Tiny Planets

Sharzard Gustafson, Matthew Pohlmann, Ricardo Sisnett
University of Southern California, Los Angeles, USA.
{*sagustaf, pohlmann, sisnetth*}*@usc.edu*

Keywords: Procedural Generation, Graphics, Perlin Noise

Abstract: In this paper we present a method of procedurally generating stylized planetary models with geography by combining multi-fractal Perlin noise and recursive icospheres. Planets are further decorated with procedurally modeled vegetation placed accordingly to the geography. A materials system was implemented to visualize the results of the generation algorithms.

1 INTRODUCTION

In this work we present the results of implementing a series of techniques in a bare bones real-time graphics engine, extending its core by adding a materials system and creating an application that utilizes procedural modeling techniques to create planetoids with natural looking geographical features. The work is a broad exploration of a few techniques in the area of procedural modeling.

The graphics engine used, is built using the Simple DirectMedia Layer (SDL), a cross-platform solution for access to low-level peripherals and graphics hardware, and the openGL Extension Wrangler library (GLEW).

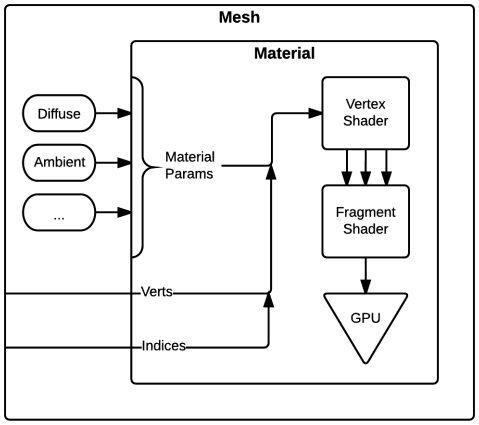

Figure 1: Technical diagram of our simple material system

2 MATERIAL SYSTEM

A material system is a framework that builds on top of and abstracts the functionality and bindings of basic vertex and pixel/fragment shaders, and is commonly used in real-time graphics applications. Materials combine a vertex and fragment shader in order to re-use common shading functionality amongst multiple meshes, and provide access to shader parameters (like emissive and diffuse colors, specular power, textures, etc.) in order to customize the appearance of different surfaces; the result is that only these parameters need to be changed to represent a multitude of surfaces rather than writing and loading shader programs for each surface. An added benefit is that it becomes possible to render all meshes with common materials without changing the shader program on the GPU by simply modifying the current shader program's inputs, improving performance. Our material system implements basic Phong shading in a single vertex and fragment shader pair, and allows customization of ambient, emissive, diffuse, and specular colors, specular power, and a single texture. The system also adopts the concept of material instances in order to allow different meshes using the same material to modify parameters without affecting each other. Figure 1 is a visual representation of the system we have designed and implemented for the project. The quicker iteration time provided by the material system in customizing the different meshes in the application allowed us to focus on achieving the stylized look of our original goal.

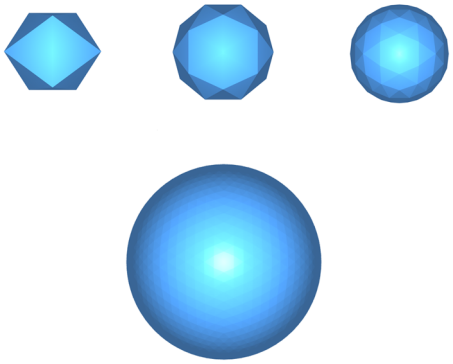

Figure 2: Icosphere after different levels of iteration produced by our implementation. As mentioned it is clear that this system lends itself nicely for LODs systems

Figure 3: Left, the unmodified algorithm after 2 iterations that produces indexed meshes and interpolates vertex normals. Right, our modified algorithm that produces surface normals.

3 ICOSPHERES

An icosphere is one of two of the most common ways of rendering spheres in computer graphics applications, the other being the UV sphere. An icosphere starts by taking a 20 sided polyhedron (icosahedron) and recursively subdividing the faces to add detail. This method of generation makes the sphere's vertexes to be evenly distributed along the surface, which is a desirable trait for our use case since it lends itself better for deformation than UV spheres [REF]. Figure 2 illustrates the basic idea behind this algorithm. The number of iterations can be changed to produce more or less detailed icospheres for planetary generation. In the general algorithm, generated vertices are cached in order to avoid duplicates, reduce generation time, and allow for vertex indexing on the GPU. Furthermore, normals are trivial to produce: because we generate a unit sphere, normals are vertices have the same x,y,z components of the vertices themselves. This method - when combined with Phong shading - produces incredibly smooth-shaded spheres after only a few iterations; however, our stylistic goal involves flat shading, so we modified the algorithm to duplicate vertices and calculate surface normals per-triangle. Figure 3 shows the difference in the two techniques. Interestingly, the iterative nature of generating an icosphere lends itself well to Level of Detail (LOD) systems, where lower iteration counts can be used for distant icospheres, though this concept was not explored in our project.

