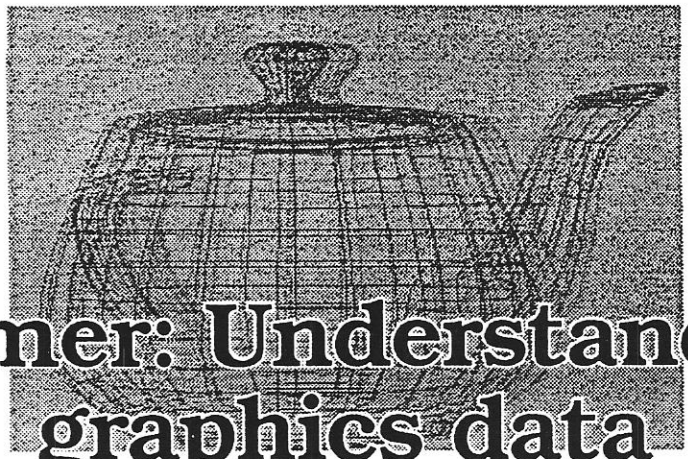


3-D Primer: Understanding 3D graphics data

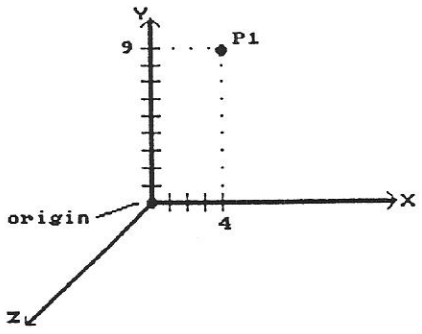
'I t'ought I taw a Utah teapot'

By Dana Dominiak

Undoubtedly you've seen incredible raytraced pictures and animations created on Amigas. The objects in these pictures, such as cubes, dragons, pyramids, and teapots galore, look surprisingly like the real things, complete with perspective views, rotations, shadows, highlights, reflections, and many other properties that make these objects look like you could reach out and touch them. Let's take a look at how these objects are represented inside the computer, and how they are stored on disk, as a help toward understanding the way Amigas deal with 3D information.

You're probably aware that all information is represented in the computer numerically; pictures, music, speech, animations, text, you name it. How can an object, such as the infamous teapot, be described merely by numbers? Moreover, how can a computer use these numbers to render a picture on the screen?

Think back to that dreaded math class of many years back. 'Member the good ole coordinate system `x-y-z` axes? If not, look at *Figure 1*. Recall that the point where all three coordinate axes meet is known as the origin, or *point* `0, 0, 0`. In addition, remember that a point, such as at the endpoint of a line, is called a *vertex*. Given a point's `x-y-z` coordinates in this system, that point's location may easily be found. For example, the point `P1 = (2, 10, 0)` in *Figure 1* lies two units to the right in the `x`-direction, 10 units up in the `y`-direction, and zero units in the `z`-direction. Points may also lie in the negative `x`, `y`, or `z` direction as can be seen in *Figure 2*. Look at point `P2 = (-5, 5, 10)`. Notice that `x = -5`. This `x` value actually lies on the opposite side of the origin along the `x`-axis. Surprisingly, that's about all you have to know about coordinate systems to understand how 3D graphics data is stored.



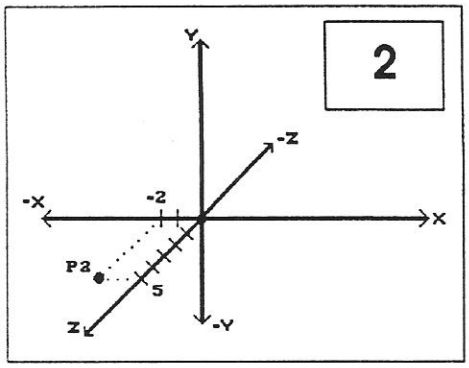
Now let's look at how to represent a geometric entity slightly more complex than a point: a line. A line can simply be described by its two endpoints, as can be seen in *Figure 3*. The line shown has endpoints `P1` and `P2`. Simple 'nuff. Now that points and lines can be described in a coordinate system, we're ready to talk about the geometric shape that ultimately represents the 3D teapots and dragons: the *polygon*. A polygon is made up of a bunch of lines. Look at *Figure 4a*, and witness the birth of several polygons. Notice how the polygons are composed of lines, and the lines are composed of endpoints and the endpoints are composed of `x-y-z` coordinates. So if we can store a multitude of the appropriate `x-y-z` coordinates, we can describe the desired 3D object.

Well, it's *almost* that simple. Keeping a list of all the `x-y-z` points is no problem, but how should we keep track of the polygons? There are a couple very popular methods for describing polygons.

