensional objects on a two-dimensional surface.

Using keys **D** or **I**, **fs** can be Decreased or Increased between 80 and 800. If you reduce the **fs** value below 80 the image distorts and becomes a little weird. The affect of **fs** at its lower vales also slightly enlarges the Object. However for better control of Object size I decided to added a Vector Size option.

QB3D Vector Size

To avoid the obvious distortion of **fs** this led me to look for a way to **E**nlarge or **R**educe an Objects size. The process of reading and storing Nodes xyz values gave me the idea of adding a multiplier to the xyz value and thereby being able to expand or reduce an Objects size in a uniform manner. The vector size **vs** is simply that with a range from 0.5 to 1.5 in 0.1 increments.

```
292 DEFine PROCedure Obj_Array
294 LOCal lp,a,b,c :RESTORE nres :READ no
296 FOR lp=1 TO no
298     READ a,b,c  :x(lp)=a*vs:y(lp)=b*vs:z(lp)=c*vs
300 END FOR lp
302 END DEFine:
```

QB3D Frame FILL
The **F** key toggles On/Off Frame FILL. On loading the sequence is set so those Frames in the forefront will be filled last. Unfortunately if the Object has been rotated on any planes since being loaded the Frames FILL action may not give the outcome you were expecting. That is to say the order of overlap for filling Frames may not be the forefront ones last.

QB3D Background
While deciding on useful things for the program it occurred to me that a user might prefer a Black or White background. Pressing **B** or **W** changes the colour of PAPER (bg1) and INK. (bg2) either a Black background with white INK, or White background with black INK.

QB3D Node ID
It would seem illogical to not include the ability to identify the Nodes of an Object. Pressing key **N** toggles On/Off **nset,** which adds printing the Node ID's as part of **Obj_Draw**. For this I make use of the CURSOR graphics coordinate system:

```
IF nset=1 :CURSOR vx(n),vy(n),-2,2 :PRINT n       (n being the Node number)
```

Note:
Using the **xXyYzZ** keys as expected will Loop/Spin/Roll respectively. However, once an Object has been rotated especially in the Roll (xy plane) then Loop (xX) key and/or Spin (yY) key commands can act differently.

QB3D QL Platforms
The **QB3D_Wire512** Mode 4 screen 512x256 was loaded and run, be it too slow for any real sort of performance on a **QemuLator** in basic QL speed with 128k memory. Not sure why but the background colour overwrite appeared not to clear all of the previous wireframe.

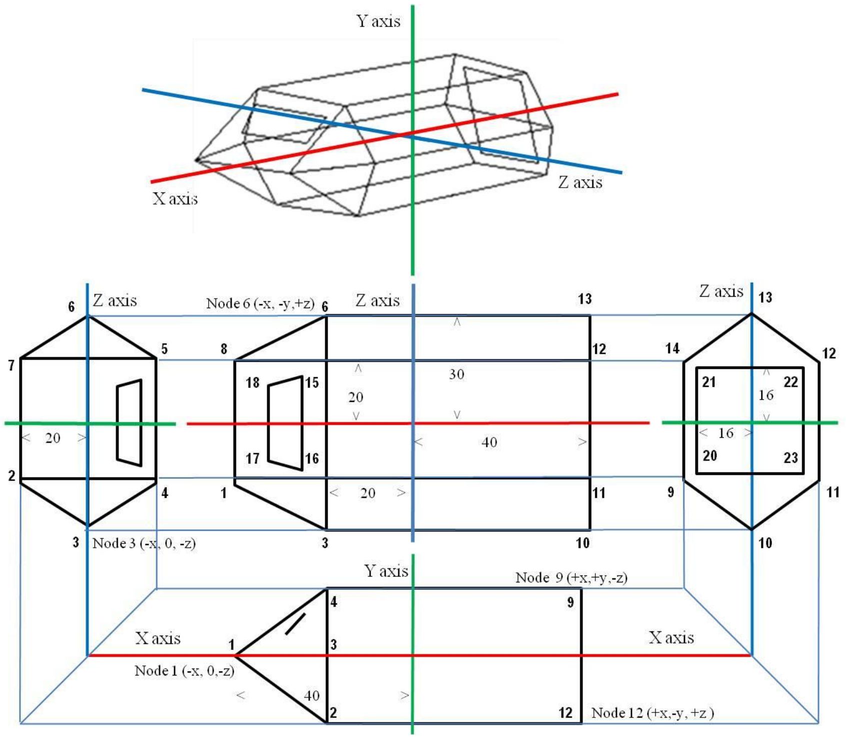
QB3D_Wire512 runs OK with the **QL2K** or **SMSQ** emulators. But by far the smoothest is the **QB3D_Wire768** version developed on the **SMSQmulator**. This takes advantage of the greater screen size and 16bit colour.

QB3D Future Thoughts
It would be nice to be able to Frame FILL with the forefront frames filled last whatever the Objects rotated position and so move and rotate the Object as a solid image. I guess a Companion Program to construct Wireframe Objects and generate their Node, Vector and Frame DATA lists would be useful.

QB3D Wireframe Design

To expand on the simple wireframe objects of Cube and Hexagon I have included my Space Shuttle. First draw your object with a plan, front and side elevation then identify the Nodes (xyz) in their planes XYZ with relevant + - values. Then select distances between Nodes from which it’s just the tedious job of listing each Node xyz as lines of DATA.

Here’s the basic layout design for my Space Shuttle.



