Easy Functions for Plotting 3D Objects

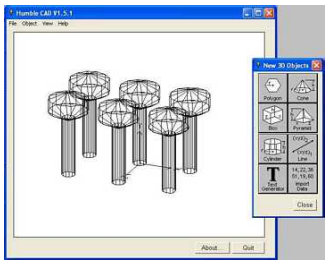
Tomas J. Nally -
steelweaver52

This preliminary article originally appeared in the Liberty BASIC Newsletter, Issue #113. *This article is not technically a part of Tom's* Liberty BASIC Wire Frame Library, *but this article did serve as the impetus for* LBWF Library. *Because this article is referenced in the* LBWF Library *series, it was thought best to include this article as part of the series.* ***Easy Functions for Plotting 3D Objects*** *is reprinted here with the permission of the author.*

View the source code for the functions described in this article!

Making Humble CAD Technology Usable

In late 2002 I published version 1.0 of **Humble CAD**, which is a Liberty Basic program for drawing "wire model images" of 3D objects. The reaction was favorable among Liberty Basic users, and I published versions 1.5 and 1.5.1 in early 2003. The latter version is available for download at two locations: LB Downloads, and the files area of the Liberty BASIC Yahoo group (HumCAD151.zip). All versions of Humble CAD have been released as open source.



Screenshot: HumbleCAD v1.5.1

But, just because the source code is "open" doesn't mean that the code is easy to extract and decipher for use in one's own programs. In fact, the most interesting part of Humble CAD's code--the vector math that converts all of the 3D data into *ready-to-plot* 2D format--is a little too disorganized and complex to be extracted for use in someone else's programs. The open source advantage, then, is wasted. Humble CAD contains good technology, but the accessibility to this technology is poor.

This article and demo attempts to correct this lack of accessibility to Humble CAD's code. Specifically, two functions are provided with this article--**ScreenX()** and **ScreenY()** (source code availabe here) --which accept coordinates of points in 3D space, and return the equivalent 2D screen coordinates of the same point.

Please be aware that numerous arguements must be passed to these functions in order for them to work properly. As you read below, however, I'm sure you will appreciate the logic and necessity of this.

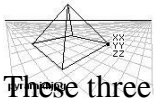
The Arguments Required by ScreenX() and ScreenY()

ScreenX() and **ScreenY()** each require twelve arguments. In fact, both functions require the *exact same arguments*! The only difference between the functions is that one returns the X-coordinate of the 3D point's projection onto the *LB GRAPHICBOX*, while the other returns the Y-coordinate. The only reason why two functions are required is because a function only returns a single value. If we could persuade a function to return two values, then this operation could be accomplished with a single function.

Without further ado, here is a list of the twelve arguments required by **ScreenX()** and **ScreenY()**. It will be followed by a discussion of the importance of these arguments.

- 1. XX - The X-coordinate of the node in space.
- 2. YY - The Y-coordinate of the node in space.
- 3. ZZ - The Z-coordinate of the node in space.
- 4. CamX - (Think "Camera-X") The X-coordinate of the camera.
- 5. CamY - (Think "Camera-Y") The Y-coordinate of the camera.
- 6. CamZ - (Think "Camera-Z") The Z-coordinate of the camera.
- 7. CtrX - (Think "Center-X") The X-coordinate of the point in space where your camera is pointed. I often call this point the "Viewing Center".
- 8. CtrY - (Think "Center-Y") The Y-coordinate of the point in space where your camera is pointed.
- 9. CtrZ - (Think "Center-Z") The Z-coordinate of the point in space where your camera is pointed.
- 10. ScrCtrX - (Think "Screen Center-X") The X-coordinate of the GRAPHICBOX point which corresponds to the Viewing Center.
- 11. ScrCtrY - (Think "Screen Center-Y") The Y-coordinate of the GRAPHICBOX point which corresponds to the Viewing Center.
- 12. Scale - A value that allows you to apply "size control" to the plotted objects in the LB GRAPHICBOX.

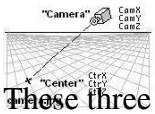
Arguments XX, YY and ZZ



Triangle

These three variables are the X-, Y-, and Z-coordinates of the ***node*** in space that we wish to plot. Understand, though, that plotting a single ***node*** by itself has little or no value. In this particular demo, the importance of a ***node*** is that it serves as one of the two endpoints of a ***line***. The value of a ***line*** is that we can arrange numerous ***lines*** in space to make representations of objects. For instance, we can arrange 8 ***lines*** in space to form a pyramid. Four ***lines*** would form the square base of the pyramid, while the remaining for four ***lines*** would extend from the corners of the base up to the apex of the pyramid.

Arguments CamX, CamY and CamZ



These three variables are the X-, Y-, and Z-coordinates of the **camera location**. The camera location is the point in space where we view our objects. You can also think of it as the location of your **eye**. On other occasions, I've named these same variables **Xeye**, **Yeye** and **Zeye**. However, it looks like the convention is to refer to this point as the camera location rather than your eye location.

Arguments CtrX, CtrY and CtrZ

These three variables are the X-, Y-, and Z-coordinates of the point in space where you are **pointing the camera**. I typically call this point the **viewing center**, or just the **center**. Often, though not always, you may want to point the camera at the geometric center of your collection of objects.

