For assignment 2 you are to write a point rendering system that adheres to (a lot of) the RenderMan Interface Specification. This will involve parsing ASCII RIB files, building and maintaining a graphics state, and eventually outputting an image file to disk.

This assignment will be broken into many smaller parts to try and keep it manageable - for some of you it may very well be the first large-scale coding effort you've ever undertaken.

The first part will be to create a system that parses the simplest legal RIB file I can think of (**simple.rib** from now on):

```
WorldBegin
Points "P" [0 0 1]
WorldEnd
```

… and outputs the raster space location (pixel location) of the point.

After we get this working, we'll teach our programs to understand more RenderMan commands and we'll code a routine to write out an actual image file. We'll be finished with this entire project when we can put an orthographic or perspective camera anywhere in 3D and view any 3D arrangement of points into an arbitrary-sized raster.

Assignment 2a - Due Tuesday February 14

Create the underlying math routines that will enable you to implement the 3D display pipeline. If I were doing this in C/C++ I would create Matrix and Point structures/classes, and I would be sure to include the following methods/functions:

multiply a matrix by another matrix, returning a new matrix
multiply a point by a matrix, returning a new point
set a matrix to the identity matrix
generate a scale matrix given sx, sy, sz
generate a translation matrix given tx, ty, tz
generate a rotation matrix about any of the three main axes (x, y, z) by a given angle theta.
and for debugging, routines to print out both matrices and points

Write a program to test your matrix and point routines. This program should take the coordinates of an arbitrary-sized raster image (you can type them in or hard-code them) and output the proper screen-to-raster transformation matrix. Include the output of this program with a raster size of 1024 x 512.

The way to test if your matrix is accurate is to send points from the corners of the screen window through the matrix (points like (-1, -1, -23), (-1, 1, 500), ((xres/yres), 1, 3.5)) and see if they land on the proper pixels of your virtual raster.

Be sure to encapsulate the library routines in separate files ("matrixMath.h", etc.) so that you can use them throughout the development of your renderer.