Hex Planet

Technical Demo

Joel Davis – joeld42@yahoo.com

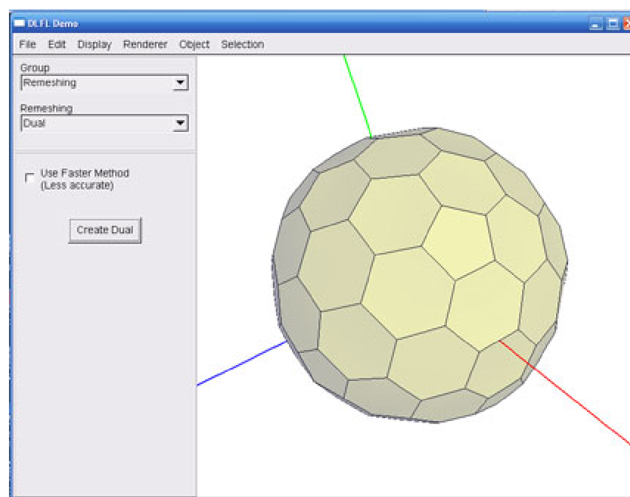
Introduction

Recently, on the yahoo Gamedesign-l mailing list, there was a discussion of different techniques for tiling a civ-like grid around a sphere for a Civ-like strategy game. One of the group members, Brandon Van Every, is developing a strategy game called "Ocean Mars" and this game takes place on such a grid. I didn't have a solution, but I kept the problem in the back of my mind.

A few weeks later, I found the work of Dr. Ergun Akleman while looking at a completely unrelated problem. One paper in particular reminded me of this problem:

> E. Akleman, V. Srinivasan, and E. Mandal, "Remeshing Schemes for Semi-Regular Tilings", Proceedings of Shape Modeling International 2005, Boston, June 2005.

This paper presented several regular tilings that would be very applicable to this problem, and it got me thinking. In addition, the construction of the tiling by subdivision of a triangular mesh, and then taking the dual mesh lends itself to a very efficient trick: By texturing the dual mesh (the triangles) with tiles of where the hexes meet, you could create seamless blends between the texture tiles, and present the hex grid without even having to construct the dual, drawing the grid with triangles but having it appear as hexes.



I used Dr. Akleman's "Topology Modeler" software to experiment with different meshes and to try different starting shapes. Starting with an icosahedron gave good results.

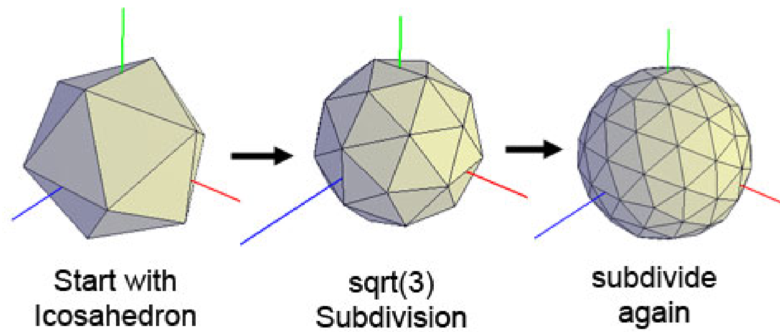
The tiling is mostly hexagonal, but there are a few pentagons at the “poles”, these are the original verts of the base mesh.

Construction

Overview

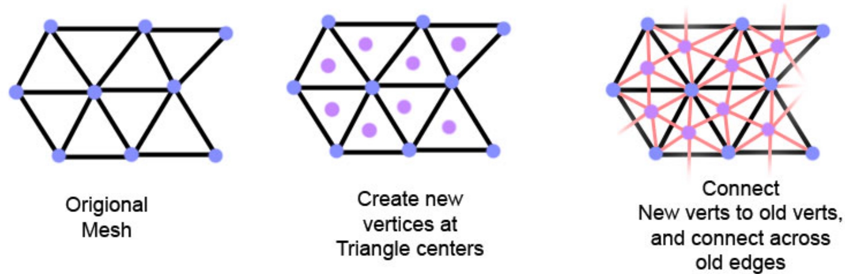
The dual mesh for the planet was generated by starting with an icosahedron (commonly known as a twenty sided die). This was subdivided using a simplified sqrt(3) subdivision scheme until the desired density was reached.

