Procedural Planet Generation based on derivate fBm noise

Ricardo B. D. d'Oliveira, Iago P. do E. Santo, Antonio L. Apolinrio Jr.
Departamento de Ciência da Computação
Universidade Federal da Bahia
ricardo.barros(at)ufba.br, iago.esp(at)gmail.com, antonio.apolinario(at)ufba.br

***Abstract*—Terrains requires large amounts of data to be represented in planetary scale. Procedural Content Generation techniques can reduce storage requirements. This paper uses a derivate fractional Brownian motion algorithm to procedurally generate a planetary terrain model and a view-dependent Level of Detail data structure to achieve realistic planetary terrains, with temporal coherence, in real-time.**

***Keywords*—procedural generation, fractal terrains, planet rendering**

I. Introduction

Computer games consists of multiple contents varying from geometric models to sound effects, known as game assets. Games often use premade assets, however due to the increasing performance of computers, Procedural Content Generation (PCG) techniques have grown in number and usage nowadays, games rely heavily on content generated procedurally. While manual content requires time and resources to be made, content generated procedurally can be a cheap alternative to content creation. PCG comprises of methods for generating data algorithmically rather than manually, which in turn can reduce project costs and effort.

In this paper, we propose to procedurally generate planetary terrains with a variation of fractional Brownian motion (fBm) with the aid of shaders, resulting in terrains which closely resembles planetary bodies. In terms of performance, this approach is better than other noise algorithms for multifractal terrain generation, it uses direct fBm through derivation instead mixing noise functions that other Procedural Terrain Generation (PTG) techniques use. As a result a synthetic planet can be generated in a single pass, instead of using multiple passes for each distance relative to the camera.

Our application renders a terrain model at nearly real time, with ∼60 frames per second, we take advantage of reducing geometry complexity in a view-dependent way through the usage of displacement mapping and view-refinement algorithms on the GPU.

This paper is organized as follows: Section II describes related work. Next Section III describes our approach. Next Section IV shows the performance evaluation of our application and finally in Section V we show our conclusion remarks.

II. Related Work

In this section we are going to review approaches for terrain and planetary rendering which focus on visual quality, LOD based rendering and efficiency. For surveys regarding terrain rendering we suggest the readers to [1] for planetary rendering [2], for procedural techniques [3] for further reference.

A. Terrain Representation

1) Seamless Patches: Livny et al. [4] presented a quadtree technique appropriated to render large terrains, which is well suit for our approach, it subdivides the terrain into a regular grid and different resolutions which are stitched together with rectangular patches.

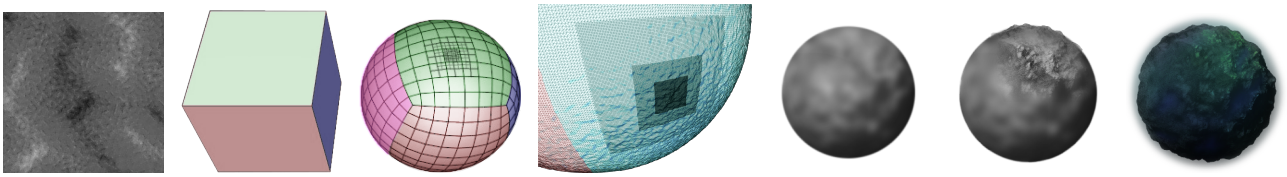
B. Planetary Rendering

1) Planetary Scale Composition: Kooima et al. [5] claimed that many planetary models exists and each model require different tools for visualization. Those tools are often unregistered, having different resolution, projections and formats. Planetary Scale Composition proposed a novel approach where terrain data only exists on the GPU, allowing powerful compositions to be applied on both the height and surface resulting in seamless and smooth interpolations in both imagery and geometry [5]. This technique serves as a basis for our approach in handling and displacing the mesh with a procedurally generated heightmap in the GPU.

2) Projective Grid Mapping for Planetary Terrain: Mahsman et al. [6] present a hybrid technique for planetary terrain visualization that combines rasterization with ray casting creating a view dependent mesh on the GPU [6]. We implemented a simplified version of the projective grid mapping as a parallax occlusion mapping for our approach, wh.

