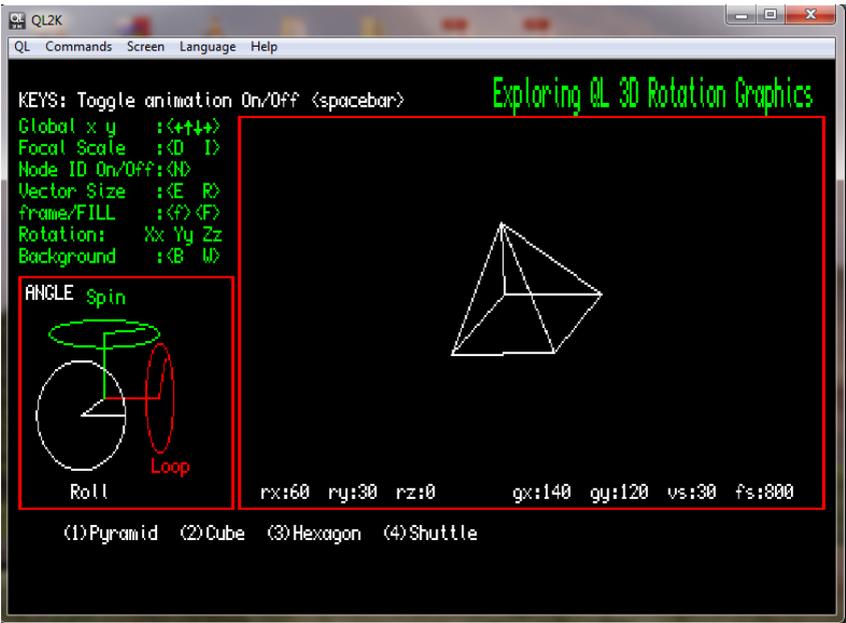


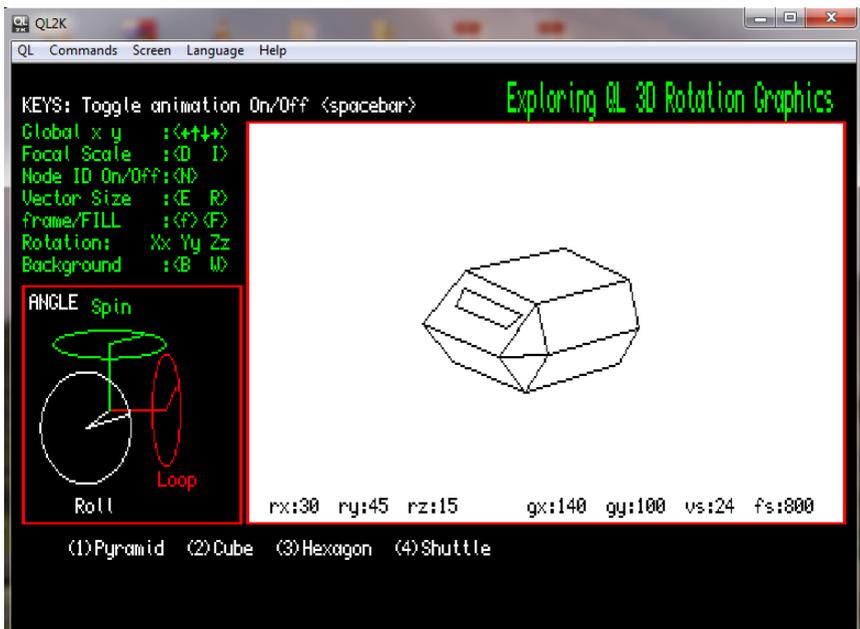
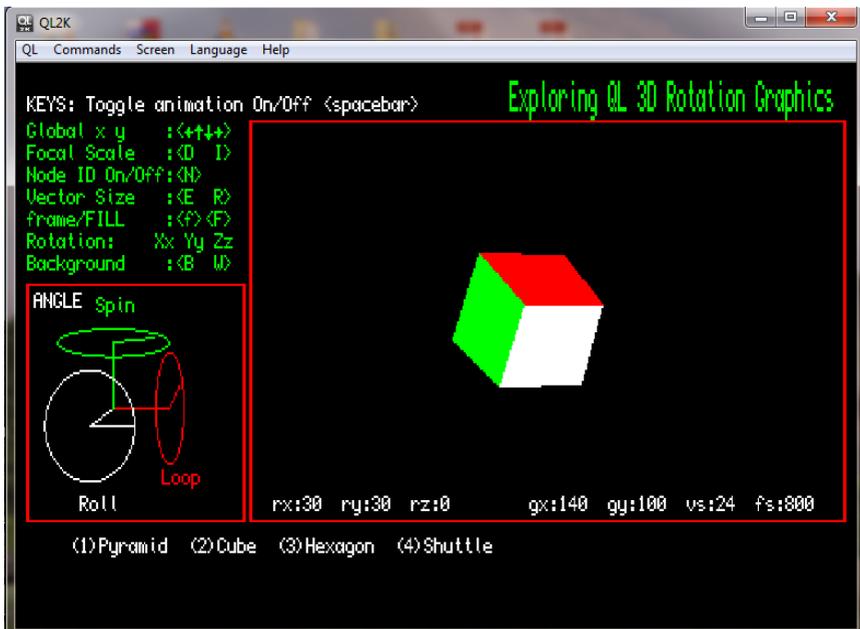


## Sinclair QL Exploring 3D Rotation Graphics



## Sinclair QL Exploring 3D Rotation Graphics







### **QBITS on Animation**

Magic Lanterns with flickering hand drawn cards, later replaced with photography pictures, all were used in the attempt 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 an Edison employee developed the Kinetoscope (a peep-hole motion picture viewer), introduced at the US Chicago World Fair of 1893. In France 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.