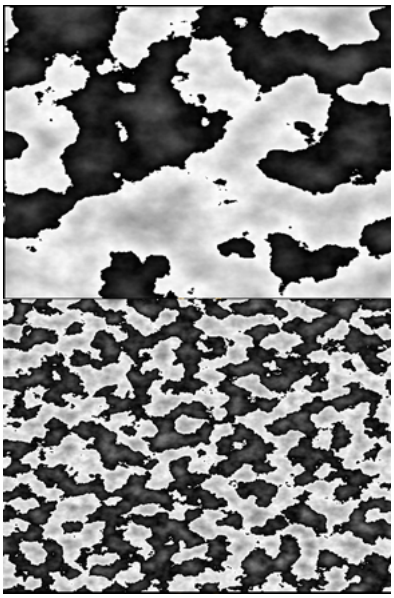

Figure 4: Textures generated by our implementation of the Perlin Noise function.

4 PERLIN NOISE

Ken Perlin's algorithm for creating pseudo-random noise is a staple in any application trying to replicate natural occurring patterns. The algorithm is fairly simple and consists of three steps: defining an n-dimensional grid, computing the dot value of distance-gradient vectors and interpolating this values.

The process to create the noise is then repeated and the generation function is scaled, we call the scaled results octaves. Stacking these octaves gives the irregularity and details on different heights usually seen only naturally occurring phenomena.

4.1 Multi Fractal Perlin Noise

Classical Perlin noise is homogeneous and isotropic, this plays against the rationale to use Perlin noise: creating natural occurring patterns. As defined in (Linda, 2007) a *multifractal* is a heterogeneous fractal, this is achieved by parameterizing the dimension of the fractal function with another attribute, such as the height value of the terrain, this makes it so values closer to 0 (sea level) are smoothed out, whereas high values higher octaves have a bigger effect causing jags. The algorithm implementation is based in (Perlin, 2002).

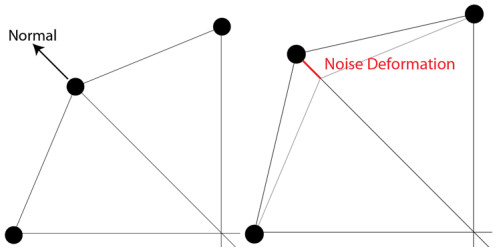

Figure 5: Terrain deformation illustrated

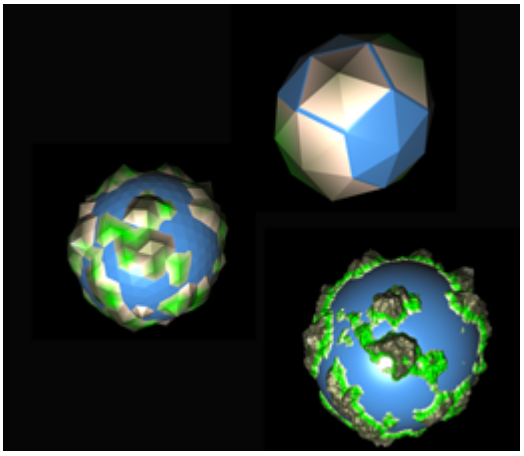

Figure 6: The same planet at different levels of recursion.

5 TERRAIN GENERATION

Using an instance of the multi-fractal Perlin noise, we deform vertexes as they are being created by the icosphere algorithm to create geographic-looking features. We use a *ridged* version of the algorithm to produce more interesting patterns as described in (Linda, 2007). The terrain generation code can further parameterize the deformation by scaling the result or

The deformation is the result of displacing the vertex by a value up to a maximum deformation in the direction of the vertex normal vector, which could be interpreted as 'up' in the terrains frame of reference, thus creating a relief.

Planetary terrain is colored per-vertex using a weighted color look-up table where the index is calculated using the vertex's 'altitude' (simply the magnitude of the vertex's position). The vertex colors are interpolated by the GPU and then applied per-pixel rather than using a texture.

6 ASSET GENERATION

Different assets to accompany the planets where also created by the system. The main focus of our runs was to create foliage, but other types of assets such as clouds for the sky or rings for the planetary bodies are other possibilities explored.

Figure 7: Different types of vegetation generated by the system.

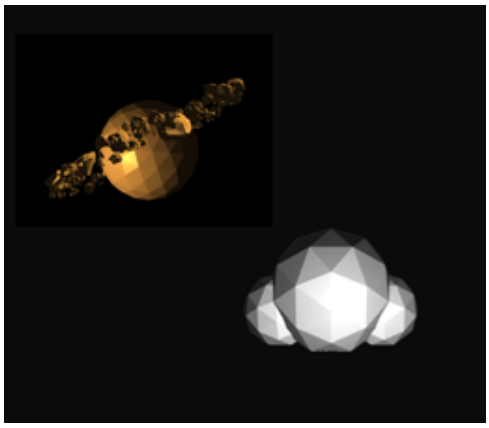

Figure 8: A planetary ring generated from a basic rock mesh and noise and cloud from icospheres. Other assets generated by the system

When planning the design of our foliage system, we initially look at using an L-System for the generation of trees, however due to our low polygon style, the detail that an L-System provides was not needed, however we did use similar concepts for creating the tree branching.

Different types of trees were generated with similar algorithms, inclusive reusing some of the icosphere generation algorithm to create geometry. Figure 7 shows examples of different trees generated by our system using different levels of branching and base meshes.

REFERENCES

Linda, O. (2007). *Generation of planetary models by means of fractal algorithms*. PhD thesis, Czech Technical University.

Perlin, K. (2002). Improving noise. In *Proceedings of the*

29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02, pages 681–682, New York, NY, USA. ACM.