III. Our Approach

We propose an approach that combines a derivate fBm by [7] with GPU planetary rendering. The terrain is generated on the CPU, while the mesh is displaced on the GPU, consisting on the following steps: (*i*) generating a heightmap with fBm, (*ii*) normalizing a base mesh into a sphere, (*iii*) indexing the normalized mesh through a view-dependent mesh refinement with seamless quadtree (*iv*) applying a interpolation on the GPU to prevent inconsistencies across different LOD regions, (*v*) using a displacement mapping on the GPU, (*vi*) adding fractal details with a progressive refinement, (*vii*) applying a projective mapping to correctly display features which rise from the horizon and (*viii*) a final shading step which consists on mapping the texture through a triplanar approach. Our approach is depicted in Figure1.



(a) (b) (c) (d) (e) (f) (g)

Figure 1: Our approach consists of (a) heightmap generator, (b) base mesh, (c) cube to sphere and quad tree indexing, (d) hybrid GPU/CPU LOD strategy, (e) GPU displacement mapping, (f) GPU synthesis refinement and (g) GPU shading.

A. *Multifractal Brownian Motion*

A terrain can be synthesized with a fractal Brownian motion[8] (Figure1a), by simply mapping a sum of noise functions with decreasing amplitudes and increasing frequencies. Where in $fbm(x,y) = \sum_{i=1}^{n} A^i \cdot noise(\omega^i \cdot x, \omega^i \cdot y)$, ω^i is the frequency and each iteration of ω is called an octave where $i = 1, \omega^1 = 2$, in contrast A is the semi-amplitude, half of the peak-to-peak amplitude, wherein $A^1 = \frac{1}{2}$. A simple iteration of the fBm function can be achieved by $f(p) = fbm(p)$ where p is a point in space comprising of its x, y components and $f(p)$ is a compact form of the image defined as a function of space which can be decomposed into $noise(x) = \frac{\partial n}{\partial x}$ and $noise(y) = \frac{\partial n}{\partial y}$ which in turn is the derivative of the noise function (∂n) of the traditional fBm function. The derivate noise function shown in those two decompositions is based on a linear interpolation of random k values at some given lattice points v, w expanded on $\partial n \partial x = (k_1 + k_4 v + k_6 w + k_7 vw) \cdot u^{'}(x)$, where $u(x)$ can be either polynomial functions such as $u(x) = 3x^2 - 2x^3$, or $6x^5 - 15x^4 + 10x^3$ [7]. Which one is chosen depends on the number of k_i iterations.

Since this approach depends on analytical derivatives computation, it results in a computationally viable fBm approach that even have a richer variety of landmarks compared to a regular fBm function, resulting in a complete multiresolution heightmap which does not require any blending with other noise function to create a diverse scene comprising of mountain, mountain ranges, cliffs and ledges. This approach doesn't require to compute multiple samples of the fBm, thus it is much faster and more accurate than central difference methods, such as Diamond-square[9].

B. *Normalizing a Cube into a Sphere*

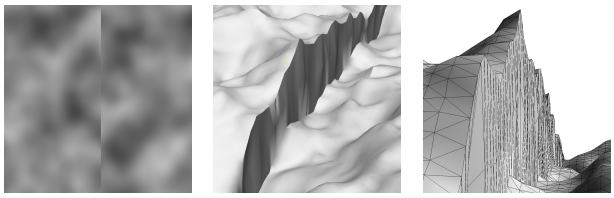
To produce a uniformly triangulated sphere, we begin with a cube (Figure1b), and proceed to subdivide its faces until a criterion defined by a product between `PATCH_SIZE`, `PLANET_RADIUS` and `PATCH_MULTIPLIER` is met. Because we use a cube, a cubic subdivision maintain the right triangular tilling for a 45 degree tile rotation, which is appropriated for quadtree and triplanar mapping. At the end of the subdivision each face is triangulated to comply with OpenGL 4.0, this compliance is guaranteed by a frame coherent hierarchy, which subdivides the triangles at a patch level [4]. Later on the GPU in the vertex shader we apply Jacobian spherical normalization on the coordinates to properly render a sphere (Figure1c) [10].