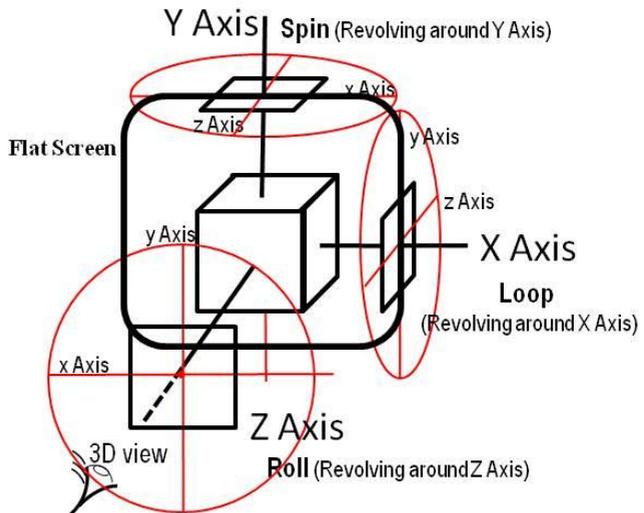
So having involved myself with QL SuperBASIC again I thought it was time to give 3D Rotation Graphics a bit of a spin (sorry for the pun). However what was my goal, the basic code for revolving a simple wireframe object for one. To move say a Cube about the screen, altering its global position and being able to zoom in and out. Create the illusion of perspective perhaps that has something to do with focal scale, but more about that later. As a finishing touch fill the visible surfaces of my wireframe 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 rotation. The most common being Roll, Pitch and Yaw associated with flying. I thought of 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, this is achieved by changing the xy coordinates values a 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. Three dimensional rotation changes the orientation within each of its relative axis. This alters the shape and size viewed and is known as a transformation. Converting a three dimensional object onto a two dimensional screen image requires manipulation of 3D coordinates into 2D coordinates. The coding for such requires a number of steps and involves basic trigonometry.

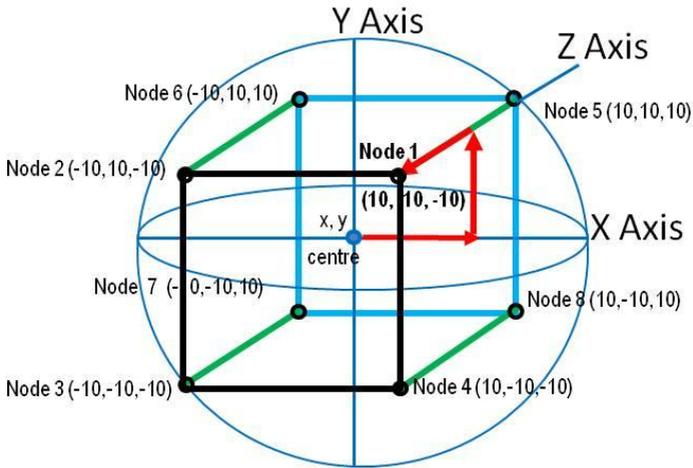


### Imaginary Eye View

Reviewing the diagram shown, 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. Using a Cube as our object in space half is sticking 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

**Initialising xyz coordinates**

Each point of reference that connects a 3D Object, be it a simple Cube or multisided polyhedron I shall refer to as a Node. These points (Nodes) identify the Objects coordinates to draw a Wireframe as referenced to each of its axis. The centre of the Cube is given as a Global x,y position. Following the red arrows <see below> Node (1) is shown on the X axis as +x units from gy [y=0]. On the Y axis it rises above the gx [y=0] by +y units. Looking down from above we can also see it lies in front of the screen on the Z axis, this places the object closer to us so here we can give it a value of -z units.



	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 and its 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 xy screen coordinates derived from a Global xy set at the centre of our object and each Node xy 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).

$$x = r \times \text{COS}(a)$$

$$y = r \times \text{SIN}(a)$$

If we then rotate further the angle to b:

$$x' = r \times \text{COS}(\alpha + b)$$

$$y' = r \times \text{SIN}(\alpha + b)$$

By using trigonometric addition of each equation:

$$x' = r \times \text{COS}(a) \text{COS}(b) - r \times \text{SIN}(a) \text{SIN}(b)$$

$$y' = r \times \text{SIN}(a) \text{COS}(b) + r \times \text{COS}(a) \text{SIN}(b)$$

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:

$$x' = x \times \text{COS}(b) - y \times \text{SIN}(b)$$

$$y' = y \times \text{COS}(b) + x \times \text{SIN}(b)$$

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:

$$ra = +.5 : c = \text{COS}(ra) : s = \text{SIN}(ra)$$

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

$$yt = y : y = c_x yt - s_x z : z = s_x yt + c_x z \quad \text{X axis (y, z planes)}$$

$$xt = x : x = c_x xt + s_x z : z = s_x xt + c_x z \quad \text{Y axis (x, z planes)}$$

$$xt = x : x = c_x xt - s_x y : y = s_x xt + c_x y \quad \text{Z axis (x, y planes)}$$

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:

$$vx = gx + (x_x fs) / (z + fs)$$

$$vy = gy + (y_x fs) / (z + fs)$$

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 vector calculation for each Node vx(n) and vy(n) screen coordination can be set within a FOR loop and stored in a Dimensioned Array.

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

### QB3D Movement & Conversion

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 then 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 its 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.

### QB3D Vector Size

To avoid the obvious distortion of **fs** this led me to look for a way to **Enlarge** or **Reduce** an Objects size. The process of reading and storing the Nodes xyz values gave me the idea of adding a multiplier 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 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 calculate **Vector** values **vx**, **vy** for the 2D screen coordinates. Having these we can work out each **Frame** and its group of coordinates.

### QB3D Screen Display

A Frame is the area contained within a set of linked Nodes. What is needed is a DATA set to identify these linked Nodes to our program. The SuperBASIC LINE function can then be used to draw the shape of each to construct a Wireframe of the Object.