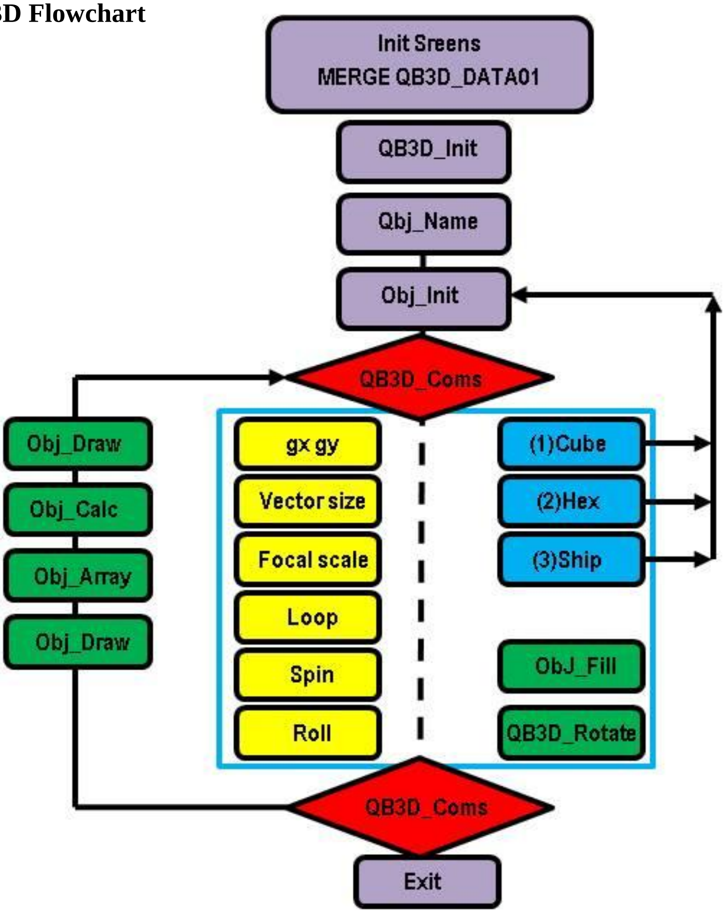
Having created your DATA List of Node xyz’s you then need to identify which Node to connect to which, so as to generate the Frames of your wireframe Object. Making these DATA lists hopefully should not be to taxing.

These can be added or created as a new **QB3D_Data** file following the layout presented on pages 17,18,19. Remember to type in the RESTORE references **nres, vres** into **OBj_Init** and Object **names** into **the Obj_Name** Procedure so they appear on the screen layouts.

QB3D_Wire Procedures

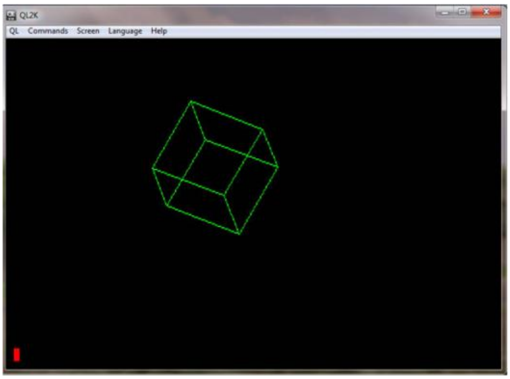
Obj_Name	Displays Object Names on screen menu
Obj_Init `	Sets the DATA RESTORE references for Object.
QB3D_Init	Sets screen layout and KEY information.
QB3D_Coms	Serves as main Menu loop to access functions.
QB3D_Rotate	Animation of 3D Object to Loop/Spin/Roll.
Obj_Array	Loads Node xyz and sets vector size.
Obj_Calc	Calculates new vx,vy coordinates of Object.
Obj_draw	Draws Object to screen.
OBj_Fill	Frames FILL with colour to present a solid Object.

QB3D Flowchart



This is the basic Program for 3D Rotation Graphics

```
100 REMark QB3D_Cube (Rotating Cube)
102 :
104 MODE 4:WINDOW 512,200,0,0:PAPER 0:INK 4:CLS:SCALE 100,0,0
106 DIM x(8),y(8),z(8),vx(8),vy(8)
108 vl=16:fs=10000:ra=.1                    :REMark Vector length : Focal point: Rotation angle

112 CLS
114 x(1)=-vl:y(1)=-vl:z(1)=-vl              :REMark Nodes
116 x(2)=-vl:y(2)=+vl:z(2)=-vl
118 x(3)=+vl:y(3)=+vl:z(3)=-vl
120 x(4)=+vl:y(4)=-vl:z(4)=-vl
122 x(5)=-vl:y(5)=-vl:z(5)=+vl
124 x(6)=-vl:y(6)=+vl:z(6)=+vl
126 x(7)=+vl:y(7)=+vl:z(7)=+vl
128 x(8)=+vl:y(8)=-vl:z(8)=+vl

132 ra=ra+.1:c=COS(ra):s=SIN(ra)

136 FOR np=1 TO 8
138    REMark Rotation on X Axix
140    yt=y(np):y(np)=c*yt-s*z(np):z(np)=s*yt+c*z(np)
142    REMark Rotation on Y Axis
144    xt=x(np):x(np)=c*xt+s*z(np):z(np)=s*xt+c*z(np)
146    REMark Rotation on Z Axis
148    xt=x(np):x(np)=c*xt-s*y(np):y(np)=s*xt+c*y(np)
150    REMark Points Projections and Translations to Screen Coordinates
152    vx(np)=80+(x(np)*fs)/(z(np)+fs)
154    vy(np)=50+(y(np)*fs)/(z(np)+fs)
156 END FOR np

160 LINE vx(1),vy(1) TO vx(2),vy(2)          :REMark Vectors
162 LINE vx(2),vy(2) TO vx(3),vy(3)
164 LINE vx(3),vy(3) TO vx(4),vy(4)
166 LINE vx(4),vy(4) TO vx(1),vy(1)
168 LINE vx(5),vy(5) TO vx(6),vy(6)
170 LINE vx(6),vy(6) TO vx(7),vy(7)
172 LINE vx(7),vy(7) TO vx(8),vy(8)
174 LINE vx(8),vy(8) TO vx(5),vy(5)
176 LINE vx(1),vy(1) TO vx(5),vy(5)
178 LINE vx(2),vy(2) TO vx(6),vy(6)
180 LINE vx(3),vy(3) TO vx(7),vy(7)
182 LINE vx(4),vy(4) TO vx(8),vy(8)

186 PAUSE 5
188 GO TO 112
```