C. *Adapted Seamless Quadtree*

A slightly modified quadtree by Livny et al. [4] is used (Figure1c), we apply composite operations to enhance planetary rendering. Where we can trim unused data depending on the position of the observer. We also use a heightmap cache to speed up the node querying, to avoid unnecessary quadtree traversing. We also provide a custom query through a specified identifier, as such a node can be found by a criteria, *e.g.*:a node can be found by its position, or even through its altitude.

D. *Mesh interpolation, Displacement and Projective Mapping on the GPU*

We use six quadtrees mapped onto a sphere (Figure1c), on both the fBm noise and the border regions discontinuities can arise (see Figure2). To prevent such issue a downsampling is applied, we pick all four coordinates from a single quadtree node and pass it to the vertex shader, we apply a linear interpolation to obtain a single vertex out of the four coordinates creating a smooth surface in the process. Livny et al. [4] contemplates this downsampling, however it is done on the CPU, we adapted this approach to fully take advantage of the GPU capabilities.



(a) (b) (c)

Figure 2: Inconsistencies can appear (a) no tile-able noise, (b) different regions without interpolation, and (c) other view and its wireframe version.

In the vertex shader we downsample four vertices to find the central node, where we pass two four dimensional arrays (`P[]` and `N[]`) containing respectively the vertex coordinates and the base normals. If the terrain is a flat surface this solely algorithm would be enough, however since we are dealing with planetary terrains we need to take curvature into account, thus

we implemented a discrete form of projective mapping, where we take the planetary curvature into account.

We check if the vertices are in the central region of their respective boundaries, if so then a linear interpolation is performed, otherwise we perform a discrete projective mapping where we take the arc length over the planetary surface to compute the normals, then we parametrize the point by its angle *theta* to the z-axis and the angle *gamma* to the x axis, we later apply a transformation on the y axis with a intersection between the meridians of the z and x axis.

With this interpolation we achieve a discrete grid projection which has various advantages, the projection of each point in respect to its arc is computed in parallel on the GPU, the projection also attempts to optimize the object space triangle in such a way that the y projection is the same in each area of the screen space resulting in approximated pixel sized triangles. With the interpolation the number of rendered triangles are concise, which allows hardware specific optimization if required. Thus, the maximum LOD, the number of triangles each patch has, the patch size, and the patch multiplier can be tweaked if necessary.

However since we project the arc along a grid, computation of normals is a bit more complex, it must be computed on each frame as the mesh changes depending on the position of the observer. We ease this process by retrieving a part of the normals from the procedurally generated heightmap, and also taking advantage of the parallel capabilities of the GPU to compute the normals on a vertex based criterion.

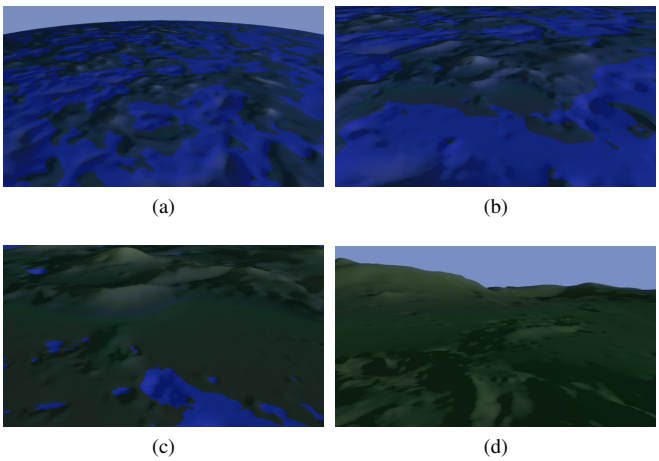
We calculate the normals by picking the coordinate through a triplanar mapping approach where we calculate the W offset by the difference of the UV channel along the procedurally generated heightmap texture, this approach is very handful since UVW coordinate system is appropriated for procedural maps. This approach well suitable for planets because the coordinate is actually a world coordinate and as each of the coordinate is sum up a vector relative to the planet surface is built, which is a normalization of each component between a range of -1 and 1, we get the transformed normal to the surface.

