# OpenGL 3D Graphics in Liberty BASIC

## Lesson Six: Creating Shapes

*by Robert McAllister*

As mentioned in lesson 1, OpenGL only has functions for making a few basic surfaces. This leaves it up to the programer to build the shapes we need using these available functions. Luckily Liberty BASIC makes this a little less painful for us with subroutines.

But first, read this quote from the OpenGL Redbook:
*A normal vector (or normal, for short) is a vector that points in a direction that's perpendicular to a surface. For a flat surface, one perpendicular direction suffices for every point on the surface, but for a general curved surface, the normal direction might be different at each point. With OpenGL, you can specify a normal for each vertex. Vertices might share the same normal, but you can't assign normals anywhere other than at the vertices.*

*An object's normal vectors define the orientation of its surface in space - in particular, its orientation relative to light sources. These vectors are used by OpenGL to determine how much light the object receives at its vertices. Lighting - a large topic by itself - is the subject of Chapter 6, and you might want to review the following information after you've read that chapter. Normal vectors are discussed briefly here because you generally define normal vectors for an object at the same time you define the object's geometry.*

*You use glNormal*() to set the current normal to the value of the argument passed in. Subsequent calls to glVertex*() cause the specified vertices to be assigned the current normal...*

As this quote mentions, lighting is a large topic so I'm not going to dig to far into it. In the [Initialize] branch of the program you will find these two lines of code:

```
CALL glEnable GL.LIGHTING
CALL glEnable GL.LIGHT0
```

The first line enables lighting in general and the second line adds a light to the scene. Unless you plan on doing some special effects, these two lines along with 'Normal' calls are all you need to light up your scene.

This bit of code demostrates how to create a cube with a call to a subroutine, and how the sub applies the normals to properly light it up. Run it as is and you will notice that most of the cube appears black. Un-comment the 6 "CALL glNormal..." lines and try it again. (Note: If you receive a "Cannot call undefined sub" error, you will need to download the latest version of the program from page one. I forgot to include the glNormal sub in the original version.)

```
    ' build a cube and spin it
    CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
    Width = 1.5
    Height = 1.5
    Depth = 1.5
    CubeCenterX = 0
    CubeCenterY = 0
    CubeCenterZ = 0
    Red = .2
    Green = .1
    Blue = .1
    CALL glGenLists 1
    CALL glNewList 1 , 4865
    CALL
 BuildCube Width , Height , Depth , CubeCenterX , CubeCenterY , CubeCe
nterZ , Red , Green , Blue
    CALL glEndList

    FOR a  = 1 TO 360
      CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
      CALL glRotatef a  , 1 , 0 , 0
      'CALL glRotatef a  , 0 , 1 , 0
      CALL glRotatef a  , 0 , 0 , 1
      CALL glCallList 1

      CALL RefreshView
      CALL Pause 15
    NEXT a

    WAIT


SUB BuildCube W , H , D , cX , cY , cZ , r , g , b

    ' build a cube and spin it
    CALL glColor4fv r , g , b , 1
    GL.QUADS=7
    'front
    CALL glBegin GL.QUADS

'CALL glNormal cX-(W/2),cY-(H/2),cZ+(D/2) , cX-(W/2),cY+(H/2),cZ+(D/2)
 , cX+(W/2),cY+(H/2),cZ+(D/2)
```

```
        CALL glVertex cX-(W/2) , cY+(H/2) , cZ+(D/2)
        CALL glVertex cX+(W/2) , cY+(H/2) , cZ+(D/2)
        CALL glVertex cX+(W/2) , cY-(H/2) , cZ+(D/2)
        CALL glVertex cX-(W/2) , cY-(H/2) , cZ+(D/2)
    CALL glEnd
    'back
    CALL glBegin GL.QUADS

'CALL glNormal cX+(W/2),cY+(H/2),cZ-(D/2) , cX-(W/2),cY+(H/2),cZ-(D/2)
 , cX-(W/2),cY-(H/2),cZ-(D/2)
        CALL glVertex cX+(W/2) , cY+(H/2) , cZ-(D/2)
        CALL glVertex cX-(W/2) , cY+(H/2) , cZ-(D/2)
        CALL glVertex cX-(W/2) , cY-(H/2) , cZ-(D/2)
        CALL glVertex cX+(W/2) , cY-(H/2) , cZ-(D/2)
    CALL glEnd
    'left
    CALL glBegin GL.QUADS

'CALL glNormal cX-(W/2),cY+(H/2),cZ-(D/2) , cX-(W/2),cY+(H/2),cZ+(D/2)
 , cX-(W/2),cY-(H/2),cZ+(D/2)
        CALL glVertex cX-(W/2) , cY+(H/2) , cZ-(D/2)
        CALL glVertex cX-(W/2) , cY+(H/2) , cZ+(D/2)
        CALL glVertex cX-(W/2) , cY-(H/2) , cZ+(D/2)
        CALL glVertex cX-(W/2) , cY-(H/2) , cZ-(D/2)
    CALL glEnd
    'right
    CALL glBegin GL.QUADS

'CALL glNormal cX+(W/2),cY+(H/2),cZ+(D/2) , cX+(W/2),cY+(H/2),cZ-(D/2)
 , cX+(W/2),cY-(H/2),cZ-(D/2)
        CALL glVertex cX+(W/2) , cY+(H/2) , cZ+(D/2)
        CALL glVertex cX+(W/2) , cY+(H/2) , cZ-(D/2)
        CALL glVertex cX+(W/2) , cY-(H/2) , cZ-(D/2)
        CALL glVertex cX+(W/2) , cY-(H/2) , cZ+(D/2)
    CALL glEnd
    'top
    CALL glBegin GL.QUADS

'CALL glNormal cX-(W/2),cY+(H/2),cZ+(D/2) , cX-(W/2),cY+(H/2),cZ-(D/2)
 , cX+(W/2),cY+(H/2),cZ-(D/2)
        CALL glVertex cX-(W/2) , cY+(H/2) , cZ+(D/2)
        CALL glVertex cX-(W/2) , cY+(H/2) , cZ-(D/2)
        CALL glVertex cX+(W/2) , cY+(H/2) , cZ-(D/2)
        CALL glVertex cX+(W/2) , cY+(H/2) , cZ+(D/2)
    CALL glEnd
    'bottom
```

```
    CALL glBegin GL.QUADS

'CALL glNormal cX-(W/2),cY-(H/2),cZ+(D/2) , cX+(W/2),cY-(H/2),cZ+(D/2)
 , cX+(W/2),cY-(H/2),cZ-(D/2)
        CALL glVertex cX-(W/2) , cY-(H/2) , cZ+(D/2)
        CALL glVertex cX+(W/2) , cY-(H/2) , cZ+(D/2)
        CALL glVertex cX+(W/2) , cY-(H/2) , cZ-(D/2)
        CALL glVertex cX-(W/2) , cY-(H/2) , cZ-(D/2)
    CALL glEnd

END SUB
```