This is the **QBITS 3D** programs to run on most QL Platforms.

QB3D_Wire512
QB3D_Data01

```
100 REMark QB3D_Wire512 (Exploring QL 3D Rotation Graphics v.01)

104 MODE 4:OPEN#4,con_10x10a10x10_4
106 OPEN#3,scr_134x108a6x100:PAPER#3,1:SCALE#3,100,0,0:CSIZE#3,0,0
108 WINDOW#2,512,256,0,0:PAPER#2,0:CLS#2:WINDOW#2,512,208,0,0
110 WINDOW#1,364,182,142,26:BORDER#1,1,2:PAPER#1,0:INK#1,7:CLS:SCALE 200,0,0
112 WINDOW#0,496,40,8,210

116 ra=180:gx=140:gy=120:fs=800:vs=.8         :REMark Angle:Global xy:focal scale:vector size
118 aset=-1:cset=1:nset=2                     :REMark Toggle switches
120 bg1=0:bg2=7:k=49:wres=512                 :REMark Screen settings

124 MERGE flp1_QB3D_Data01

128 QB3D_Init:Obj_Name:Obj_Init:QB3D_Coms

132 DEFine PROCedure QB3D_Init
134 ch=1:CSIZE#ch,0,0:INK#ch,4
136 ch=2:CSIZE#ch,0,1:INK#ch,4
138 CURSOR#ch,300,6:PRINT#ch,'Exploring QL 3D Rotation Graphics'
140 CSIZE#ch,0,0:INK#ch,7
142 CURSOR#ch,6,14:PRINT#ch,'KEYS: Toggle animation On/Off <spacebar>'
144 INK#ch,4
142 CURSOR#ch,6,26:PRINT#ch,'Global x y        :<←↑↓→>'
144 CURSOR#ch,6,36:PRINT#ch,'Focal Scale       :< D I>'
146 CURSOR#ch,6,46:PRINT#ch,'Node ID On/Off    :< N >'
148 CURSOR#ch,6,56:PRINT#ch,'Vector Size       :< E  R >'
150 CURSOR#ch,6,66:PRINT#ch,'Frame FILL        :<F>'
152 CURSOR#ch,6,76:PRINT#ch,'Rotation          :Xx Yy Zz'
154 CURSOR#ch,6,86:PRINT#ch,'Background        :< B  W > '
160 ch=3:BORDER#ch,1,3:INK#ch,2:CLS#ch
162 CURSOR#ch,80,82:PRINT#ch,'Loop'
164 CIRCLE#ch,60,48,24,.25,0:LINE#ch,36,48 TO 60,48
166 INK#ch,4
168 CURSOR#ch,40,4:PRINT#ch,'Spin'
170 CIRCLE#ch,36,76,24,.25,PI/2:LINE#ch,36,48 TO 36,76
172 INK#ch,7
174 CURSOR#ch,30,94:PRINT#ch,'Roll'
176 CIRCLE#ch,26,40,24,.8,0:LINE#ch,26,40 TO 36,48
178 END DEFine
```

This is the **QBITS SMSQ** version with screen size 768x512and 16bit colour.

