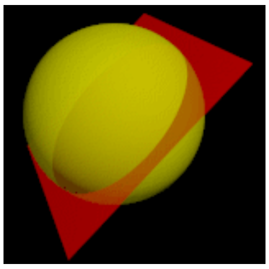
Spherical Landscapes

You'll need this if you ever get a job making graphics for space films. This article describes one way of creating undistorted continents on planets. The usual way of covering a sphere with some texture results in distortion at the poles. Another artifact of some methods is directionality. The plasma method, for example, is very noticably on a grid. This system has no directionality whatsoever.

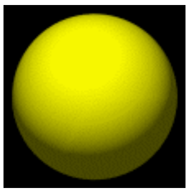
Method

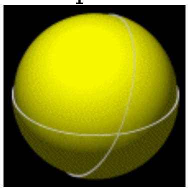


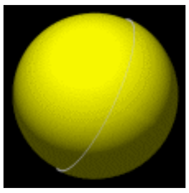
This concept here is quite simple. Implementing it is harder.

- Take a sphere
  - Cut it in half
    - Make one side a little larger
      - Make the other side a little smaller

      You repeat this process many times, choosing a random plane each time you slice the sphere in half.


The sphere before slicing.

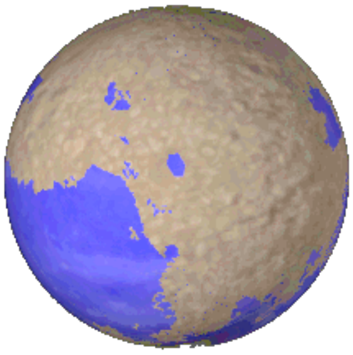
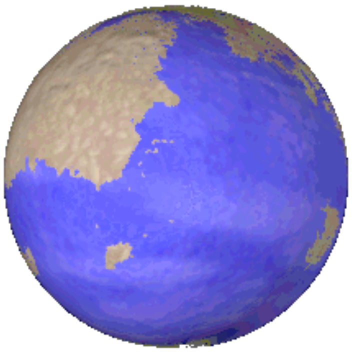

After two slices.


After one slice.

After three slices.

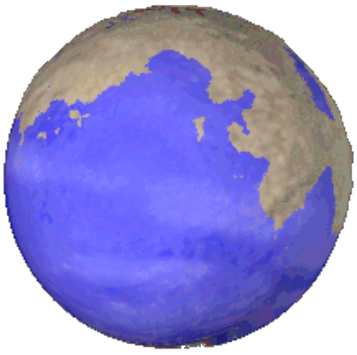
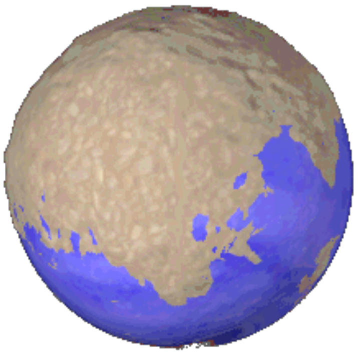
After a large number of slices, the little ridges on the sphere begin to take shape as continents. Eventually, the little ridges disappear, giving way to smooth mountain ranges. If you leave it longer, the pattern of land keeps changing, so you could leave this running till you liked the look of the planet, then stop it.

Front View | Back View

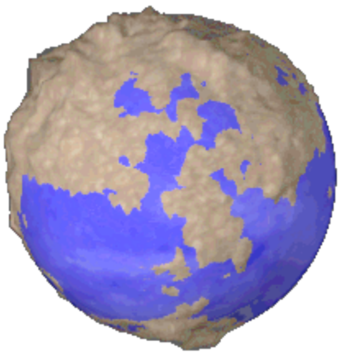
100 iterations

After just 100 iterations, nicely shaped continents are already visible.

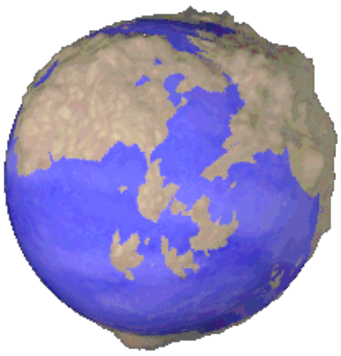
1000 iterations

Now, there is the first sign of mountains. Islands are also beginning to appear.



10000 iterations

You can now see large mountains. The shorelines are quite complex, and there are islands and lakes.



Deficiencies

The astute amongst you will no doubt have noticed something odd about these planets. The back view looks very much like the front view upside down, with land and sea swapped.
This is the unfortunate artifact of this method. Where there is land on one side of the planet, there will be sea on the other. This is not so bad, and is often not even noticeable. It took me some time to realise, and it was only because I had these pairs of pictures to compare.