Each vertex of the dual mesh represents one hex in the final grid, i.e. the level-0 mesh would have twelve hexes.



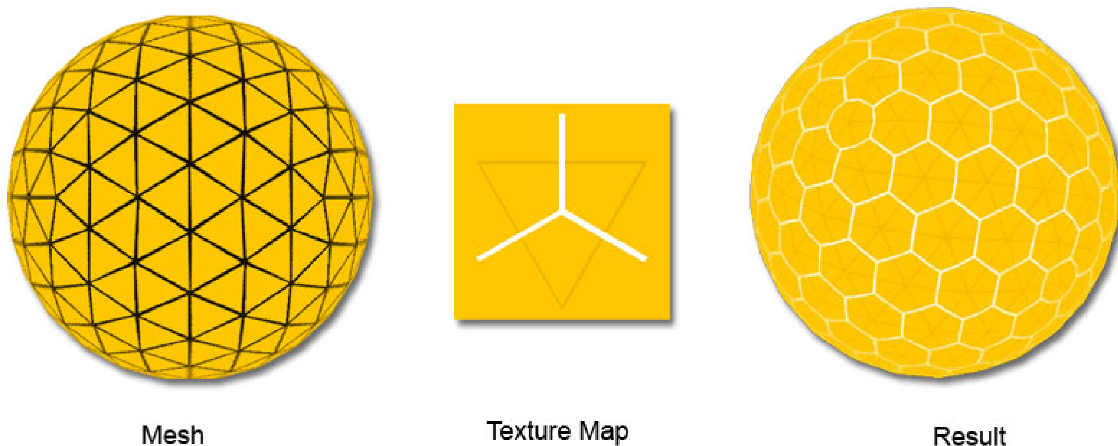
Subdivision

Subdivision was done with a modified version of the sqrt(3) subdivision process. Since I knew the result would be spherical, I only performed the topological step of the subdivision, and created the new vertices at the center of the triangles to be subdivided. The new verts were then simply projected to the surface of the sphere.



Texturing

By texturing each triangle with a pattern of three lines radiating from the triangle center and crossing the midpoint of the edge, we get a visualization of the dual mesh, without actually having to construct and draw the polygons.



(The darker orange lines on the texture indicate where the triangle edges are in texture space).

