CSCI 111, Lab 11
3d height field visualizer

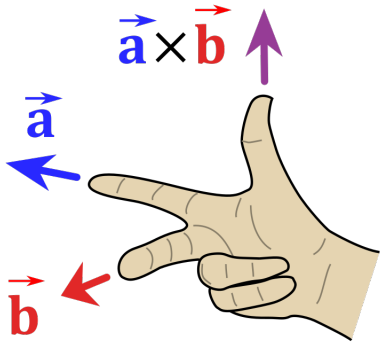
Due date: Midnight, Tuesday, December 6, on Canvas. No late work accepted.

File names: Names of files, functions, and variables, when specified, must be EXACTLY as specified. This includes simple mistakes such as capitalization.

Individual work: All work must be your own. Do not share code with anyone other than the instructor and teaching assistants. This includes looking over shoulders at screens with the code open. You may discuss ideas, algorithms, approaches, *etc.* with other students but NEVER actual code.

3d vector type: You will need your solution to Lab 10, the vector module, for this assignment. Add the following method to the class. It returns the cross product of two vectors, a vector perpendicular to both.

```
    def cross(self, other):
        a1, a2, a3 = self.x, self.y, self.z
        b1, b2, b3 = other.x, other.y, other.z
        s1 = a2*b3 - a3*b2
        s2 = a3*b1 - a1*b3
        s3 = a1*b2 - a2*b1
        return V(s1,s2,s3)
```



If you didn't finish this project you can use my implementation, available on the lab website.

The program In the folder `poly3d` on the lab website you can find a skeleton of this program. The main program, `poly3d.py` is complete, as well as the `vector.py`, `functions.py`, `gradient.py` and `polygon.py` modules.

You will only have to complete the `parametricsurface.py` module.

Step 1, heightfield: Start with implementing `makeHeightfield`. This should create an empty dictionary. Loop over i and j in range(n), and lerp i and j into the xrange and yrange, getting x and y in parameter space. Applying the function to x and y gives a point p in world space. Store p in self.heightfield[i,j]

Test this with a simple function, for example x+y, in the range 1..10 with n=10. Print out the heightfield so you can see it's working.

Step 2, project points: Now project these points using the eye vector. The method to make an orthonormal frame around the eye is part of the `Poly3d` object.

Create a forward, up, right frame around the eye, and then take the dot product of the 3d point in the heightframe with the right and up vectors to get the two dimensional x,y values in the camera frame. For each [i,j] store the x,y tuple in `self.points[i,j]`.

Also calculate the distance from the point to the eye. This is just the length of the vector from the point to the eye. Store these in `self.distance[i,j]`

Also, since we're calculating every x,y value for every point, we can also remember the minima and maxima for both x and y. Save these, as well, in `self.minx` etc.

Test this on some simple heightframes with some simple eyes. For example, what would you expect if the eye is V(1,0,0)? What about V(0,1,0)?

Step 3, make polygons: Now make polygons out of the projected points. For each point with i,j in range(n-1), use the points at [i,j], [i+1,j], [i+1,j+1], [i,j+1] to make a polygon. Use the average of these four distances as the distance to the polygon. You can use blue, `'#0000ff'` for the color.

Sort your polygons by distance, farthest first.

The best way to test now is to rerun the `poly3d` module, which will plot all your polygons. Not good? Debug!

Step 4, add color: Go back to where you projected all the points, in Step 2. Use the gradient object to find the color of the point, based on the z value in 3d. Store these in `self.color[i,j]`

Now go to the step where we made polygons, and add this color to the polygon, instead of blue.

Replot. Colorful?

Step 5, shading: Go back to where you found the color using the gradient and the z value.

Find the normal to the surface at [i,j] using the cross product of the vectors from [i,j] to [i+1,j] and from [i,j] to [i,j+1]. Take the dot product of this vector and a normalized vector pointing to the light.

You can put the light anywhere you want. Mine is in this direction: `V(2,-1,4)` . Remember to normalize the light vector!

The dot product of the normal and the light gives you the shade value, but it if's less than 0.25 use 0.25 instead of the value.