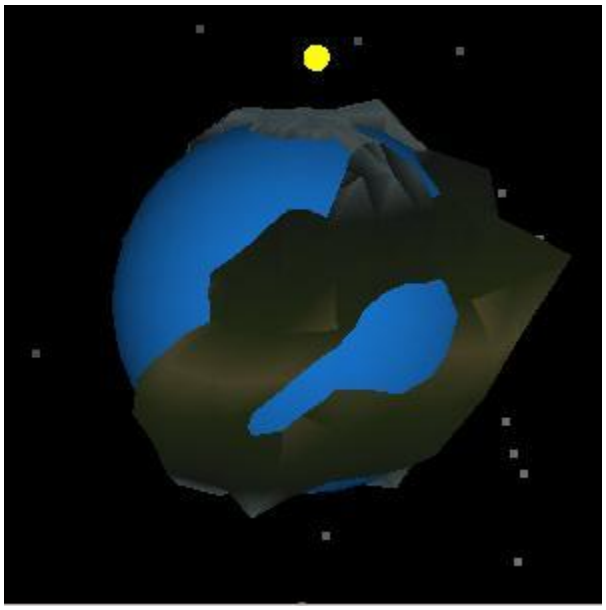
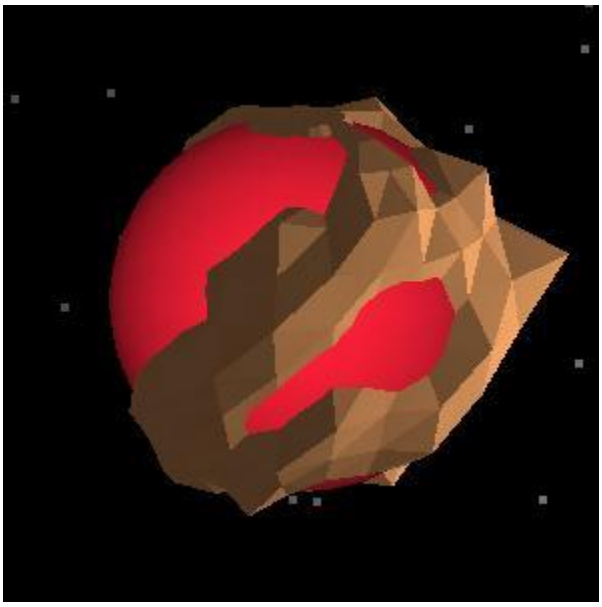
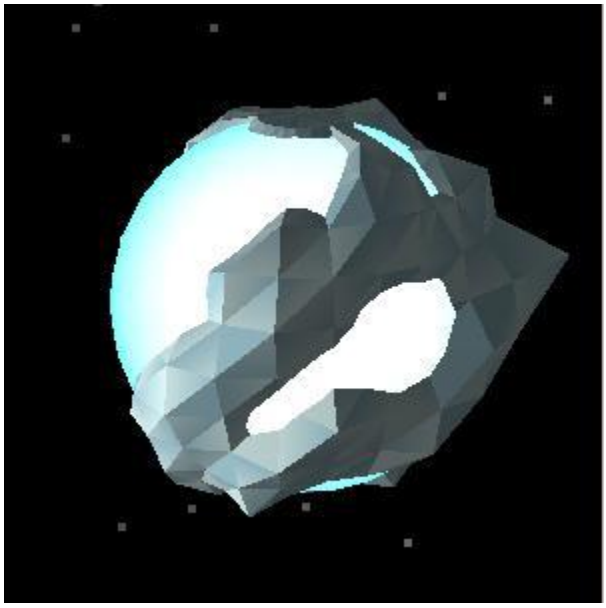
Procedural Generation of Dynamic Terrestrial Simulacra

Data Structures, Vector Algebra, and the Creative Process







Steve McKinney

Rensselaer Polytechnic Institute

Inspiration

"Space," it says, "is big. Really big. You just won't believe how vastly hugely mindbogglingly big it is. I mean you may think it's a long way down the road to the chemist, but that's just peanuts to space, listen..."

-Douglas Adams, A Hitchhikers Guide to the Galaxy

Using technology to envision the far reaches of space is certainly far from a new idea in the field of computer graphics. For years computer-generated images of the cosmos have been produced for scientists and Discovery Channel viewers alike. However a somewhat recent trend in computing referred to as “Procedural Generation” has paved the way for a whole new way of imagining the final frontier. Procedural generation allows programmers to produce vast amounts of content, which is created solely through the use of algorithms rather than being manually assembled. With the proper coding and the right sets of rules it is possible to generate iterations of just about anything.

While procedural generation has been around for quite a while(fractals are a good example), it wasn’t until 2005 that I took notice of the concept. At 2005’s Game Developers Conference industry legend Will Wright revealed the first details of his new game: Spore. Wright touted the game as a living, breathing galaxy filled with unique creatures, planets, and architecture. While the game would go on to be panned by critics and fans alike, Spore was very successful in demonstrating the power of procedural generation. Wright’s behemoth boasted thousands and thousands of unique planets, all generated on the fly using procedural generation.