Tile Artwork

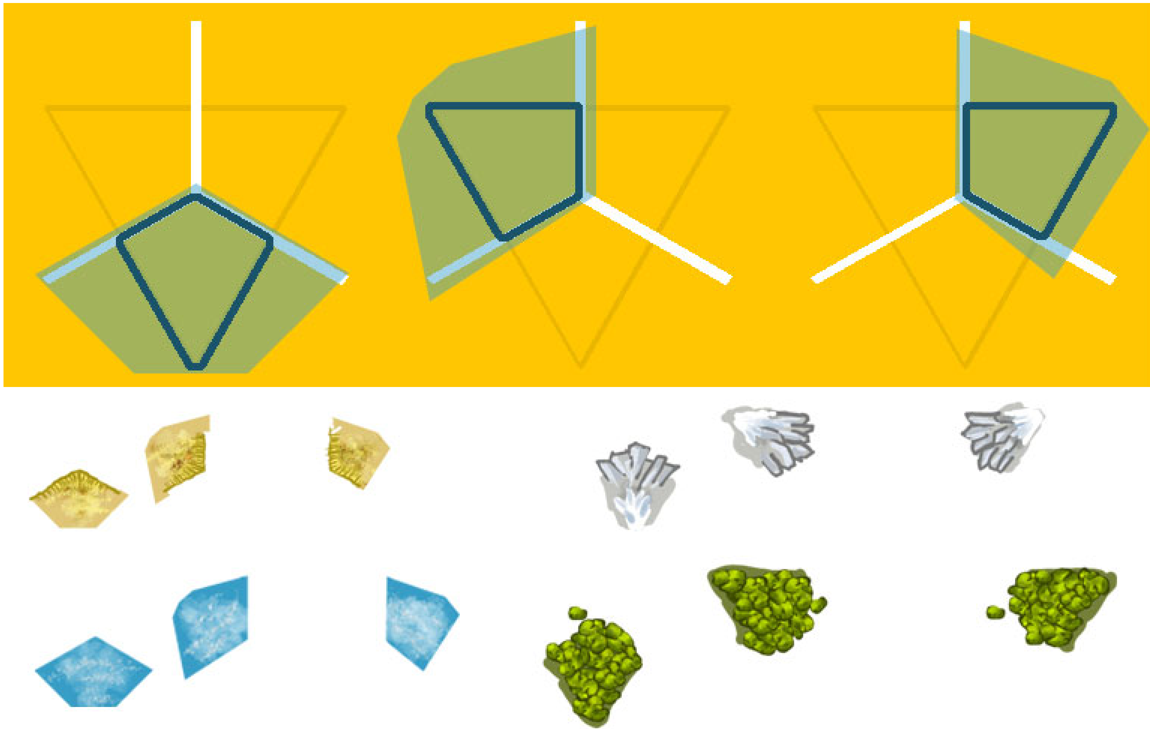
The next step is to create a custom set of texture maps (actually just one large texture atlas) that contains every combination of terrain corners. I choose to have five terrain types:

Water, Desert, Plains, Grassland, Mountains

So for each of the three corners, that gives 125 combinations. Another option would be to also consider rotation and mirroring of the texture coordinates, needing fewer maps, but this would give less variety and make the process more complicated.

Starting with the template texture map pictures above, I created a template for the tile artwork, and painted texture chips for each of the texture types that I wanted.

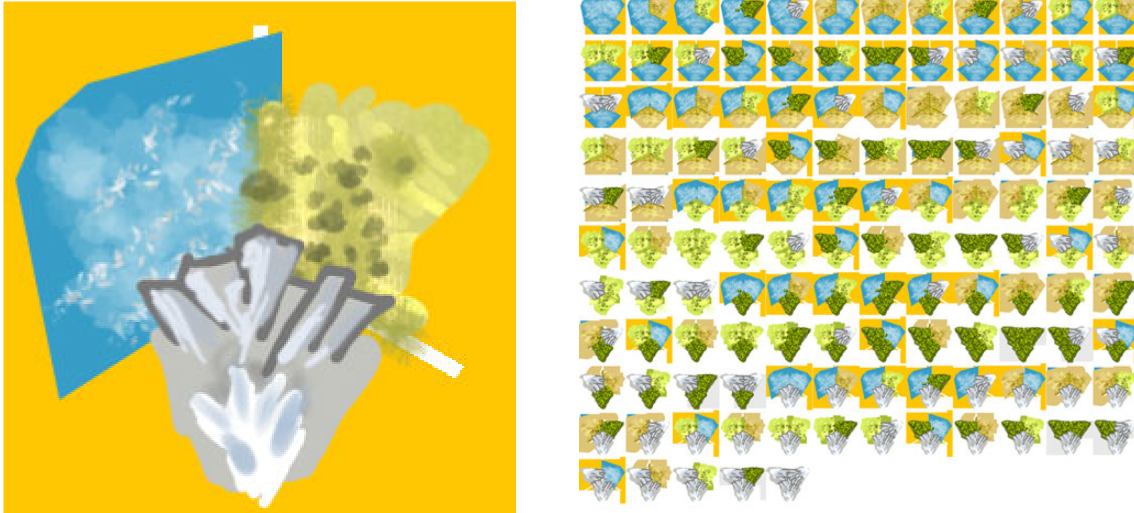
The template I used and some example texture chips are shown below:



I painted the tiles with a fair amount of overlap to ensure interesting transitions at tile borders.