	<b>vres</b>	DATA 6
Frame (1) Vector a - b - c - d		DATA 8,7,6,5
Frame (2) Vector a - b - c - d		DATA 2,6,7,3
Frame (3) Vector a - b - c - d		DATA 4,3,7,8
Frame (4) Vector a - b - c - d		DATA 5,1,4,8
Frame (5) Vector a - b - c - d		DATA 5,6,2,1
Frame (6) Vector a - b - c - d		DATA 1,2,3,4

```

RESTORE vres :READ vn
FOR lp=1 TO vn
  READ a,b,c,d
  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 calls upon the lines of DATA that provide the instruction set to load and build the Wireframe. The order in which they are presented has a significance that will be explained later when exploring how Wireframe images are turned into Solid images.

### QB3D Node ID

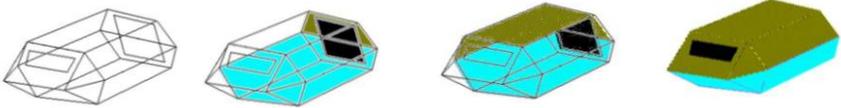
At this point it would seem logical to include the ability to identify the Nodes displayed in their screen positions as part of an Objects image. Pressing the **N** key toggles On/Off **nset**, which actions the print of Node ID's. For this I make use of the CURSOR graphics coordinate system:

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

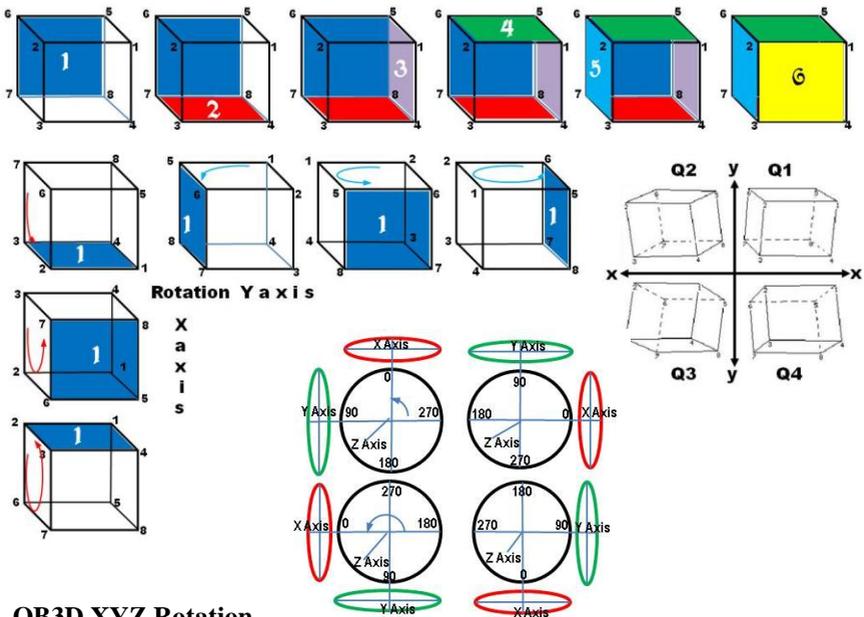
**Note:** When using the **xXyYzZ** keys to Loop/Spin/Roll respectively, once an Object has been rotated from its initial position the Roll/Loop and Spin key commands can act differently to what maybe expected. The position of the **Z,X,Y** axis become changed and so rotate in altered planes. An example of this is where the actions of **xX** (Loop) and **yY** (Spin) can be reversed.

### QB3D Wireframe to Solid Objects

As a Frame is by definition a closed area we have the option to leave it unfilled as a Wireframe or coloured in to create a Solid Object using the SuperBASIC FILL function.



The frame sequence mentioned on the previous page loads those Frames hidden from view first with the ones covering the viewed surfaces last. The problem is as the Object image is rotated away from initial settings in any of its three axis then the sequence of Frames hidden from view and those that come into view will change. The row of images below show the initial load and display of Frame surfaces for our Cube, and then the back frame as it **Spins** and **Loops** to different positions on screen, some hidden and some in view.



### QB3D XYZ Rotation

As an object is rotated in any of the three axis, as already mentioned the actions of the programs function keys **xXyYzZ** change. In the example shown rotation around the Z axis the actions of Spin and Loop change as it moves through each quadrant. Hopefully my diagram above explains this better than I can put into words. This shows the complexity you may face when writing code to display the image of a rotating 3D object.

### QB3D Hidden Surface Removal

However all is not lost. In Exploring QL 3D Rotation Graphics I have used planar polygons of which each Frame surface has a unique property. It has two sides, one which looks internally and the other outwardly. Therefore by determining the outward direction of a frames surface we can then use this to identify if it is pointing away or towards our view point.

The two basic types of hidden surface removal are object-space used for three dimensional processing and image-space used for two dimensional processing when determining hidden surfaces.

As the above title implies a method is sought to remove those hidden surfaces (Frames) of an object to provide a more realistic representation. Namely we seek an algorithm that identifies those frame surfaces of an object that are not seen from the view point. The most common method used in computing for carrying out this action is called the plane equation method.

In simple terms you compute a Vector Normal to a plane (Frame surface) such that its value indicates whether it is facing away from or towards the viewer. I have used the counter or anti clockwise coordinates system for defining my Frames. This is known as the left handed rule for the plane equation shown below. (The is an alternative called the right handed or clockwise system)

These are based on the equation:  $Ax+By+Cz+D=0$

where the Vector Normal (N) to the plane is  $N=[A \ B \ C]$

and where  $C > 0$  is a surface facing away

and where  $C \leq 0$  is a surface facing towards the viewer.

Obtaining the Vector Normal we use an equation based on the plane passing through three points:  $P1=(x1, y1, z1)$ ,  $P2=(x2, y2, z2)$ ,  $P3=(x3, y3, z3)$ :

$$\begin{aligned} &x - x1 \quad y1 - y1 \quad z - z1 \\ &x2 - x1 \quad y2 - y1 \quad z2 - x1 = 0 \\ &x3 - x1 \quad y3 - y1 \quad z3 - x1 \end{aligned}$$

This is equivalent to:  $Ax+By+Cz+D=0$

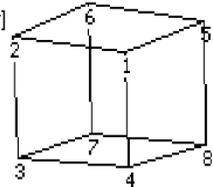
and where  $C = (x2 - x1)*(y3 - y1) - (x3 - x1)*(y2 - y1)$

C is the value we are interested in to determine the outward facing direction of the Frame surface and whether it is towards or away from the viewer.

### QB3D Anti Clockwise Coordinate system

Going back to our Frame DATA lists you will notice that the Nodes for the front facing surface are 1,2,3,4 and are ordered in an anticlockwise manner. The back face 5,6,7,8 in the DATA list is ordered as 8,7,6,5 or clockwise. However if you were to view this surface rotated 180 degrees to the front 8,7,6,5 is counting in an anticlockwise direction.