In the [Initialize] branch you will find "CALL glEnable GL.NORMALIZE". Normals need to have a length of 1 to work properly. In other words, they should be 1 unit above the surface that is being created. With "GL.NORMALIZE" enabled, OpenGL automatically corrects their length.

The sub "glNormal" takes 3 consecutive clockwise (from the front) vertices for the surface, calculates the normal vector and then makes the call to "glNormal3d". To keep it simple, I always create my surfaces in a clockwise direction and then use the values from the first 3 glVertex calls.

This code will create a cylinder or oval, depending on the sizes entered. Can you figure out how to add a top and bottom to the cylinder using triangles? Hint: all the values are already being calculated for you.

```
    ' build a cylinder
    CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
    Angle = 0
    Width = 1
    Depth = .75
    Height = 1
    OvalCenterX = 0
    OvalCenterY = 0
    OvalCenterZ = 0
    Red = .5
    Green = 0
    Blue = 0
    Sides = 120

    CALL glGenLists 1
    CALL glNewList 1 , 4865
    CALL
 BuildCylinder Angle , Width , Depth , Height , OvalCenterX , OvalCent
```

```
erY , OvalCenterZ , Red , Green , Blue , Sides
    CALL glEndList

    FOR a  = 1 TO 360
      CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
      CALL glRotatef a  , 1 , 0 , 0
      CALL glRotatef a  , 0 , 1 , 0
      'CALL glRotatef a  , 0 , 0 , 1
      CALL glCallList 1

      CALL RefreshView
      CALL Pause 15
    NEXT a


    WAIT


SUB BuildCylinder Angle , W , D , H , cX , cY , cZ , R , G , B , Sides
    CALL glColor4fv R , G , B , 1
    GL.QUADS = 7
    GL.TRIANGLES = 4
    PI = 3.14159265

    sin.angle = Sin(Angle * PI / 180)
    cos.angle = Cos(Angle * PI / 180)

    theta = 0
    dtheta = 2 * PI / Sides

    XOval = W * Cos(theta)
    YOval = D * Sin(theta)
    X1 = cX + XOval * cos.angle + YOval * sin.angle
    Z1 = cZ - XOval * sin.angle + YOval * cos.angle

    While theta < 2 * PI

        theta = theta + dtheta

        XOval = W * Cos(theta)
        YOval = D * Sin(theta)

        X2 = cX + XOval * cos.angle + YOval * sin.angle
        Z2 = cZ - XOval * sin.angle + YOval * cos.angle
```

```
        CALL glBegin GL.QUADS
          CALL glNormal X1 , cY+(H/2) , Z1 , X1 , cY-(H/2)
 , Z1 , X2 , cY-(H/2) , Z2
          CALL glVertex X1 , cY-(H/2) , Z1

          CALL glNormal X1 , cY-(H/2) , Z1 , X2 , cY-(H/2)
 , Z2 , X2 , cY+(H/2) , Z2
          CALL glVertex X2 , cY-(H/2) , Z2

          CALL glNormal X2 , cY-(H/2) , Z2 , X2 , cY+(H/2)
 , Z2 , X1 , cY+(H/2) , Z1
          CALL glVertex X2 , cY+(H/2) , Z2

          CALL glNormal X2 , cY+(H/2) , Z2 , X1 , cY+(H/2)
 , Z1 , X1 , cY-(H/2) , Z1
          CALL glVertex X1 , cY+(H/2) , Z1
        CALL glEnd

        X1 = X2
        Z1 = Z2

    WEND
END SUB
```

In the next lesson we will learn about "Texture mapping"