These images were scaled down and combined into a single large texture map containing all combinations of the five terrain types. (I used a Python script with PIL to do this).

One terrain tile, and the complete tileset:

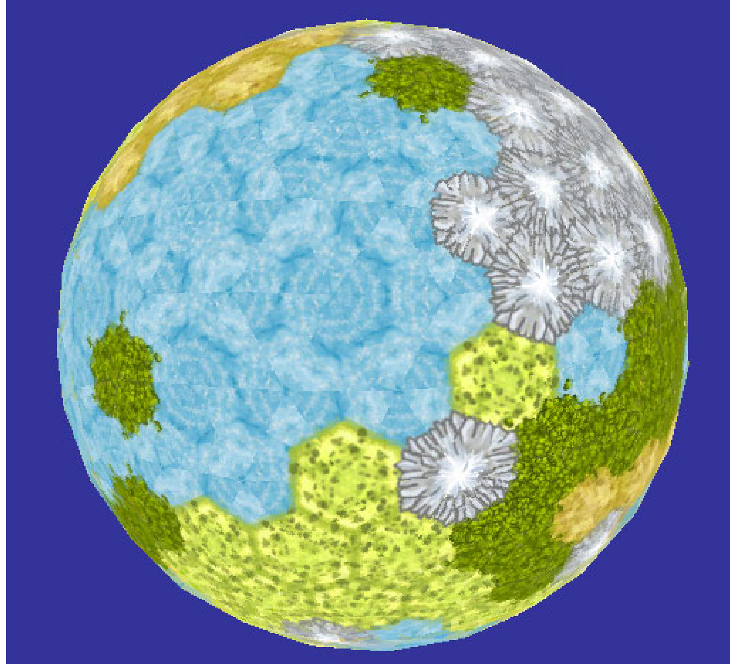


Each terrain was given a priority, from Water=0 to Mountain=4. When compositing the tiles atop each other, this priority would be used to determine which texture was on top. This allowed nice, overlapping transitions without too many extra combinations. (I learned this trick from studying the terrain texturing in Warcraft II).

Because this was just a demo, the tiles weren't packed as tightly as they could be, and there is a fair amount of wasted texture space.

A second tilemap texture was created that also had white lines overlaid on the hexes, as in the template. Switching to this texture allowed a "grid" display to be toggled on.

I didn't spend too much tile making sure the tiles would line up across triangle borders. Instead, I tried to paint the textures so that such seams would not be too noticeable, and they are. For a real project, it would be worth investigating a solution that allows the source art to be created as actual hexagons, and splits them up into terrain clips to eliminate these texture seams.



Results: The texture tiles applied to the planet mesh. Almost all hexes, but notice the pentagon mountain tile in the upper right, near the “ocean”.

Terraforming

I used a very simple terraforming scheme for the demo, a real game would need to do something more complicated. Random hexes were chosen as by checking against a percentage that they would be “water” (called the “watery factor”). If they were land, the type of land was randomly chosen. Every tile in the initial mesh was chosen like this.

When new tiles were created during subdivision, there was a chance (the “random” factor) that the tiles would be chosen randomly, as above. If they were not chosen randomly, they would inherit the terrain type of one of the three triangles at the mesh level above them.

This simple scheme gave a pretty good amount of control. By painting terrain on a lower level hex, and then subdividing with a low randomness value you can quickly create desired continent shapes and terrain makeup. By subdividing with a high randomness and watery a few times, and then a low ones on later iterations, you can create terrain that is varied and interesting, but also contains groups of similar terrain features.

Using the Hexes

This works great for drawing, but sometimes you need to get the actual polygonal border for the hex. For example, when drawing a cursor to highlight a tile, or to trigger a visual effect based on the tile's area. You also need to be able to convert from world space to a map tile, and back.

There is example code to do all of this except for pathfinding in the demo.