QB3D_Wire768
QB3D_Data01

```
100 REMark QB3D_Wire768 (Exploring QMSQ 3D Rotation Graphics v.01)

104 MODE 4:OPEN#4,con_0x0a0x0_4
106 OPEN#3,scr_160x160a20x178:PAPER#3,1:SCALE#3,100,0,0:CSIZE#3,0,0
108 WINDOW#2,768,512,0,0:PAPER#2,0:CLS#2:WINDOW#2,754,460,8,4
110 WINDOW#1,548,428,200,32:BORDER#1,2,5:PAPER#1,7:INK#1,0:CLS#1:SCALE#1,320,0,0
112 WINDOW#0,732,40,16,460

116 ra=180:gx=140:gy=120:fs=800:vs=.8        :REMark Angle :Global x,y :focal scale :vector size
118 aset=-1:cset=1:nset=2                    :REMark Toggle switches
120 bg1=7:bg2=0:k=49:wres=768                :REMark Screen settings

124 MERGE flp1_QB3D_Data01

128 QB3D_Init:Obj_name:Obj_Init:QB3D_Coms

132 DEFine PROCedure QB3D_Init
134 ch=1:CSIZE#ch,0,0:INK#ch,4
136 ch=2:CSIZE#ch,1,1:INK#ch,5:OVER#ch,1
138 FOR i=1 TO 3:CURSOR#ch,460+i,6:PRINT#ch,'Exploring QL 3D Rotation Graphics'
140 OVER#ch,0:CSIZE#ch,0,1:INK#ch,6
142 CURSOR#ch,12,6:PRINT#ch,'KEYS: Toggle animation On/Off <spacebar>'
144 INK#ch,5
146 CURSOR#ch,12,30:PRINT#ch,'Global x y          : <←↑↓→>'
148 CURSOR#ch,12,50:PRINT#ch,'Focal Scale         : <D  I>'
150 CURSOR#ch,12,70:PRINT#ch,'Node ID On/Off      : <N>'
152 CURSOR#ch,12,90:PRINT#ch,'Vector Size         : <E  R>'
154 CURSOR#ch,12,110:PRINT#ch,'Frame FILL         : <F>'
156 CURSOR#ch,12,130:PRINT#ch,'Rotation           : Xx Yy Zz'
158 CURSOR#ch,12,150:PRINT#ch,'Background         :<B or W>'
160 ch=3:BORDER#ch,1,3:INK#ch,3:CLS#ch
162 CURSOR#ch,120,130:PRINT#ch,'Loop'
164 CIRCLE#ch,60,48,24,.25,0:LINE#ch,36,48 TO 60,48
166 INK#ch,5
168 CURSOR#ch,40,8:PRINT#ch,'Spin'
170 CIRCLE#ch,36,76,24,.25,PI/2:LINE#ch,36,48 TO 36,76
172 INK#ch,6
174 CURSOR#ch,30,140:PRINT#ch,'Roll'
176 CIRCLE#ch,26,40,24,.8,0:LINE#ch,26,40 TO 36,48
178 END DEFine
```

Note: **QB3D_Wire768** uses window sizes above the range of Basic QL 512x256 and with 16bit colour. Hence the WINDOW size and CURSOR xy pixel coordinates are set to utilise the extra dimensions as is use of the 8 colours available.

Program from here on the same apart from Global **gy** and **CURSOR** settings

```
182 DEFine PROCedure QB3D_Coms
184 REPeat lp
186   SELect ON k
188      =27    :INK#2,7:CLOSE#3:STOP
190       =66,98    :bg1=0:bg2=7 :PAPER#1,0   :CLS#1   :REMark (B)lack background
192       =87,119   :bg1=7:bg2=0 :PAPER#1,7   :CLS#1   :REMark (W)hite background
194      =49,50,51,52,53 :Obj_Init                     :REMark Load Object DATA
196      =32        :IF aset=-1:aset=-1:ELSE aset=-1   :REMark Toggle On/Off animation
198      =70,102    :IF cset= 1:cset= 2:ELSE cset= 1   :REMark (F)rame FILL
200      =78,110    :IF nset= 1:nset= 2;ELSE nset=1    :REMark (N)ode ID On/Off
202      =69,101    :vs=vs+.1     :IF vs>=1.5   :vs=1.5   :REMark (E)xpand Vector size
204       =82,114   :vs=vs-.1     :IF vs<= .5   :vs= .5   :REMark (R)educe Vector size
206       =68,100   :fs=fs-10     :IF fs< 80    :fs= 80   :REMark (D)ecrease Focal scale
208       =73,105   :fs=fs+10     :IF fs>800    :fs=800   :REMark ( I )ncrease Focal scale
210      =192       :gx=gx-2      :IF gx<= 20   :gx= 20 : :REMark 🡐 move left
212      =200       :gx=gx+2      :IF gx>=280   :gx=280   :REMark 🡒 move right
214      =208       :gy=gy+2      :IF gy>=190   :gy=190   :REMark 🡑 move up
216      =216       :gy=gy-2      :IF gy<= 10   :gy= 10   :REMark 🡓 move down
218      =88        :rx=rx-15     :IF rx<=0:rx=360        :REMark (x) Clockwise Loop
220      =120       :rx=rx+15     :IF rx>=360:rx=0        :REMark (X) Anti-clockwise Loop
222      =89        :ry=ry-15     :IF ry<=0:ry=360        :REMark (y) Clockwise Spin
224      =121       :ry=ry+15     :IF ry>=360:ry=0        :REMark (Y) Anti-clockwise Spin
226      =90        :rz=rz-15     :IF rz<=0:rz=360        :REMark (z) Clockwise Roll
228      =122       :rz=rz+15     :IF rz>=360:rz=0        :REMark (Z) Anti-clockwsie Roll
230   END SELect
232  Obj_Draw:INK bg2
234  CURSOR 160,168:PRINT 'gx:'&gx&'  gy:'&gy&'  vs:'&(vs*20)&'  fs:'&fs&'
236  CURSOR  12,168:PRINT 'rx:'&rx&'  ry:'&ry&'  rz:'&rz&'   '
238  k=CODE(INKEY$(#4,aset))
240 END  REPeat lp
242 END DEFine
```

Note: **QB3D_Wire768** higher screen resolution has an increased pixel range.

```
214                                :IF gy>=300:gy=300
216                                :IF gy<= 20:gy= 20
234  CURSOR 320,406
236  CURSOR   12,168
```

Note: Global **gx,gy,** vector size **vs** and focal scale **fs** are displayed on screen.
On start up a Cube is displayed.
Selection of (**1**), (**2**), (**3**) displays the selected Object in static mode.
To select background press Keys (**B**)lack or (**W**)hite
To switch Node ID On/Off press <**N**>.
Press <**F**> tol Fill an Objects Frames to create as solid image.
Press <**E**> enlarge or <**R**> reduce to change size of Object.
Press <**D**> decrease or < **I** > increase to change Focal scale (Perspective)
For Global positions use the < 🡐 🡒 🡑🡓> cursor keys.
Pressing <**xXyYzZ**> keys Loops/Spins/Rolls the Object clockwise or anticlockwise.
Pressing the <**spacebar**> activates the animation to Loop, Spin, Roll the Object.
Pressing the <**spacebar**> again returns to static mode.