Implementation

I have currently thought of two possible ways to do this, depending on what you want to produce. One method produces a texture map which can be wrapped around a sphere with no distortion. The other method produces a 3D model of the planet using polygons.

Choosing a Plane

Any method requires that you randomly choose planes to cut the sphere in half. How do you do this? To define a plane, you need to know the normal vector to the plane, and the location of one point on the plane.
All planes must pass through the center of the sphere, which makes things easier. Now all you have to do is randomly select normal vectors. You will obviously want to produce an even distribution of vectors, so you should choose vectors which lie inside the sphere.

```
procedure to randomly select plane normals

VECTOR: n

loop
    n = VECTOR( random(-1,1), random(-1,1), random(-1,1) )
    magnitude = mag(n)
until magnitude(n) < 1
```

3D Polygon Method

This method produces the best results, but can take up quite a lot of memory. The planets above each contained 10000 polygons.

First, generate a sphere. You can either write your own program to do this, or use a 3D modeling program. The more vertices the better. Also, it is best to have a sphere with an even distribution of vertices across it's surface. For example, 3D Studio can generate two kinds of sphere which it calls Lsphere and Gsphere. Lsphere is a longitude/latitude sphere and contains a higher concentration vertices at the poles than at the equator. The Gsphere is much better, with a very even distribution. At some point I will get round to writing a document on generating spheres.

OK, so now you have a sphere. The method is simple from now on.

```
loop

    n = random vector                           ; see method above

    loop through each vertex of the sphere
        d = dotproduct( n , this vertex)        ; is this vertex infront of
                                                ; or behind the plane?

        if d > 0 then                           ; if infront then
            move this vertex out a little bit
        else                                    ; if behind then
            move this vertex in a little bit
        end if
    end loop

end loop
```

Here you can use the fast dot product without square roots. Also, for speed and accuracy it's a good idea to keep an extra number (initialised to 1000 or something) with each vertex. When you move each vertex in or out a little bit, just add or subtract 1 from this number instead of moving the location of the vertex. This is obviously much faster. Then, when you have done lots of iterations, you can use these numbers to move the vertices.

Map Method

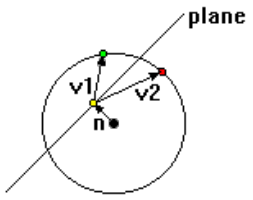
I shall explain the map method later when I have figured out a better way of doing it.

Update:

It was pointed out to me, by Toby Haynes, that the above problem, causing the land on one side of the planet to look the same as the sea on the other side, can be overcome. Rather than splitting the sphere through the middle, you can slice it absolutely anywhere you like. This produces better looking continents which don't appear to be symmetrical at all.

The modification to the existing code is very small. Now, when you generate a random plane normal, you treat it as both the normal to the plane, and the distance of the plane from the centre of the sphere.

All that needs to change is how we decide if a point is behind or infront of the plane.

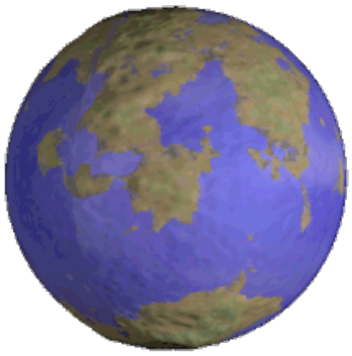


The plane is specified by the vector **n**. This is both the normal vector to the plane, and the plane's distance from the centre of the sphere. On the diagram are two points. The green one is infront of the plane, and the red one is behind. You will see that the dot product of **v1** (the vector from the end of **n** to the green point) is greater than 1. The dot product of **v2** and **n** is less than 1.

```
loop
```

```
    n = random vector                               ; see method above
    m = randomly either -1 or 1

    loop through each vertex of the sphere

        p = coordinates of this vertex
        v = p - n
        d = dotproduct(n , v)                       ; is this vertex infront of
                                                    ; or behind the plane?

        if d > 0 then                               ; if infront then
            move this vertex by m
        else                                        ; if behind then
            move this vertex by -m
        end if
    end loop

end loop
```

You will have noticed the introduction of a new variable **m**. This is here for a good reason. All the points on the sphere, behind the plane are are moved out a little bit, and those infront are moved in a little. Whenever a plane is generated, the centre of the sphere is behind the plane (unless the plane passes exactly through the middle). This means that most points are behind the plane. This in turn means that most points will be moved out a little bit. The result is the planet becomes composed entirely of land.
To overcome this, we can randomly choose whether it is points that are behind or infront of the plane than are moved outwards (and vice versa). The variable **m** indicates this. A value of -1 means "in a little bit", 1 means "out a little bit".



Here is an axample of the new method. You can see that there are no similarities between the two views of the planet.