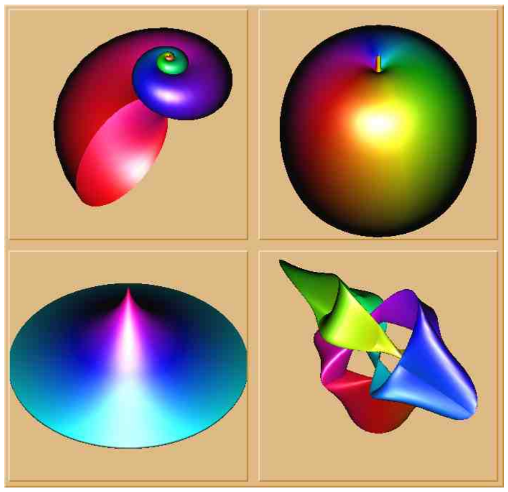
Programming Assignment #3
Parametric Surfaces



CSCI 480 Computer Graphics

Due date: Wednesday, Nov. 28, midnight. Put all your code in a folder, zip or tar the folder, and submit to moodle. **Please use your name for the folder name so I can find yours quickly and easily!** I'll have prizes for the prettiest surfaces!

The Assignment: Make a parametric surface viewer. The viewer will have mouse interaction to rotate the object, zoom with the keyboard up and down arrows, and have a variety of surfaces to choose from with the number keys.

Parametric Surfaces: A mapping from two dimensions (s, t) into three dimensions (x, y, z) determines a surface. There are many examples and discussions on the net:

- `https://www.math.duke.edu//education/ccp/materials/mvcalc/parasurfs/para1.html`
- `http://www.math.uri.edu/~bkaskosz/flashmo/tools/parsur/`
- `http://www.math.oregonstate.edu/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/parsurf/parsurf.html`

If you search for images, you will see many beautiful examples like those above.

Camera Frame: To implement the projection, your program should maintain a camera object, which will store the position, forward, and up vectors for the camera. (You do not need to mess with the projection matrix for this one. That can be a fixed uniform.) Given these vectors, you can find the right-pointing vector to build a frame. Put the vectors and position into a matrix, find the inverse, and that will be your camera (view) matrix.

Warning: Strange behavior from numpy. In one of my examples this did not work:

```
matrix2 = numpy.linalg.inv(matrix1)
```

But a version that copied components:

```
matrix3[:,:] = numpy.linalg.inv(matrix1)
```

worked fine. I have no idea why. I even compared `matrix2` and `matrix3` component by component and got identical contents, but `matrix2` gave me the dreaded black screen, and `matrix3` gave me the surface I wanted.

Trackball Mouse: Maintaining the camera frame makes a number of movement operations easy. For example, if we want to circle around an object of interest, a virtual trackball is often used. Dragging the mouse produces movement as if the object were fixed in place, but allowed to turn freely on any axis.

If you search for virtual trackballs on the web, you'll find a number of them implemented with complex algorithms for arbitrary rotation, such as using quaternions. We will take a much simpler approach, that works fairly well in practice.

Pygame provides a very convenient function for tracking mouse movements. `pygame.mouse.get_rel()` returns the relative screen u and v movement since the last time the function was called. We will assume that screen mouse movement in the horizontal direction is an attempt to move in the `right` vector direction, and screen vertical movement is an attempt to move in the `up` direction. Everytime the mouse moves (while the left button is pressed), we move the camera proportionally by those vectors.

However, we will also assume that the camera movement is constrained to lie on a sphere of radius r, and always look at the origin. Pressing the up and down arrow keys should change r, giving us the ability to zoom in or out on the origin. Moving the mouse moves the camera slightly off the sphere, so we just need to reposition the camera, and repoint the forward vector toward the origin (and adjust the other vectors as well).

Surfaces: Make at least four surfaces that can be displayed. Picking a number key will select which one to be displayed. The examples above are a good starting place, and remember to use texture coordinates (which can hold the s and t values of the parametric surface) to create good colorings.