```
DATA 8,7,6,5,bg2 :REMark back Frame [bg2 = background Ink colour]
DATA 2,6,7,3,2
DATA 4,3,7,8,4
DATA 5,1,4,8,3
DATA 5,6,2,1,5
DATA 1,2,3,4,bg2 :REMark front Frame
```



### QB3D Obj\_Cull

To obtain the Vector Normal of the Frame surface the three points P1,P2,P3 from the equation are substituted by three of the Frames Node xy coordinates.

In this case  $x(a), y(a)$   $x(b), y(b)$   $x(c), y(c)$

$$C=(x(b)-x(a))*(y(c)-y(a))-(x(c)-x(a))*(y(b)-y(a))$$

### QB3D Obj\_Draw

We now have all the elements required to draw our objects image to screen, the Node xyz coordinates, the Vector vx,vy coordinates, the Frame instruction set and a method of eliminating hidden frames.

```
205 DEFine PROCEDURE Obj_Draw
206 LOCAl lp,v,a,b,c,d,i:RESTORE vres:READ v
207 iset=2:Obj_Calc
208 FOR lp=1 TO v
209 READ a,b,c,d,i
210 IF cset=1:INK bg2:FILL 0:END IF
211 IF cset=2:Obj_Cull:IF c1>0:GO TO 214:END IF :INK bg2:FILL 0:END IF
212 IF cset=3:Obj_Cull:IF c1>0:GO TO 214:END IF :INK i :FILL 1:END IF
213 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
214 END FOR lp
215 Obj_Node:IF nset=2:INK bg2:FOR n=1 TO no:CURSOR vx(n),vy(n),-2,2:PRINT n
216 END DEFine
```

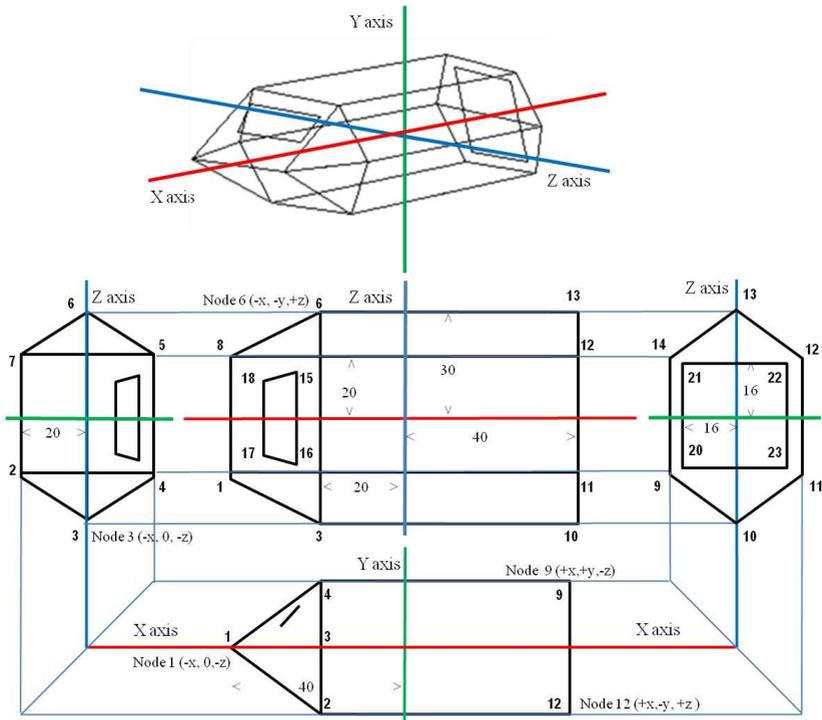
### QB3D Solid Object Mode

To control the Wireframe in default mode colour set **cset=1** and all of the Wireframe is drawn to screen. If **cset=2** the procedure **Obj\_Cull** is used to eliminate hidden frames and a Solid Wireframe is displayed. If **cset=3** again **Obj\_Cull** is used to eliminate hidden frames, but the viewed frame surfaces are now FILled. The colour for these frame surfaces is the fifth value entered on my **Frame** DATA lines (see DATA lines for the Cube above).

### QB3D Wireframe Design

To expand on the simple wireframe objects of Pyramid, Cube and Hexagon I have included my Space Shuttle. First draw your object with an overview including front and side elevations. Map this to the XYZ planes, then identify the Nodes (xyz) and their relevant units of distance +/- values. Then create a DATA list of each Node xyz value.

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



To generate the Wireframe a second list is required linking Nodes to form Frames. These are READ and used by the Plane Equation of **Obj\_Cull** to determine if the outward surface of the polygon is facing towards or away from point of observation. It is therefore important they are ordered correctly, that is Counter or anticlockwise.

These DATA lists can be added to or created within a new **QB3D\_Data** file following the format presented on pages 18,19,20. 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 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.

### QBITS 3D Programs

Version 3 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 four Objects a Pyramid, Cube, Hexagon and my Space Shuttle.

<b>QB3D_Cube</b>	Basic code to rotate a Cube.
<b>QB3D_Wire512v3</b>	Program for standard QL 512x256 screens.
<b>QB3D_Wire768v3</b>	For the SMSQ 768x512 screen and 16 bit colour.
<b>QB3D_Data03</b>	Data coding for Pyramid, Cube, Hexagon, Shuttle

### QB3D QL Platforms

The **QB3D\_Wire512** Mode 4 screen 512x256 was loaded and run with a **QemuLator** in basic QL user Mode original speed and with 128k memory. It's very slow.

**QB3D\_Wire512** has been checked with the **QL2K** set with a clock multiplier of 1000 and the **SMSQ** emulators. The best results were obtained with the **QB3D\_Wire768** version developed with the **SMSQmulator**. This takes advantage of the greater screen resolutions and 16bit colour.

### History Notes:

**Version 02a** The Roll./Spin/Loop Graphics linked to the motion of Object. Some improvements to control xyz range of angle and changes to default Increase/Decrease in **fs** and **gx, gy** variables. Additional toggle switch settings. Change of names to some Procedures.

**Version 03** Includes Solid Object Frames and added procedure **Obj\_Cull**.

### Future Thoughts:

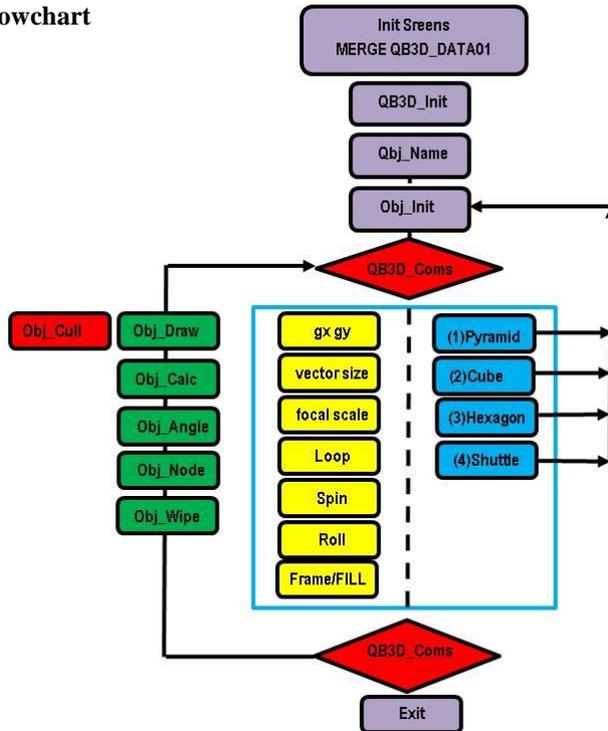
I guess a Companion Program to construct Wireframe Objects and generate their **Node, Vector** and **Frame DATA** lists would be useful.

## QB3D\_Wire Procedures

Set up of Screen windows and common variables MERGE QB3D\_Data01.

<b>QB3D_Init</b>	Sets screen layout and KEY information.
<b>Obj_Name</b>	Displays Object Names on screen menu
<b>Obj_Init</b>	Sets the DATA RESTORE references for Object.
<b>QB3D_Coms</b>	Serves as main Menu loop to access functions.
<b>gx,gy</b>	Screen global positioning
<b>vector sizes</b>	Enlarge / Reduce Object size
<b>focal scale</b>	Increase Decrease Focal distance
<b>f F</b>	Wireframe (default) <f >Solid frame <F>Surface FILL
<b>XxYyZz</b>	Loop / Spin / Roll Object
<b>B W</b>	Change Screen background (Black/White)
<b>Obj_Auto</b>	Sets auto Loop/ Spin/ Roll of Object
<b>Obj_Ang</b>	Updates and Draws Loop/ Spin/ Roll Angle Graphics.
<b>Obj_Calc</b>	Calculates new vx,vy coordinates of Object.
<b>Obj_Draw</b>	Draws Object to screen.
<b>Obj_Wipe</b>	Wipes existing Object
<b>Obj_Node</b>	Loads Node xyz and sets vector size.
<b>Obj_Cull</b>	Identifies hidden frames

## QB3D\_Wire 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

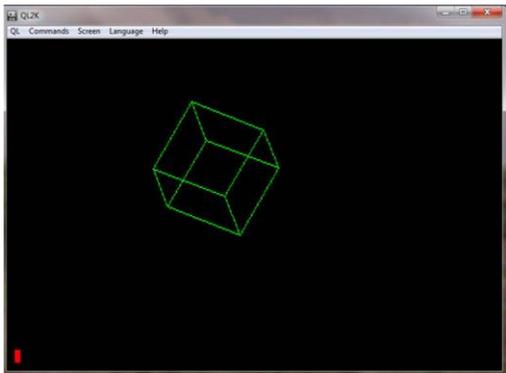
132 ra=ra+.1:c=COS(ra):s=SIN(ra)

136 FOR np=1 TO 8
138 REMark Rotation on X Axis
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 - Draws A Cube
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