Will Wright's *Spore* in Action

Infinity: The Quest for Earth

Now, the creator of *The Sims* isn’t the only one harnessing the power of procedural generation. What really influenced me to do this project was the work of Flavien Brebion, a single programmer building a galaxy sized game in his spare time. Briebon’s magnum opus, *Infinity: The Quest for Earth,* will allow players to traverse and colonize an entire galaxy of procedurally generated content. Though the project is still in alpha, Briebon’s accomplishments are already quite impressive. Having seen footage of the game in action, I decided I should try my own hand at building planets albeit on a much smaller scale.

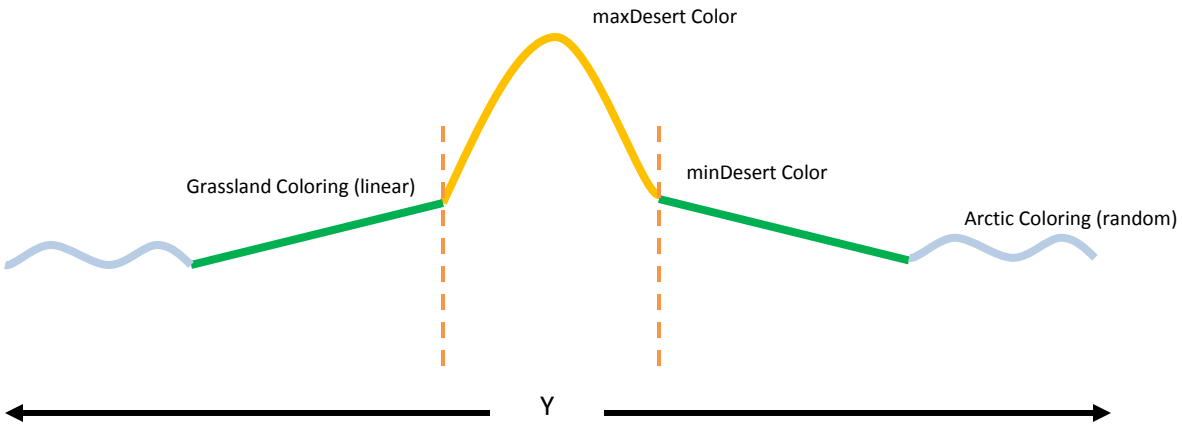
About Planet Processor

Planet Processor is a series of algorithms designed to mould a simple sphere into a unique planetary body. My program takes a set of specifications from the user, and feeds those specifications into a series of functions which appropriately shape, size, and color the new planet. Save for a few persistent features, such as polar ice caps, each planet produced is completely different from the one before. Planet processor can produce planets of varying sizes, densely or sparsely populated landmasses, and various different climates. The user may choose to dictate the specifications of these features, or generate a completely random planet.

Algorithms

To achieve my goals with this project I crafted a series of functions which determine the ultimate configuration of each planet. Specifically I relied on trigonometric functions for the planet coloring, and a mesh-traversal algorithm to sculpt the surface.

My coloring algorithm is a fairly simple yet effective setup. A piecewise function is applied to the entire sphere, with each vertex's distance from the equator.



As the figure above demonstrates, depending on a vertex's relation to the equator it will be colored very differently. The desert equator regions are assigned a color which is a factor of the cosine function and the position within the region. Similarly the grassy areas are linearly assigned colors based on their position on the sphere. Finally the polar regions are assigned a white/blue gradient to signify an ice covering or permafrost.

The coloring system is dynamic as well. The user can at any time adjust the relative heat of the planet. Increasing the heat shrinks the polar regions and gradually widens the sandy desert regions. With enough increase in heat the planet soon resembles a molten rock. Similarly the planet can be cooled back to its original state, or cooled further which leads to the expansion of the arctic regions.

I also used a series of mesh traversal schemes to elevate certain portions of the sphere and therefore serve as landmasses. The surfaceBuilder function selects a triangle at random, then it is made sure the current triangle is underwater, and the extruded triangles expand in a circular fashion. There is also a chance that the direction of the landmass creation will change, allowing for uniquely shaped continents. The user has the option of determining a factor between 0 and 10 which will dictate how long this traversal runs, and therefore determine the size and abundance of dry land.

Once the continents are unearthed, the new "continent triangles" are randomly selected to be elevated to mountains. The exposed land is randomly tweaked so as to not appear unnaturally spherical.

Pitfalls

Often the hardest part of this project was determining my best next step. I was faced with so many options for expansion, however each new feature added many new bugs to the project as a whole. Ultimately I had to sacrifice a more complex final product, for one that actually worked as intended

A Pointy South Pole, the result of adding mesas

Conclusion

This project thoroughly challenged my programming skills. I spent roughly 40 man hours on the project and received no outside assistance besides the code provided by Professor Cutler et al. through the homework assignments.