These Procedures are the 3D Rotation Graphics engine.

```
246 DEFine PROCedure QB3D_Rotate
248 ch=3:OVER#ch,-1
250  INK#ch,7:LINE#ch,26,40 TO 26-20*COS(RAD(ra+90)),40-22*SIN(RAD(ra+90))
252  INK#ch,4:LINE#ch,36,76 TO 36+22*SIN(RAD(ra)),76-7*COS(RAD(ra))
254  INK#ch,2:LINE#ch,60,48 TO 60+7*SIN(RAD(ra)),48-22*COS(RAD(ra))
256  ra=ra-15:IF ra<0:ra=360
258 OVER#ch,0:ch=1
260 rx=rx+5:IF rx>360:rx=0
262 ry=ry+5:IF ry>360:ry=0
264 rz=rz+5:IF rz>360:rz=0
266 END DEFine

270 DEFine PROCedure Obj_Array
272 LOCal lp,a,b,c:RESTORE nres:READ no
274 FOR lp=1 TO no
276    READ a,b,c:x(lp)=a*vs:y(lp)=b*vs:z(lp)=c*vs
278 END FOR lp
280 END DEFine

284 DEFine PROCedure Obj_Calc
286 cx=COS(RAD(rx)):sx=SIN(RAD(rx))
288 cy=COS(RAD(ry)):sy=SIN(RAD(ry))
290 cz=COS(RAD(rz)):sz=SIN(RAD(rz))
292 FOR np=1 TO no
294     yt=y(np):y(np)=cx*yt-sx*z(np):z(np)=sx*yt+cx*z(np)
296     xt=x(np):x(np)=cy*xt+sy*z(np):z(np)=sy*xt+cy*z(np)
298     xt=x(np):x(np)=cz*xt-sz*y(np):y(np)=sz*xt+cz*y(np)
300     vx(np)=gx+(x(np)*fs)/(z(np)+fs)
302     vy(np)=gy+(y(np)*fs)/(z(np)+fs)
304 END FOR np
306 END DEFine

310 DEFine PROCedure Obj_Draw
312 LOCal lp,vn,a,b,c,d,i:RESTORE vres:READ vn:INK bg1
314 FOR lp=1 TO vn
316     READ a,b,c,d,i:FILL 1
318     LINE vx(a),vy(a) TO vx(b),vy(b) TO vx(c),vy(c) TO vx(d),vy(d) TO vx(a),vy(a):FILL 0
320 END FOR lp
322 Obj_Array:FOR n=1 TO no:CURSOR vx(n),vy(n),-2,2:PRINT n
324 IF aset=1:cset=1:QB3D_Rotate
326 Obj_Calc :RESTORE vres:READ vn
328 FOR lp=1 TO vn
330     READ a,b,c,d,i
332     IF cset=1:INK bg2:FILL 0:ELSE INK i:FILL 1:PAUSE 5
334     LINE vx(a),vy(a) TO vx(b),vy(b) TO vx(c),vy(c) TO vx(d),vy(d) TO vx(a),vy(a):FILL 0
336 END FOR lp
338 IF nset=1 AND cset=1:FOR n=1 TO no:CURSOR vx(n),vy(n),-2,2:PRINT n
340 END DEFine
```

This is the DATA file MERGE'd with the QB3D_Wire? (512 or 768)

```
400 REMark QB3D_DATA01 (Cube Hexagon Shuttle ? ?)

404 DEFine PROCedure Obj_Name
406 IF wres=512:CURSOR#0,12,4:PRINT#0,'  (1)Cube  (2)Hexagon  (3)Shuttle'
408 IF wres=768
410   CSIZE#2,0,1:INK#2,6
412   CURSOR#2,12,338:PRINT#2,'(1)Cube'
414   CURSOR#2,12,358:PRINT#2,'(2)Hexagon'
416   CURSOR#2,12,378:PRINT#2,'(3)Shuttle'
418   CURSOR#2,12,398:PRINT#2,'(4)?'
420   CURSOR#2,12,418:PRINT#2,'(5)?'
422 END IF
424 END DEFine
```

Note: When DATA QB3D_Data01 file is MERGE'd with QB3D_Wire program file the wres value is read and the relative Object names are loaded.

WARNING:

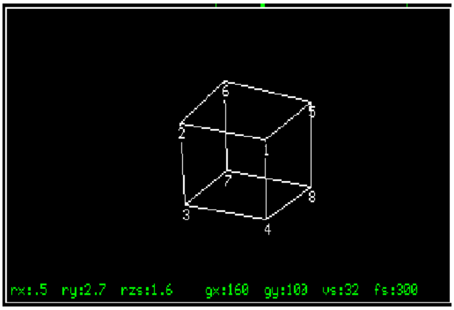
nres, vres hold line number references to **DATA**.
(If program Lines are renumbered these references may not still apply)

```
428 DEFine PROCedure Obj_Init
430 REMark WARNING maintain correct nres:vrs:fres numbers
432 IF k=49   :nres=452  :vres=470  :rx=345  :ry=330  :rz=  0
434 IF k=50   :nres=490  :vres=518  :rx= 15  :ry= 60  :rz= 60
436 IF k=51   :nres=544  :vres=592  :rx= 60  :ry=150  :rz=165
438 REMark IF k=52 :nres=600 :vres=600 :rx= .2 :ry= .2  :rz= .2
440 REMark IF k=53 :nres=600 :vres=600 :rx= .2 :ry= .2  :rz= .2
442 RESTORE nres :READ n:DIM x(n),y(n),z(n),vx(n),vy(n)
444 CLS#1
446 END DEFine
```