```



## QBITS Exploring QL 3D Rotation Graphics

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

### QB3D\_Wire512v2a

#### QB3D\_Data02

100 REMark QB3D\_Wire512v2a (Exploring QL 3D Rotation Graphics v.02a)

102 MODE 4:OPEN#4,con\_10x10a10x10\_4

103 OPEN#3,scr\_134x108a6x100:PAPER#3,1:SCALE#3,100,0,0:CSIZE#3,0,0

104 WINDOW#2,512,256,0,0:PAPER#2,0:CLS#2:WINDOW#2,512,208,0,0

105 WINDOW#1,364,182,142,26:BORDER#1,1,2:PAPER#1,0:INK#1,7:CLS:SCALE 200,0,0

106 WINDOW#0,496,40,8,210

108 gx=140:gy=120:fs=800:vs=.8 :REMark Angle:Global xy:focal scale:vector size

109 aset=-1:cset=1:nset=2:iset=1 :REMark Toggle switches

110 bg1=0:bg2=7:k=49:wres=512 :REMark Screen settings

#### 112 MERGE flp1\_QB3D\_Data03

#### 114 QB3D\_Init:Obj\_Name:Obj\_Init:QB3D\_Coms

116 DEFine PROCedure QB3D\_Init

117 ch=1:CSIZE#ch,0,0:INK#ch,4

118 ch=2:CSIZE#ch,0,1:INK#ch,4

119 CURSOR#ch,300,6:PRINT#ch,'Exploring QL 3D Rotation Graphics'

120 CSIZE#ch,0,0:INK#ch,7

121 CURSOR#ch,6,14:PRINT#ch,'KEYS: Toggle animation On/Off <spacebar>'

122 INK#ch,4

123 CURSOR#ch,6,26:PRINT#ch,'Global x y :<←↑↓→>'

124 CURSOR#ch,6,36:PRINT#ch,'Focal Scale :<D l>'

125 CURSOR#ch,6,66:PRINT#ch,'frame/FILL :<f F>'

126 CURSOR#ch,6,46:PRINT#ch,'Node ID On/Off:<N>'

127 CURSOR#ch,6,56:PRINT#ch,'Vector Size :<E R>'

128 CURSOR#ch,6,76:PRINT#ch,'Rotation: Xx Yy Zz'

129 CURSOR#ch,6,86:PRINT#ch,'Background :<B W>'

130 ch=3:BORDER#ch,1,3:INK#ch,2:CLS#ch

131 CURSOR#ch,80,82:PRINT#ch,'Loop'

132 CIRCLE#ch,60,48,24,.25,0:LINE#ch,36,48 TO 60,48

133 INK#ch,4

134 CURSOR#ch,40,4:PRINT#ch,'Spin'

135 CIRCLE#ch,36,76,24,.25,PI/2:LINE#ch,36,48 TO 36,76

136 INK#ch,7

137 CURSOR#ch,30,94:PRINT#ch,'Roll'

138 CIRCLE#ch,26,40,24,.8,0:LINE#ch,26,40 TO 36,48

139 INK#ch,7

140 CURSOR#ch,2,2:PRINT#ch,'ANGLE'

141 END DEFine

## QBITS Exploring QL 3D Rotation Graphics

This is the **QBITS SMSQ** version with screen size 768x512 and 16bit colour.