- **Finding the polygon for a hex.**
  The actual polygon can be found for any hex (or pentagon) by taking the triangle centers of each of the triangles that share the vertex for this hex. These can be sorted around the "hex normal", which is trivial to compute since it's a sphere, just the normalized hex center. In my demo, I find the neighbors and store them in the proper order for each hex during the construction process for easy lookup later.

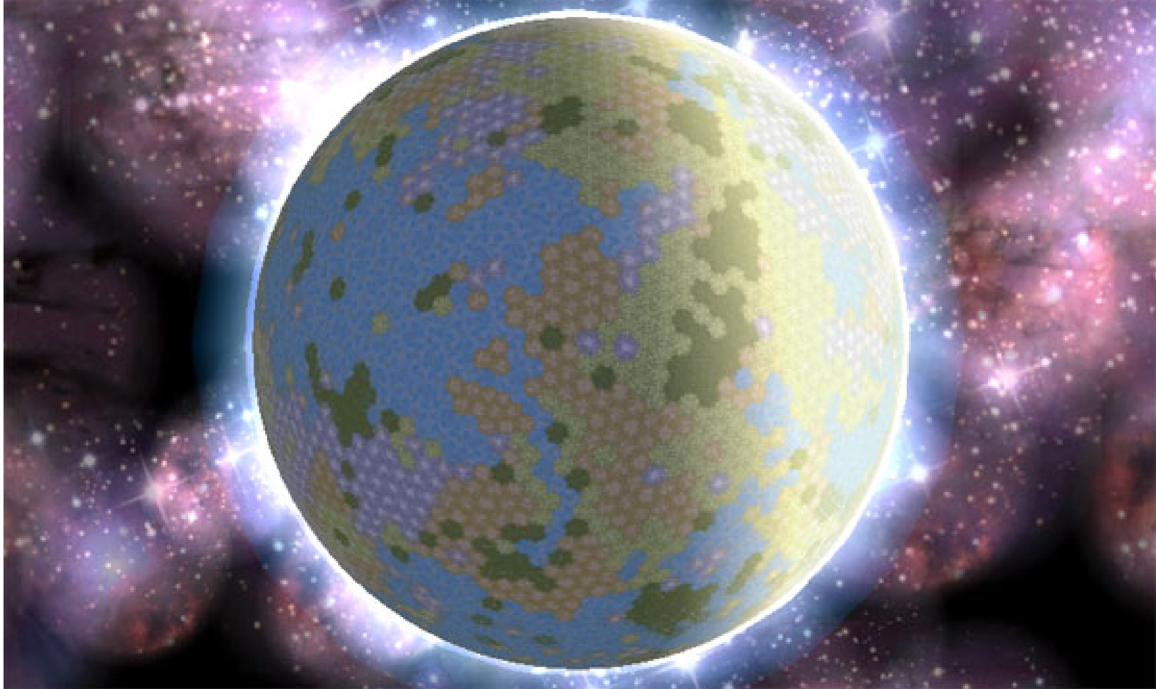
- **Finding the hex for a point in world space**
  First, the point normalized to a direction vector. The angle between the point and the hex center can be found by acos((n dot hexpos) / planetRadius). Whichever hex has the smallest such angle contains the point. In my demo, checking every hex was fast enough even on a large subdivision level, for a game you would probably want to add an acceleration structure (perhaps as a 2D kd-tree in spherical coordinates).

- **Finding the hex the cursor is over**
  A ray-sphere intersection will give you the mouse position on the planet's surface, and then a the hex can be found as above.

- **Find neighboring hexes**
  The neighbors can be found the same way as in the "finding a polygon" example above, except collecting the other verts of the triangles instead of the centers.