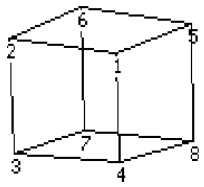
Note:
New **Objects DATA** can be added with values entered into respective **nres, vres,** at Lines 432 - 440 to introduce the new **Objects DATA** for **Nodes & Frames.**

Note:
The **rx,ry,rz** values are based on positioning the Object in line with designed Node points. This way the sequence of Frames FILLs can be worked out.

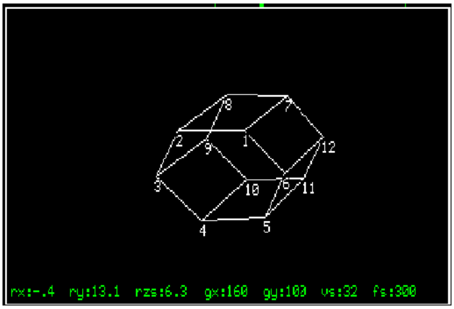
```
450 REMark Cube
452 DATA   8                :REMark Nodes
454 DATA  20, 20,-20        :REMark Node 1
456 DATA -20, 20,-20
458 DATA -20,-20,-20
460 DATA  20,-20,-20        :REMark Node 4
462 DATA  20, 20, 20        :REMark Node 5
464 DATA -20, 20, 20
466 DATA -20,-20, 20
468 DATA  20,-20, 20        :REMark Node 8

472 DATA   6                :REMark Frames
474 DATA 2,3,7,6,2          :REMark back Frame
476 DATA 3,4,8,7,240
478 DATA 5,6,7,8,240
480 DATA 1,2,6,5,240
482 DATA 1,2,3,4,240
484 DATA 1,4,8,5,4          :REMark front Frame
```



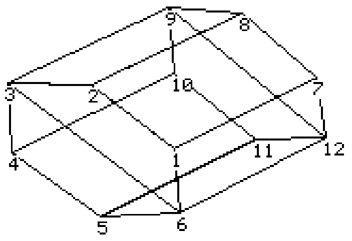
```
rx:.5  ry:2.7  rzs:1.6    gx:160  gy:100  vs:32  fs:300
```



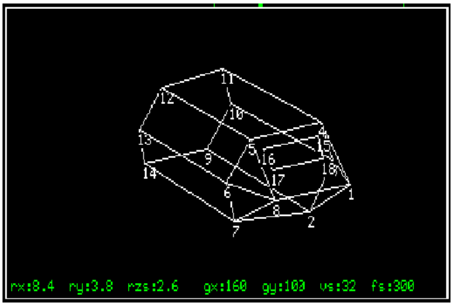
```
488 REMark Hexagon
490 DATA  12                :REMark Nodes
492 DATA  30,  0,-20        :REMark Node 1
494 DATA  15, 20,-20
496 DATA -15, 20,-20
498 DATA -30,  0,-20
500 DATA -15,-20,-20
502 DATA  15,-20,-20        :REMark Node 6
504 DATA  30,  0, 20         :REMark Node 7
506 DATA  15, 20, 20
508 DATA -15, 20, 20
510 DATA -30,  0, 20
512 DATA -15,-20, 20
514 DATA  15,-20, 20        :REMark node 12

518 DATA  10                :REMark Frames
520 DATA 9,10,11,12,2       :REMark rear frames
522 DATA 9,8,7,12,4
524 DATA 4,5,11,10,2        :REMark side frames
526 DATA 3,4,10,9,2
528 DATA 2,3,9,8,4
530 DATA 1,2,8,7,4
532 DATA 6,1,7,12,4
534 DATA 5,6,12,11,2
536 DATA 3,4,5,6,2          :REMark front frames
538 DATA 3,2,1,6,4
```