IV. Results

In this section we are going to present and discuss the results obtained from planetary generation using the variation of fBm noise by Quilez [7], presenting visual comparisons between other noise algorithms and a performance evaluation on a workstation equipped with Intel Core i7 3GHz, with 16GB of RAM and a Nvidia Titan X GPU. Our evaluation focuses on the visual quality and the performance compared to other procedural techniques, we compared both classical Perlin noise generated planet and state of art value noise generator [11]. In all experiments, unless otherwise stated, we use a planetary body with 1 earth radius (6,371 km). The patches we use in the application are 32 x 32 pixels wide. Since terrain is generated procedurally we do not use any terrain database. For atmospheric effects we used a simple scattering model where we apply a fog to each rendered fragment based on its depth

TABLE I. Framerate Performance

Our Approach				
Configuration		Time		RAM
Triangles	Quads	Altitude (kms)	FPS	Mb
126150	75	94	67	19.9
317898	189	20	61	22.1
Ridged noise				
Configuration		Time		RAM
Triangles	Quads	Altitude (kms)	FPS	Mb
—	—	—	70	17.4
—	—	—	64	21.3
Value noise				
Configuration		Time		RAM
Triangles	Quads	Altitude (kms)	FPS	Mb
166518	99	70	67	18.5
307806	183	22	62	21.7
Central point fBm				
Configuration		Time		RAM
Triangles	Quads	Altitude (kms)	FPS	Mb
—	—	—	58	22.6
—	—	—	44	32.2



(a) (b) (c) (d)

Figure 3: The Multifractal Brownian Motion by [7] is able to generate a planet on a single pass, with planet-wise landmarks such as canyons, valleys, mountain ranges and trenches

[12]. We also simulate a sky color by computing gradients in respect to the position of the observer [13].

The performance of our algorithm is summarized in Table I. The metrics we use to collect results are a) how many triangles are rendered, b) how many quads are necessary to handle those triangles, c) the observer altitude (in km) d) the frame rate, and e) how much RAM is used.

We also compared four noise generators for visual comparison. For each noise generator we compared basically two steps: 1) view from space and 2) atmospheric entry. The comparison between Ridged Noise, Value Noise, Central Point fBm and fBm by Quilez [7] is presented in Figure4.

The fBm algorithm by Quilez [7] procedurally generates a planet with planet-wise landmarks such as canyons, valleys,