### QB3D\_Wire768v2a

#### QB3D\_Data02

```
100 REMark QB3D_Wire768v3 (Exploring QL 3D Rotation Graphics v3)
101 :
102 MODE 4:OPEN#4,con_10x10a10x10_4
103 OPEN#3,scr_160x160a20x178:PAPER#3,0:SCALE#3,100,0,0:CSIZE#3,0,0
104 WINDOW#2,768,512,0,0:PAPER#2,0:CLS#2:WINDOW#2,754,460,8,4
105 WINDOW#1,548,428,200,32:BORDER#1,1,5:PAPER#1,7:INK#1,0:CLS#1:SCALE#1,320,0,0
106 WINDOW#0,732,40,16,460
107 :
108 gx=140:gy=120:fs=800:vs=.8 :REMark Angle:Global xy:focal scale:vector size
109 aset=-1:cset=1:nset=2:iset=1 :REMark Toggle switches
110 bg1=7:bg2=0:k=49:wres=768 :REMark Screen settings
111 :
112 MERGE flp1_QB3D_Data03
113 :
114 QB3D_Init:Obj_Name:Obj_Init:QB3D_Coms
115 :
116 DEFine PROCedure QB3D_Init
117 ch=1:CSIZE#ch,0,0:INK#ch,4
118 ch=2:CSIZE#ch,1,1:INK#ch,5:OVER#ch,1
119 FOR i=1 TO 3:CORSOR#ch,460+i,6:PRINT#ch,'Exploring QL 3D Rotation Graphics'
120 OVER#ch,0:CSIZE#ch,0,1:INK#ch,6
121 CURSOR#ch,12,6:PRINT#ch,'KEYS: Toggle animation On/Off <spacebar>'
122 INK#ch,5
123 CURSOR#ch,12,30:PRINT#ch,'Global x y : <←↑↓→>'
124 CURSOR#ch,12,50:PRINT#ch,'Focal Scale :<D |>'
125 CURSOR#ch,12,70:PRINT#ch,'frame/FILL :<F><F>'
126 CURSOR#ch,12,90:PRINT#ch,'Node ID On/Off:<N>'
127 CURSOR#ch,12,110:PRINT#ch,'Vector Size :<E R>'
128 CURSOR#ch,12,130:PRINT#ch,'Rotation: Xx Yy Zz'
129 CURSOR#ch,12,150:PRINT#ch,'Background :<B W>'
130 ch=3:BORDER#ch,1,3:INK#ch,3:CLS#ch
131 CURSOR#ch,120,130:PRINT#ch,'Loop'
132 CIRCLE#ch,60,48,24,.25,0:LINE#ch,36,48 TO 60,48
133 INK#ch,5
134 CURSOR#ch,40,8:PRINT#ch,'Spin'
135 CIRCLE#ch,36,76,24,.25,PI/2:LINE#ch,36,48 TO 36,76
136 INK#ch,6
137 CURSOR#ch,30,140:PRINT#ch,'Roll'
138 CIRCLE#ch,26,40,24,.8,0:LINE#ch,26,40 TO 36,48
139 INK#ch,7
140 CURSOR#ch,2,2:PRINT#ch,'ANGLE'
141 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 increased resolution as is use of the 8 colour palette available.

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

```

143 DEFine PROCEDURE QB3D_Coms
144 REPEAT com
145 SElect ON k
146 =27 :CLOSE#3,#4:INK#2,7:STOP
147 =66,98 :bg1=0:bg2=7:PAPER#1,0:CLS#1 :REMark (B)lack background
148 =87,119:bg1=7:bg2=0:PAPER#1,7:CLS#1 :REMark (W)hite background
149 =49,50,51,52,53 :iset=1:Obj_Ang:Obj_Init :REMark Load Object DATA
150 =32 :IF aset=-1:aset=5:ELSE aset=-1 :REMark Toggle animation
151 =102 :IF cset= 1 OR cset=3:cset=2:ELSE cset= 1 :REMark (f)rame On/Off
152 =70 :IF cset= 1 OR cset=2:cset=3:ELSE cset= 1 :REMark (F)ILL On/Off
153 =78,110:IF nset= 1:nset=2:ELSE nset=1 :REMark (N)ode ID On/Off
154 =69,101:vs=vs+.1 :IF vs>=1.5 :vs=1.5 :REMark (E)xpand Vector size
155 =82,114:vs=vs-.1 :IF vs<= .5 :vs= .5 :REMark (R)educe Vector size
156 =68,100:fs=fs-10 :IF fs< 80 :fs= 80 :REMark (D)ecrease Focal scale
157 =73,105:fs=fs+10 :IF fs>800 :fs=800 :REMark (I)ncrease Focal scale
158 =192 :gx=gx-10 :IF gx<= 20 :gx= 20 :REMark ← move left
159 =200 :gx=gx+10 :IF gx>=280 :gx=280 :REMark → move right
160 =208 :gy=gy+10 :IF gy>=190 :gy=190 :REMark ↑ move up
161 =216 :gy=gy-10 :IF gy<= 10 :gy= 10 :REMark ↓ move down
162 = 88 :iset=1:Obj_Ang:rx=rx-5: :IF rx< 0:rx=rx+360 :REMark (X) Clockwise Loop
163 =120 :iset=1:Obj_Ang:rx=rx+5 :IF rx>360:rx=rx-360 :REMark (x) Anti- Loop
164 = 89 :iset=1:Obj_Ang:ry=ry-5 :IF ry< 0:ry=ry+360 :REMark (Y) Clockwise Spin
165 =121 :iset=1:Obj_Ang:ry=ry+5 :IF ry>360:ry=0 :REMark (y) Anti- Spin
166 = 90 :iset=1:Obj_Ang:rz=rz-5 :IF rz< 0:rz=rz+360 :REMark (z) Clockwise Roll
167 =122 :iset=1:Obj_Ang:rz=rz+5 :IF rz>360:rz=rz-360 :REMark (Z) Anti- Roll
168 END SElect
169 Obj_Wipe:Obj_Draw:INK bg2
170 IF aset=5:iset=1:Obj_Auto:ELSE Obj_Ang
171 CURSOR 168,168:PRINT 'gx:&gx&' gy:&gy&' vs:&(vs*20)&' fs:&fs&'
172 CURSOR 12,168:PRINT 'rx:&rx&' ry:&ry&' rz:&rz&'
173 k=CODE(INKEY$(#4,aset))
174 END REPEAT com
175 END DEFine

```

**Note:** QB3D\_Wire768 higher screen resolution.