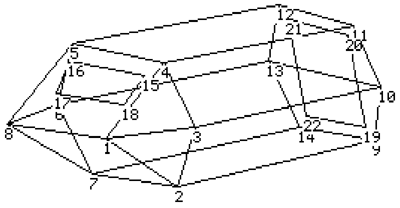
```
rx:-.4  ry:13.1  rzs:6.3  gx:160  gy:100  vs:32  fs:300
```



```
542 REMark Space Shuttle
544 DATA  22                    :REMark Nodes
546 DATA  40, 20,  0            :REMark Node 1
548 DATA  22, 20,-20            :REMark Node 2
550 DATA  20, 30,  0
552 DATA  20, 20, 20
554 DATA  20,-20, 20
556 DATA  20,-30,  0
558 DATA  20,-20,-20            :REMark Node 7
560 DATA  40,-20,  0            :REMark Node 8
562 DATA -40, 20,-20            :REMark Node 9
564 DATA -40, 30,  0
566 DATA -40, 20, 20
568 DATA -40,-20, 20
570 DATA -40,-30,  0
572 DATA -40,-20,-20            :REMark Node 14
574 DATA  24, 16, 16            :REMark Node 15
576 DATA  24,-16, 16
578 DATA  30,-14,  8
580 DATA  30, 14,  8            :REMark Node 18
582 DATA -40, 16,-16            :REMark Node 19
584 DATA -40, 16, 16
586 DATA -40,-16, 16
588 DATA -40,-16,-16            :REMark Node 22

592 DATA  16                    :REMark Frames
594 DATA 9,10,13,14,243         :REMark rear Frames
596 DATA 10,11,12,13,240
598 DATA 19,20,21,22,0          :REMark Rear Door

600 DATA 2,9,14,7,243           :REMark side Frames
602 DATA 7,5,13,14,243
604 DATA 6,5,12,13,240
606 DATA 5,4,11,12,240
608 DATA 3,4,11,10,240
610 DATA 2,3,10,9,243

612 DATA 1,2,3,1,243            :REMark Front Frames
614 DATA 1,3,4,1,240
616 DATA 1,2,7,8,243
618 DATA 1,4,5,8,240
620 DATA 8,6,7,8,243
622 DATA 8,6,5,8,240
624 DATA 15,16,17,18,0          :REMark Pilot Window
```



```
rx:8.4  ry:3.8  rzs:2.6   gx:160  gy:103  vs:32  fs:300
```



QBITS Exploring 3D Rotation Graphics

Having obtained copies of the QB3D SuperBASIC code and loaded it into a recognised QL device. Use the QDOS command LRUN, as shown:-

LRUN flp1_QB3D_Wire512

Note: For most QL platform runs in Mode 4 512x256 screen size.

LRUN flp1_QB3D_Wire768

Note: For SMSQmulator with screen size 768x512 and 16 bit colour.

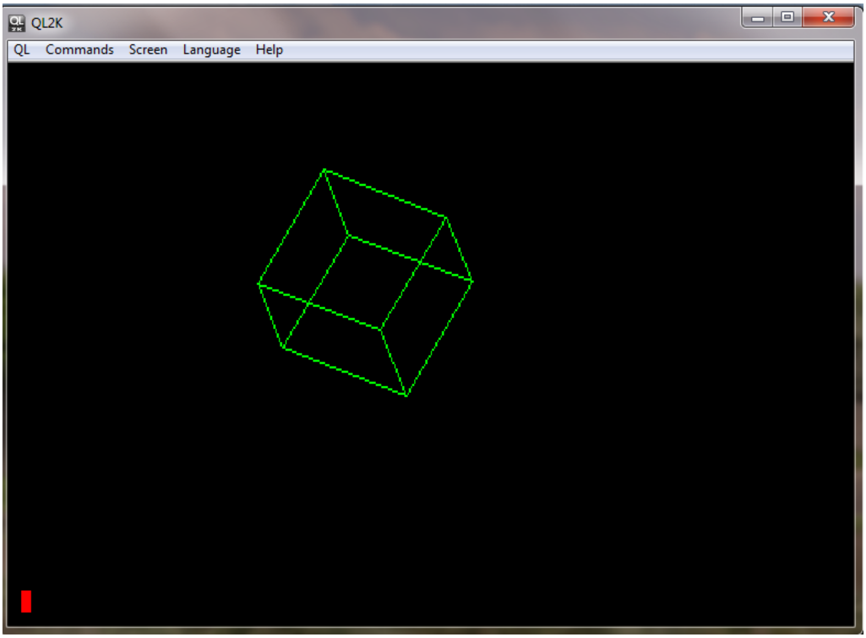
Check out the KEYS to activate the various functions and enjoy the experience. Reading this pdf file if your lacking in knowledge of 3D Rotation Graphics you're hopefully gain a better understanding..

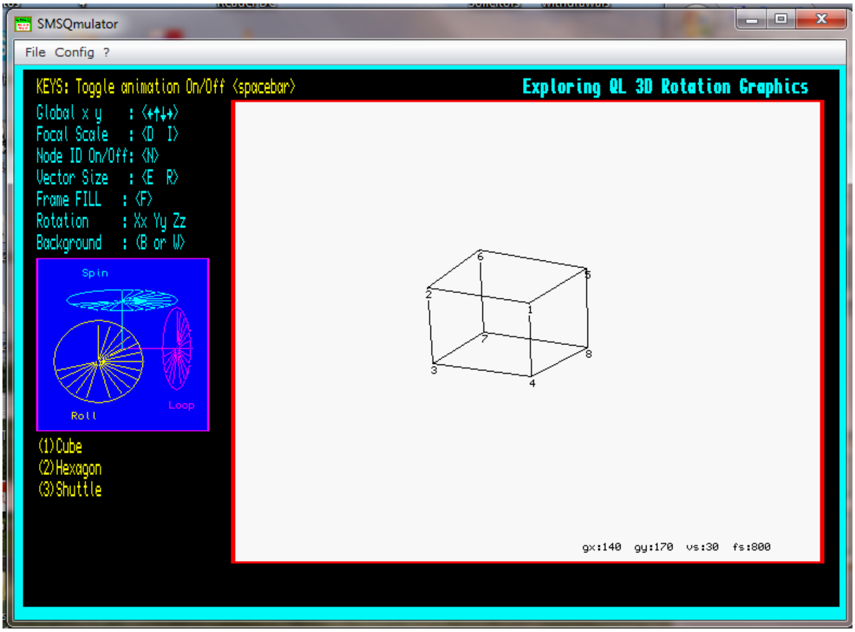
QL Emulators

There are several available for the original QL as well as its later spin offs. You can download these and run them on PC's, Desktops Laptops and Tablets under the Windows, Mac or Linux operating systems. Then there are the additional ROM's and toolkit extensions and an extensive number of useful programs all with plenty of helpful documentation available.