Arguments ScrCtrX and ScrCtrY

These two variables are the X- and Y-coordinates of that point in the `LB GRAPHICBOX` which corresponds to the **viewing center**. For instance, if you want your objects to appear centered in the `GRAPHICBOX`, then **ScrCtrX** and **ScrCtrY** should be the ***centerpoint*** of the `GRAPHICBOX`. That is, if your `GRAPHICBOX` is 400 x 400, then

```
ScrCtrX = 200 and ScrCtrY = 200
```

Argument Scale

The usefulness of **ScreenX()** and **ScreenY()** would be limited if it did not allow you to plot objects that were very large in scale (say thousands of units long or tall) or very small in scale (less than a unit long or tall). The ***Scale*** variable gives you "size control" over the plot. Of course, you must experiment with the ***Scale*** value until your plots produce objects that are properly sized. ***Scale*** can also be used to `zoom in` and `zoom out` when plotting objects.

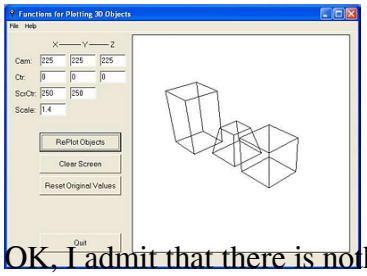
OK, So What Does the Function Look Like When Used?

Without listing the code of the functions themselves (source code available here), the example below is just about the simplist implementation of **ScreenX()** and **ScreenY()** than can be developed. (Please note that code to setup the `WINDOW` and the `GRAPHICBOX` control is omitted.) In this little code snippet, the

coordinates of two points in space are defined, followed by all the other input arguments of the functions. **ScreenX()** and **ScreenY()** are each called twice: once for the first ***node*** in space, once for the second ***node*** in space. Each time **ScreenX()** or **ScreenY()** is called, the function returns a screen coordinate. At the end of the snippet, a ***line*** is plotted using the screen coordinates returned by **ScreenX()** and **ScreenY()**.

```
XX1 = 5          '
YY1 = 5          '
ZZ1 = 5          'Coordinates of the two
XX2 = 100        'nodes in space
YY2 = 100        '
ZZ2 = 100        '

CamX = 300       '
CamY = 300       'Coordinates of the camera in space
CamZ = 300       '

CtrX = 5         'The "center",
CtrY = 5         'Coordinates of the point where the
CtrZ = 5         'camera is pointing

ScrCtrX = 200    'Coordinates of the GRAPHICBOX point
ScrCtrY = 200    'corresponding to the "center"

Scale = 1        'Define the Scale Factor

'Send all of the arguments to the two functions for the FIRST node.
'sx1 and sy1 become the screen coordinates of Node 1.

sx1 = ScreenX(XX1, YY1, ZZ1, CamX, CamY, CamZ, CtrX, CtrY, CtrZ, ScrCt
rX, ScrCtrY, Scale)
sy1 = ScreenY(XX1, YY1, ZZ1, CamX, CamY, CamZ, CtrX, CtrY, CtrZ, ScrCt
rX, ScrCtrY, Scale)

'Send all of the arguments to the two functions for the SECOND node.
'sx2 and sy2 become the screen coordinates of Node 2.

sx2 = ScreenX(XX2, YY2, ZZ2, CamX, CamY, CamZ, CtrX, CtrY, CtrZ, ScrCt
rX, ScrCtrY, Scale)
sy2 = ScreenY(XX2, YY2, ZZ2, CamX, CamY, CamZ, CtrX, CtrY, CtrZ, ScrCt
rX, ScrCtrY, Scale)

'Draw a line in the GRAPHICBOX between (sx1,sy1) and (sx2,sy2)

print #main.gbox1, "line "; sx1; " "; sy1; " "; sx2; " "; sy2
```

Demonstration Program: "Plot3D.bas"



OK, I admit that there is nothing impressive about plotting a single ***line*** in a `GRAPHICBOX`. As I indicated elsewhere, the significance of projecting ***nodes*** in 3D space onto a 2D plane is that ***nodes*** help define ***lines***, and ***lines*** help define ***objects***.

To help illustrate the usefulness of **ScreenX()** and **ScreenY()**, I'm providing a Liberty Basic program called Plot3D.bas. This program provides 3 pre-built objects. The objects are built from 24 ***nodes*** and 36 ***lines***. The window of the application provides a number of textboxes which will allow you to change the coordinates of the camera, the coordinates of the "center", the coordinates of the "screen center" and the scale. Begin experimenting by changing these values, one at a time, by small amounts. Then press the button that says `RePlot Objects`. Note how the image changes.

Limitations of ScreenX() and ScreenY()

ScreenX() and **ScreenY()** will not produce good output if the user places the camera in the midst of the objects (or inside of an object) which we are viewing. This is because **ScreenX()** and **ScreenY()** are not sophisticated enough to know which ***nodes*** are "behind" the camera and therefore shouldn't be plotted. If you wish to try it, start **Plot3D.bas** and place (CamX, CamY, CamZ) at (-5, 10, 5). Nothing will break, it will only produce bizarre output.

That limitation aside, feel free to incorporate **ScreenX()** and **ScreenY()** into your own programs. Like ***Humble CAD***, this code is provided as open source.

View the source code for the functions described in this article!

Tom Nally
Steelweaver52@aol.com