```

160 :IF gy>=300:gy=300
161 :IF gy<= 20:gy= 20
171 CURSOR 320,410
172 CURSOR 20,410

```

**Note:** Global **gx,gy**, vector size **vs** and focal scale **fs** are displayed on screen.  
 On start up a Pyramid is displayed.  
 Selection of (1), (2), (3), (4) displays the selected Object in static mode.  
 To select background press Keys (B)lack or (W)hite  
 Press <N> to toggle Node ID On/Off  
 Press <F> Wireframe to Solid frame <F> FILL's Objects surfaces with colours  
 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.

## QBITS Exploring QL 3D Rotation Graphics

The Procedures to create the 3D Rotation Graphics.

### 177 **DEFine PROCEDURE Obj\_Auto**

```
178 rx=rx+5:IF rx>=360:rx=0
179 ry=ry+5:IF ry>=360:ry=0
180 rz=rz+5:IF rz>=360:rz=0
181 END DEFine
```

### 183 **DEFine PROCEDURE Obj\_Ang**

```
184 ch=3
185 IF iset=1:zink=0:yink=0:xink=0:ELSE zink=7:yink=4:xink=2
186 INK#ch,zink:LINE#ch,26,40 TO 26+19*COS(RAD(rz)),40+21*SIN(RAD(rz))
187 INK#ch,yink:LINE#ch,36,76 TO 36+21*COS(RAD(ry)),76 +5*SIN(RAD(ry))
188 INK#ch,xink:LINE#ch,60,48 TO 60+ 5*COS(RAD(rx)),48+21*SIN(RAD(rx))
189 ch=1
190 END DEFine
```

### 192 **DEFine PROCEDURE Obj\_Calc**

```
193 cx=COS(RAD(rx)):sx=SIN(RAD(rx))
194 cy=COS(RAD(ry)):sy=SIN(RAD(ry))
195 cz=COS(RAD(rz)):sz=SIN(RAD(rz))
196 FOR np=1 TO no
197 yt=y(np):y(np)=cx*yt-sx*z(np):z(np)=sx*yt+cx*z(np)
198 xt=x(np):x(np)=cy*xt+sy*z(np):z(np)=sy*xt+cy*z(np)
199 xt=x(np):x(np)=cz*xt-sz*y(np):y(np)=sz*xt+cz*y(np)
200 vx(np)=gx+(x(np)*fs)/(z(np)+fs)
201 vy(np)=gy+(y(np)*fs)/(z(np)+fs)
202 END FOR np
203 END DEFine
```

### 205 **DEFine PROCEDURE Obj\_Draw**

```
206 LOCAL lp,v,a,b,c,d,i:RESTORE vres:READ v
207 iset=2:Obj_Calc
208 FOR lp=1 TO v
209 READ a,b,c,d,i
210 IF cset=1:INK bg2:FILL 0:END IF
211 IF cset=2:Obj_Cull:IF c1>0:GO TO 214:END IF :INK bg2:FILL 0:END IF
212 IF cset=3:Obj_Cull:IF c1>0:GO TO 214:END IF :INK i :FILL 1:END IF
213 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
214 END FOR lp
215 Obj_Node:IF nset=2:INK bg2:FOR n=1 TO no:CURSOR vx(n),vy(n),-2,2:PRINT n
216 END DEFine
```

### 218 **DEFine PROCEDURE Obj\_Wipe**

```
219 LOCAL lp,v,a,b,c,d,i:RESTORE vres:READ v:INK bg1
220 FOR lp=1 TO v
221 READ a,b,c,d,i:FILL 1
222 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
223 END FOR lp
224 Obj_Node:FOR n=1 TO no:CURSOR vx(n),vy(n),-2,2:PRINT n
225 END DEFine
```

## QBITS Exploring QL 3D Rotation Graphics

```
227 DEFine PROCEDURE Obj_Node
228 LOCAL lp,a,b,c:RESTORE nres:READ no
229 FOR lp=1 TO no
230 READ a,b,c:x(lp)=a*vs:y(lp)=b*vs:z(lp)=c*vs
231 END FOR lp
232 END DEFine
```

```
234 DEFine PROCEDURE Obj_Cull
235 c1=(x(b)-x(a))*(y(c)-y(a))-(x(c)-x(a))*(y(b)-y(a))
236 END DEFine
```

Below is the **DATA** file **MERGE**'d with the **QB3D\_Wire (512 or 768)**