The first method is called a *jump list*. Groups of points are stored in a file or memory, and they are separated by a blank line or a `'j'`, which represents a `'jump'`. Everything between two jumps represents one polygon. For example, to represent the cube in *Figure 5*, the following jump list may be used:

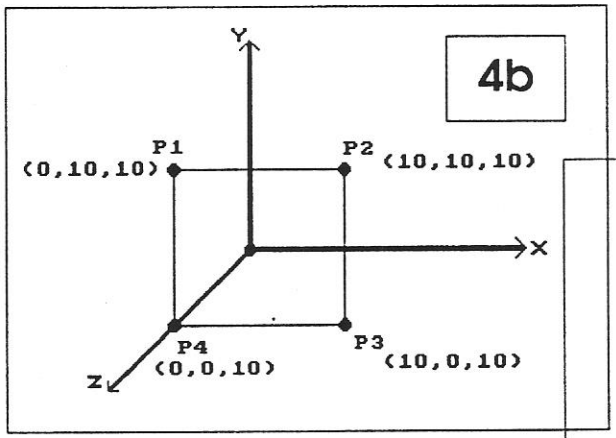
```
j
0     0     0   /* the back face
                    of the cube
0    10     0
10   10     0
10    0     0
0     0     0
j
0     0    10   /* the front face
0    10    10
10   10    10
10    0    10
0     0    10
j
0    10     0   /* the top face
0    10    10
10   10    10
10   10     0
0    10     0
j
0     0     0   /* the bottom face
0     0    10
10    0    10
10    0     0
0     0     0
j
0     0     0   /* the left face
0     0    10
0    10    10
0    10     0
0     0     0
j
10    0     0   /* the right face
10    0    10
10   10    10
10   10     0
10    0     0
j
```

Notice that the lists of points between the jumps are polygons which are simply square, each of which represents one of the six faces on a cube. The cube is rendered by starting at point `(0, 0, 0)` and drawing a line to the second point `(0, 10, 0)`. Next, a line will be drawn from `(0, 10, 0)` to the third point, `(10, 10, 0)`. This drawing process continues until the complete square has been drawn, as the final line is drawn back to `(0, 0, 0)`, which is the last point before the first jump. Once the back face of the cube has been drawn, another jump has been hit, and another polygon will start; it is important



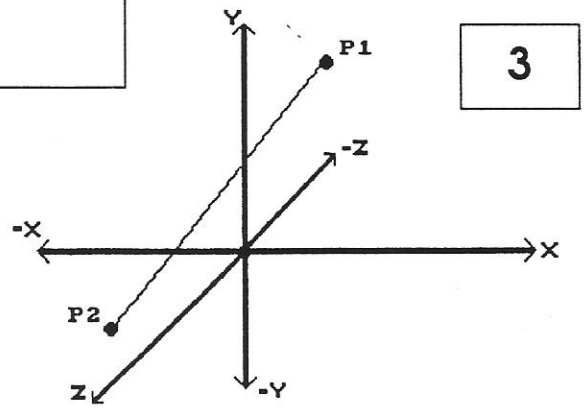
to move to the next point, `(0, 0, 10)`, *without* drawing a line from the last point of the previous polygon, `(0, 0, 0)`. Since a new polygon has been started, it should not be attached to the previous one.

While this method of representing 3D data is simple, there is one very obvious problem. Duplicate points. Notice that the point `(0, 0, 0)` is listed twice. This is to ensure that the polygon (in this case a square) is closed off. However, since we know that the polygons should

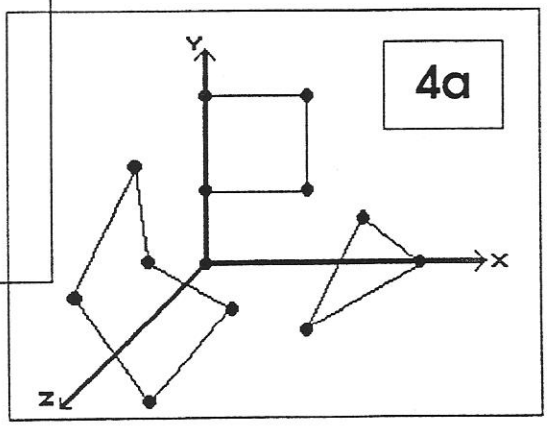


be closed, the line back to the first point can be implied; in other words, when the end of a polygon is reached, the drawing routine will automatically draw a line back to first point, which it has bothered to remember. Unfortunately, this fix does not eliminate all the duplicate points. Points where polygon edges are shared are still repeated in the list.