- **Path-finding on the sphere**
  Since you can find the neighbors, any traditional path-finding algorithm should work

Rendering

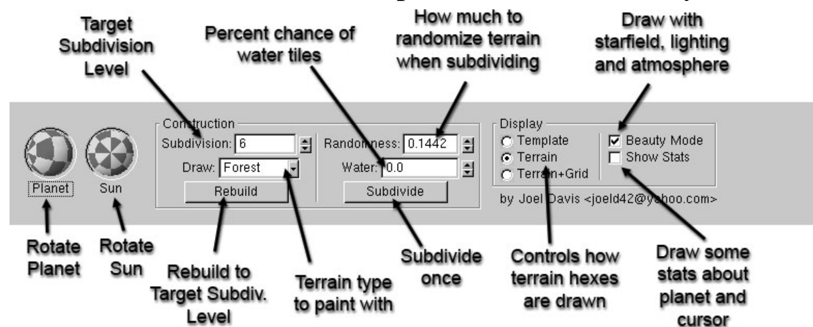


Rendering eye-candy includes:

- Background stars/nebula
    - These are simply a bunch of billboards drawn in additive blend mode.
- Planet glow – A ring drawn behind the planet
- Lighting and atmosphere
    - Stupid GL lighting. I almost broke down and started writing shaders at this point, but I resisted. This is just a glutSolidSphere rendered as an overlay on top of the planet. Because GL lighting doesn't give you an easy way to get a "hard" falloff across the day/night side of a planet, this is drawn completely overbright, and then the planet is rendered again on top with transparency. By drawing the "lighting sphere" slightly larger than the rest of the planet you get the effect of an atmosphere.

User Interface

The user interface is built with the GLUI OpenGL interface library.

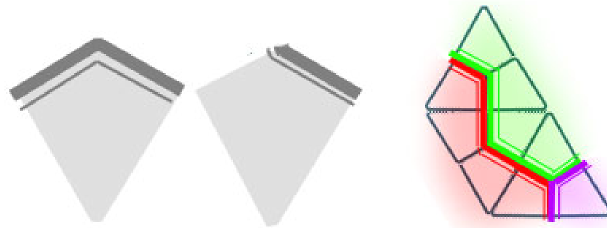


Future Work

I'm not planning on doing anything else with this demo. It was just built to satisfy my curiosity. But, there were several ideas and areas of improvement that I thought about when building it but didn't get around to doing.

Political Borders

In a strategy game, political borders are often very important. By drawing the tiles in a slightly different way, this scheme could be easily adapted to draw arbitrary borders around regions of tiles.



By building a set of texture "darts" as alpha textures, you could draw each hexTri as a set of three quads instead of a triangle(as pictured). By choosing the appropriate texture dart, political borders could be displayed. This method of drawing has the disadvantage that the "borders" cannot blend across hexes, so it wouldn't be as good for drawing the hexes themselves, but would work great for any type of overlay graphics. The border mesh could be drawn as a second pass on top of the planet mesh.

Misc. Enhancements

- Right now, "findNeighbors", which finds which triangles are connected across an edge, is the slowest part of the subdivision. It uses a brute force N^2 search after subdividing. This could be much improved by keeping track of the neighbors when performing the subdivision.
- On modern graphics hardware, rather than using a single textured triangle, a set of prebuilt geometry triangle tiles could be built, with hundreds of polygons each. With instancing, this could create an incredibly detailed hex planet with millions of polygons.
- Beauty Mode needs clouds and better lighting.
- Regions and borders as mentioned above.
- Better editing terraforming tools

Sample Code

The source code for the hexplanet demo is included, and released under a BSD license. It should be pretty much platform independent, but I haven't set up and build linux makefiles for it yet. It needs the following libraries to build:

- GLUI User interface library (http://glui.sourceforge.net)
- The Imath vector math library (part of OpenEXR, http://www.openexr.org)
- The GLEW GL Extension Wrangler (http://www.glew.sourceforge.net)
- The GLUT framework (from http://www.opengl.org)

Prebuilt versions of all of these libraries are bundled with the NVidia SDK, which is convenient if you don't already have them installed.

Links and More Information

Dr. Ergun Akleman's Topology Research

Includes the TopMod modeling software and the papers that inspired this project.

http://www-viz.tamu.edu/faculty/ergun/research/topology/index.html