```
400 REMark QB3D_Data03 (Pyramid Cube Hexagon Shuttle)
```

```
402 DEFine PROCEDURE Obj_Name
403 IF wres=512:CURSOR#0,12,4:PRINT#0,' (1)Pyramid (2)Cube (3)Hexagon (4)Shuttle'
404 IF wres=768
405 CSIZE#2,0,1:INK#2,6
406 CURSOR#2,12,338:PRINT#2,'(1)Pyramid'
407 CURSOR#2,12,358:PRINT#2,'(2)Cube'
408 CURSOR#2,12,378:PRINT#2,'(3)Hexagon'
409 CURSOR#2,12,398:PRINT#2,'(4)Shuttle'
410 CURSOR#2,12,418:PRINT#2,"
411 END IF
412 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)

```
414 DEFine PROCEDURE Obj_Init
415 REMark WARNING maintain correct nres:vres:fres numbers
416 IF k=49:nres=426:vres=433:rx=60:ry=30:rz= 0
417 IF k=50:nres=441:vres=451:rx=30:ry=30:rz= 0
418 IF k=51:nres=460:vres=474:rx= 15:ry= 60:rz= 60
419 IF k=52:nres=487:vres=511:rx= 30:ry= 45 :rz= 15
420 REMark IF k=53:nres=600:vres=600:rx= .2:ry= .2 :rz= .2
421 RESTORE nres:READ n:DIM x(n),y(n),z(n),vx(n),vy(n)
422 RESTORE vres:READ v:DIM fr(v,5):CLS#1
423 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**.

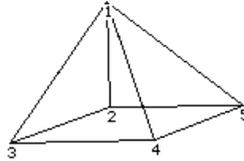
**Note:**

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.

# QBITS Exploring QL 3D Rotation Graphics

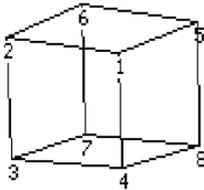
## 425 REMark Pyramid

426 DATA 5 :REMark Nodes  
 427 DATA 0, 20,-20 :REMark Node 1  
 428 DATA 20, 20, 20  
 429 DATA 20,-20, 20  
 430 DATA -20,-20, 20  
 431 DATA -20, 20, 20



433 DATA 5 :REMark Frames

434 DATA 1,2,3,3,2  
 435 DATA 1,3,4,1,4  
 436 DATA 1,4,5,1,3  
 437 DATA 1,5,2,1,5  
 438 DATA 5,4,3,2,bg2



## 440 REMark Cube

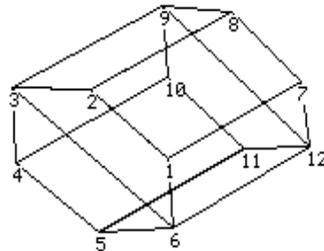
441 DATA 8 :REMark Nodes  
 442 DATA -20, 20,-20 :REMark Node 1  
 443 DATA -20,-20,-20  
 444 DATA 20,-20,-20  
 445 DATA 20, 20,-20 :REMark Node 4  
 446 DATA -20, 20, 20 :REMark Node 5  
 447 DATA -20,-20, 20  
 448 DATA 20,-20, 20  
 449 DATA 20, 20, 20 :REMark Node 8

450 :  
 451 DATA 6 :REMark Frames  
 452 DATA 8,7,6,5,bg2 :REMark back Frame  
 453 DATA 2,6,7,3,2  
 454 DATA 4,3,7,8,4  
 455 DATA 5,1,4,8,3  
 482 DATA 5,6,2,1,5  
 484 DATA 1,2,3,4,bg2 :REMark front Frame

## 488 REMark Hexagon

460 DATA 12 :REMark Nodes  
 461 DATA 30, 0,-20 :REMark Node 1  
 462 DATA 15, 20,-20  
 463 DATA -15, 20,-20  
 464 DATA -30, 0,-20  
 465 DATA -15,-20,-20  
 466 DATA 15,-20,-20 :REMark Node 6  
 467 DATA 30, 0, 20 :REMark Node 7  
 468 DATA 15, 20, 20  
 469 DATA -15, 20, 20  
 470 DATA -30, 0, 20  
 471 DATA -15,-20, 20  
 472 DATA 15,-20, 20 :REMark node 12

473 :  
 474 DATA 10 :REMark Frames  
 475 DATA 7,8,9,12,5 :REMark rear frames  
 476 DATA 9,10,11,12,3  
 477 DATA 4,5,11,10,5 :REMark side frames  
 478 DATA 3,4,10,9,2  
 479 DATA 2,3,9,8,4  
 480 DATA 1,2,8,7,3  
 481 DATA 6,1,7,12,4  
 482 DATA 5,6,12,11,2  
 483 DATA 6,3,2,1,5 :REMark front frames  
 484 DATA 5,4,3,6,3



# QBITS Exploring QL 3D Rotation Graphics

## 486 REMark **Space Shuttle**

```

487 DATA 22 :REMark Nodes
488 DATA -40, 0, 20 :REMark Node 1
489 DATA -20,-20, 20 :REMark Node 2
490 DATA -20, 0, 30
491 DATA -20, 20, 20
492 DATA -20, 20,-20
493 DATA -20, 0,-30
494 DATA -20,-20,-20 :REMark Node 7
495 DATA -40, 0,-20 :REMark Node 8
496 DATA 40,-20, 20 :REMark Node 9
497 DATA 40, 0, 30
498 DATA 40, 20, 20
499 DATA 40, 20,-20
500 DATA 40, 0,-30
501 DATA 40,-20,-20 :REMark Node 14
502 DATA -24, 14, 16 :REMark Node 15
503 DATA -30, 8, 14
504 DATA -30, 8,-14
505 DATA -24, 14,-16 :REMark Node 18
506 DATA 40,-16, 16 :REMark Node 19
507 DATA 40, 16, 16
508 DATA 40, 16,-16
509 DATA 40,-16,-16 :REMark Node 22

511 DATA 16 :REMark Frames
512 DATA 9,10,13,14,243 :REMark rear Frames
513 DATA 10,11,12,13,240
514 DATA 19,20,21,22,1911 :REMark Rear Door
515 DATA 2,9,14,7,243 :REMark Side Frames
516 DATA 6,7,14,13,243
517 DATA 5,6,13,12,240
518 DATA 5,12,11,4,,240
519 DATA 4,11,10,3,240
520 DATA 3,10,9,2,243
521 DATA 3,2,1,3,243 :REMark Front Frames
522 DATA 1,2,7,8,243
523 DATA 7,6,8,7,243
524 DATA 8,6,5,8,240
525 DATA 4,1,8,5,240
526 DATA 1,4,3,1,240
527 DATA 15,16,17,18,0 :REMark Pilot Window

```