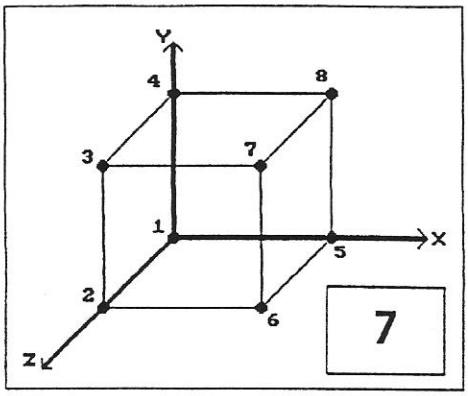
An example of this repetition can be seen in *Figure 6*. Points `P2` and `P5` are the same point. Likewise, `P3` and `P8` are also the same. Since these points are equal, there is no need to define their `x-y-z` coordinates more than once. With a small object, this waste of space is no



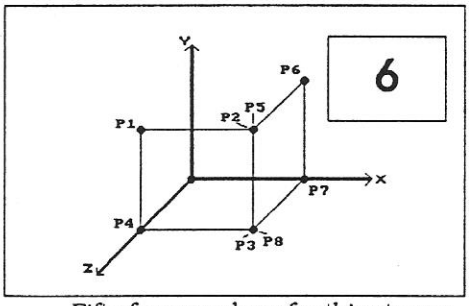
big deal, but with an object consisting of thousands of polygons, like the teapot, memory is too scarce. A solution to this space problem is the use of point-lists combined with polygon-lists. The idea is to first create a list of all the `x-y-z` points in an object, making sure that each point is represented once and only once. Next, a polygon-list is created. The polygon list consists of one polygon per line, where each line starts with the number of vertices that follow on that



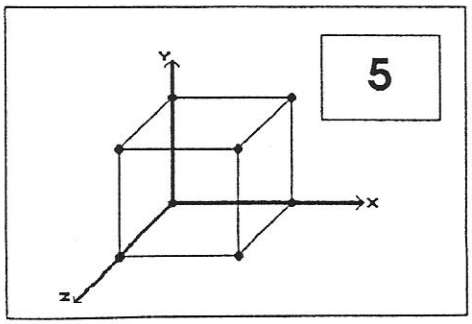
line. Study *Figure 7*, which is represented by the following polygon-list.




```
/* the point-list
0     0     0   /* point 1
0     0    10   /* point 2
0    10    10   /* point 3
0    10     0   /* point 4
10    0     0   /* point 5
10    0    10   /* point 6
10   10    10   /* point 7
10   10     0   /* point 8
/* the polygon-list
4   1 2 3 4  /* the left face of the cube
4   5 6 7 8  /* the right face
4   2 3 7 6  /* the front face
4   1 4 8 5  /* the back face
4   3 4 8 7  /* the top face
4   1 2 6 5  /* the bottom face
```



Fifty-four numbers for this storage technique as opposed to 96 numbers for the jump-list. Quite a saving, especially if you multiplied it by the thousands of points and faces it would take to define a complex object.

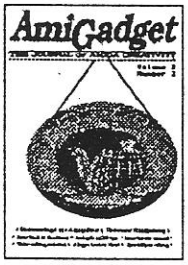


About this issue's Cover . . .

U t'ought Utah when U taw a teapot

(with apologies to Tweetie)

The teapot illustration on the cover of this issue of ***AmiGadget*** magazine was created from the official, Utah teapot 3D data. The University of Utah has been heavily involved in computer graphics for many years. Their definitive teapot data has become symbolic of computer graphics, as well as a benchmark for measuring the performance of algorithms and techniques.



Did you look at the teapot through the red-blue glasses? Did you notice that you're looking at the *back* of the teapot. For purposes understood only by graphic designers, your ruthless editor insisted on a 180-degree change in view to the data, which is normally displayed with the spout pointing the opposite direction. The lines for the red and blue stereo view are swapped, too, however. So to make the 3D viewing work, just use the viewing glasses. Put the left-eye lens over your left eye (bananas facing out), and let the right-eye lens fall where it may (over your right eye, natch), and the teapot will miraculously assume its true, 3D shape before your very eyes.

3-D files are on the disk

The 3D data, in the format discussed here, for the definitive Utah teapot, as well as several other interesting 3D shapes, are included on the ***AmiGadget* Issue Disk** for this issue, Volume 2, Number 2, available from the magazine. (*The inevitable Order Form is on page 11*). The Issue Disk also contains a viewer program, written by Dana Dominiak, which will display the data in red-blue wireframe, so that they will appear to be three-dimensional, when viewed through the appropriate lenses. And what could be more appropriate than the 3-D glasses supplied with the magazine, complete with pictures of a famous gorilla, and--no kidding) a scratch-and-sniff patch affixed just over the nosepiece (Where else!), to bring up the aroma of (What else!) bananas. Note that the 3D data is *not* itself in red-blue format, just the method of viewing.