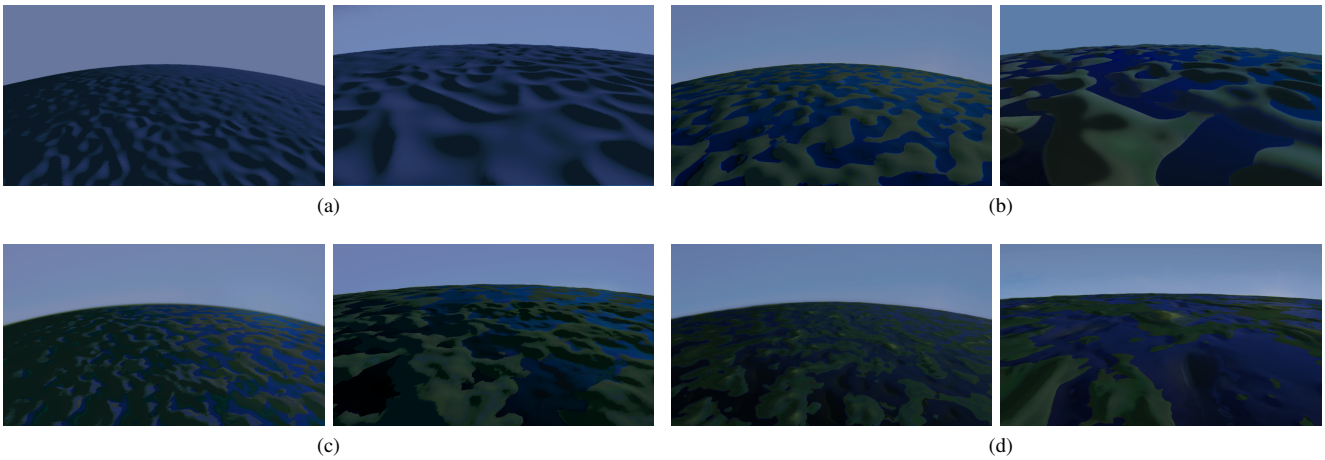


Figure 4: Comparison of different noise generators: (a) Ridged noise; (b) Value noise; (c) Central point fBm and (d) fBm by [7].

mountain ranges and trenches, since it is a fractal algorithm level of detail is added with granularity which could enhance the terrain and give a realistic behavior while the observer zooms into the planet as shown in Figure3.

V. Conclusion and Future Works

We presented a real-time procedural planetary terrain approach, it is able to generate planets with diverse spectrum of landmarks. The planetary terrain is rich in detail and due to the nature of fBm, it is able to produce a multi resolution terrain which can be refined using a LOD technique. In our experimental results we show that our approach is able to render large number of triangles in a real frame time.

As future works we intend to port the fBm algorithm into the vertex shader, to take advantage of the parallelism, we also plan to use other LOD based terrain rendering techniques, to test if there is a better data structure for planetary rendering, and to use hardware-supported tessellation to procedurally insert high frequency details on the surface.

Acknowledgment

The authors would like to thank The State of Bahia Research Foundation (FAPESB) for the sponsorship and NVIDIA for providing a TITAN X Graphics Card through its GPU Grant Program.

References

[1] M. Thöny, M. Billeter, and R. Pajarola, "Vision paper: The future of scientific terrain visualization," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '15. New York, NY, USA: ACM, 2015, pp. 13:1–13:4.

[2] P. Cozzi and K. Ring, *3D Engine Design for Virtual Globes*, 1st ed. A K Peters/CRC Press, 6 2011.

[3] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games (Computational Synthesis and Creative Systems)*. Springer, 2016.

[4] Y. Livny, Z. Kogan, and J. El-Sana, "Seamless patches for gpu-based terrain rendering," *The Visual Computer*, vol. 25, no. 3, pp. 197–208, Mar 2009.

[5] R. Kooima, J. Leigh, A. Johnson, D. Roberts, M. SubbaRao, and T. A. DeFanti, "Planetary-scale terrain composition," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 5, pp. 719–733, Sept 2009.

[6] J. D. Mahsman, C. J. White, D. S. Coming, and F. C. Harris, "Projective grid mapping for planetary terrain," *High Performance Computation and Visualization Lab*, 2011.

[7] I. Quilez, "Advanced value noise - fbm derivatives," 2017, retrieved April 10, 2018. [Online]. Available: http://www.iquilezles.org/www/articles/morenoise/morenoise.htm#fbm

[8] F. K. Musgrave, C. E. Kolb, and R. S. Mace, "The synthesis and rendering of eroded fractal terrains," *SIGGRAPH Comput. Graph.*, vol. 23, no. 3, pp. 41–50, Jul. 1989.

[9] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Commun. ACM*, vol. 25, no. 6, pp. 371–384, Jun. 1982.

[10] H. Parks, "Lecture notes - the jacobian for polar and spherical coordinates," February 1996. [Online]. Available: https://math.oregonstate.edu/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/jacpol/jacpol.html

[11] I. Parberry, "Modeling real-world terrain with exponentially distributed noise," *Journal of Computer Graphics Techniques Vol*, vol. 4, no. 2, pp. 1–9, 2015.

[12] E. Bruneton and F. Neyret, "Precomputed atmospheric scattering," *Computer Graphics Forum*, vol. 27, no. 4, pp. 1079–1086, jun 2008.

[13] J. Abad, "A fast, simple method to render sky color using gradients maps," 2006. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.7917&rep=rep1&type=pdf