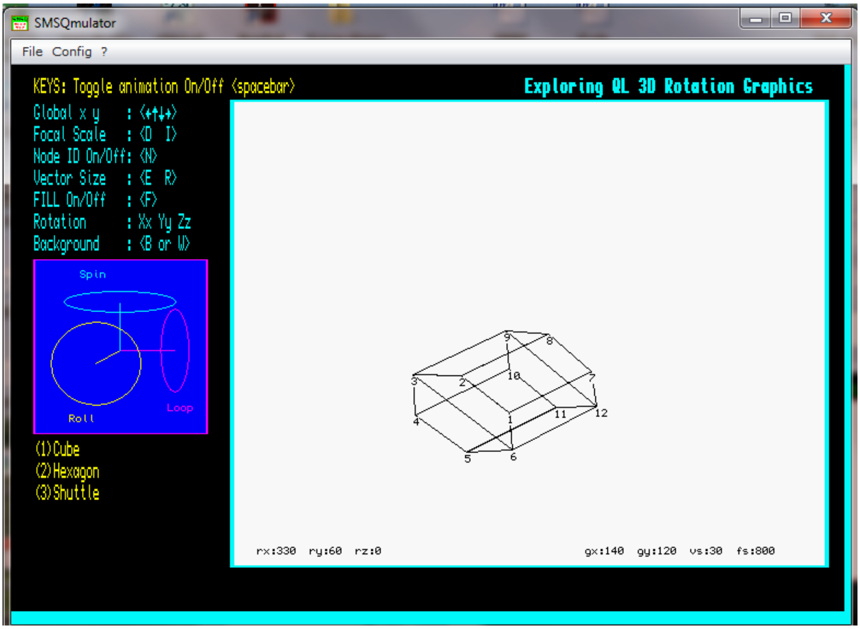
Check out Dilwyn's web site below for downloads helpful information and links to other suppliers of QL software and documentation.

http://www.dilwyn.me.uk/





```
KEYS: Toggle animation On/Off <spacebar>          Exploring QL 3D Rotation Graphics
Global x y    : <←→↑↓>
Focal Scale   : <D  I>
Node ID On/Off: <N>
Vector Size   : <E  R>
Frame FILL    : <F>
Rotation      : Xx Yy Zz
Background    : <B or W>

(1)Cube
(2)Hexagon
(3)Shuttle

gx:140  gy:170  vs:30  fs:800
```



SMSQmulator

File Config ?

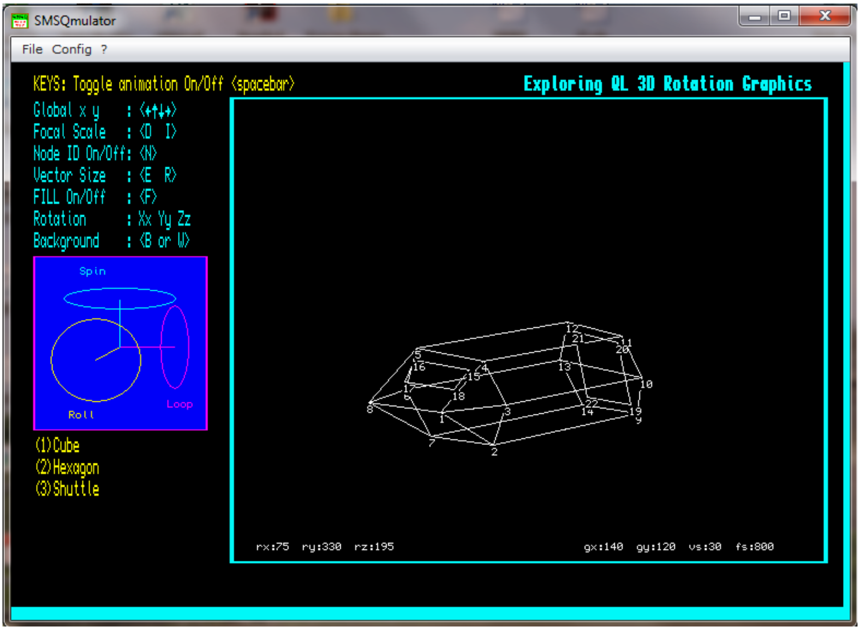
KEYS: Toggle animation On/Off (spacebar)

Exploring QL 3D Rotation Graphics

```
Global x y    : <←↑↓→>
Focal Scale   : <D  I>
Node ID On/Off: <N>
Vector Size   : <E  R>
FILL On/Off   : <F>
Rotation      : Xx Yy Zz
Background    : <B or W>
```

(1)Cube
(2)Hexagon
(3)Shuttle

```
rx:330  ry:60  rz:0                              gx:140  gy:120  vs:30  fs:800
```



SMSQmulator

File Config ?

KEYS: Toggle animation On/Off (spacebar)

Exploring QL 3D Rotation Graphics

```
Global x y    : <←↑↓→>
Focal Scale   : <D  I>
Node ID On/Off: <N>
Vector Size   : <E  R>
FILL On/Off   : <F>
Rotation      : Xx Yy Zz
Background    : <B or W>
```

(1)Cube
(2)Hexagon
(3)Shuttle

```
rx:75  ry:330  rz:195                            gx:140  gy:120  vs:30  fs:800
```