Driftworld Tectonics 1.0: an overview

Adalbert Delong

19th May 2022

Driftworld Tectonics is a project written for Unity editor, used to create basic forms of planets. It utilizes ideas and methods described in an article by Yann Cortial et al. in 2019 [1]. Planets are created by reading basic template topology, performing a simplified tectonic simulation of the planet's crust and then exporting the raw data to customized binary files. Users can display various overlays of the planet during the simulation, as well as see the surface elevation mesh.

This documentation describes some basic theoretical framework, details of the simulation and the implementation. Hopefully it can be of use to anyone interested in this topic, who either just wants to play around with Driftworld or build their own projects.

Acknowledgements

I would like to thank Dr. Yann Cortial of the National Institute of Applied Sciences of Lyon for discussing his article. His answers to my various inquiries helped me decide the scope and form of Driftworld. I would also like to thank Dr. Daniel Meister of AMD Japan Co. Ltd. for his comments on the use of bounding volume hiearchy algorithms. Driftworld implements a part of an algorithm described in one of his publications [2]. More thanks to PROWAREtech for allowing me to include an adapted version of an example implementation of Mersenne Twister random number generator in the project.

I would like to thank Ben Golus for his help with UV texture mapping and his advice on texturing.

Special thanks to my friends Vilém and Matyáš and other members of our Discord server for discussing ideas and for their feedback.

The project was created using Unity Editor, lately in its version 2021.3.1f1. The C# code is kept in a MS Visual Studio Community 2022 project, image materials come from Unity Editor screenshots and online Geogebra projects. Documentation uses LaTeX in its TeX Live implementation.

Disclaimer

Over the past two years, Driftworld evolved both in terms of ideas and terms of implementation. The absolute majority of concepts beyond mathematics were completely new to me when the project started, so the code changed often and many times was almost completely rewritten as I learned. Some older parts remained which can cause a correct impression of inhomogeneity. I do not claim the implementation is flawless and although a lot of the shaky cases were accounted for, some unforseen mistakes may and probably do remain.

I would like to ask anyone using the project to tolerate possible mistakes. There is more work to be done and I would be grateful for feedback.

Thank you for your consideration.

Contents

1 Introduction

Driftworld Tectonics is a Unity project created for use in the Unity editor. The entirety of interactivity is within the editor GUI and the project has no meaningful executable scene. Any feedback is in a console log and the state of the planet is observed within the static scene rendering. This is the most obvious difference from the implementation in the original article from which Driftworld draws inspiration - simulation described in the article offers interactivity while the simulation is running [1].

The workflow follows Cortial et al. in a lot of details, although experience and chosen software tools pose several restrictions. At first a Delaunay triangulation mesh is imported from prepared binary files. Then a set of tectonic plates is created by partitioning said mesh. The planet evolution is performed in repeated tectonic steps. Every step deals with plate subduction, possible continental collision, new crust creation because of diverging ocean plates, slab pull due to subduction influence, erosion and crust damping, and finally, rifting plates. At any time the current state can be saved as a binary file for further use. This follows the original article [1]. Driftworld, however, differs in two rather important steps: continental collisions are always plate-wide and plate rifting follows somewhat different probability mechanics. This is mainly for fine-tuning and can change in future updates, as these changes further simplify an already simplified model and were done as a saving grace from implementation difficulties.

The output of Driftworld is binary planet data with varying resolution of sphere sampling. Provided data are: crust age, elevation, plate assignment, crust thickness and orogeny. The binary file keeps the original topology for easier manipulation. The user can also at any time export the current texture overlay as a PNG image.

This documentation serves both as a user's manual and a quick introduction into the problematic. Section 2 defines basic terms, mathematical objects and their properties. Section 3 follows with details of the used simplified tectonic model. The actual implementation with neccessary details and context are discussed in the section 4. As an important part, performance of the simulation and related issues are the topic of section 5. We conclude the status quo of the project in section 6.

1.1 Motivation

Procedural terrain generation is an important part for a number of computer games [3]. Usually, these games employ random generators to increase variety on a theme, such as a map layout. Indeed, in my subjective opinion a player's experience is greatly enriched by variety, especially in the game environment. This comes with an apparent caveat that purely procedural generation may lack the sense of creativity, leading to mundanely repeating patterns [4].

With the onset of newer technologies (e. g. increased GPU power), we are able to perform more computationally-intensive tasks. When it comes to the terrain generation, even a regular user without access to high-tier hardware can try more sophisticated alternatives to simpler algorithms. Arguably, more realistic worlds bring the feeling of familiarity to the experience. If we can create a more realistic, yet still random map/world/neighbourhood, the possibilities are endless.

Following thoughts are purely my personal view. As a life-long video games fan, I have always gravitated towards story-telling games, especially those taking place in an open world. Among these, I'd like to mention Baldur's Gate series, The Elder Scrolls series, Might & Magic series, Fallout 4 and Mass Effect series. At the same time, I have been also drawn to grand building games taking place in complex worlds. Transport Tycoon or Caesar III and its modern re-implementation Augustus [5] were a heavy influence, lately Factorio or Rimworld. Rimworld stands apart in its uniqueness, as it can be understood rather as a story generator than a game [6]. In large part, the idea of Driftworld came from the works of J. R. R. Tolkien and watching the 1997-2007 Stargate series - the series' take on mythology context in human societies in particular.

Epic stories build on cohesion. In this regard, countless debates take place about details. It takes a great deal of time to create viable environment to match an idea for a story, especially if that story is told over long periods of time. Driftworld aspires to one thing: help create a platform in which stories can take place. First step is this project – to create a rough map.

2 Spherical geometry & topology

The most fundamental object with which Driftworld Tectonics works is a mesh of a sphere in the 3D Euclidean space. For simplicity, we assume the sphere is a unit sphere centered on the origin unless stated otherwise. Because of the spherical nature of the project, several (arguably) uncommon mathematical concepts are described in this section – such as vertex sampling, triangulation, transformations or bounding volume hiearchies.

2.1 Unity coordinate system

Unity uses a left-handed coordinate system with the x axis pointing to the right, y axis pointing upwards and z axis pointing forward (see Figure 1). This is reflected in the scenes – nevertheless, the mathematical expressions of vectors themselves are identical to a standard right-handed coordinate system, i. e. the following holds for the basis:

$$\mathbf{e}_x \times \mathbf{e}_y = \mathbf{e}_z$$

All implementations must be aware of the fact that the cross product expressions do not distinguish between right-handed and left-handed. It is simply a matter of axes display, where visually 'switching' axes y and z alternates between left-handedness and right-handedness. In the left-handed coordinate system, right-hand rule of cross product shows the inverse final direction of the cross product.

There are several ways to rotate points, vectors or whole transformations. For clarity, let us assume a 3-dimensional vector $\mathbf{u}$ that is to be rotated. We define a rotation unit vector $\mathbf{n}$ and an angle ϕ by which we rotate $\mathbf{u}$ so that $\mathbf{u}$ rotates by ϕ within a plane to which $\mathbf{n}$ is normal. We also assume that the rotation plane passes through the origin. Then from the perspective of a sundial (with $\mathbf{n}$ being the gnomon) $\mathbf{u}$ rotates *clockwise* for positive ϕ (Figure 1). This holds for all relative rotations.

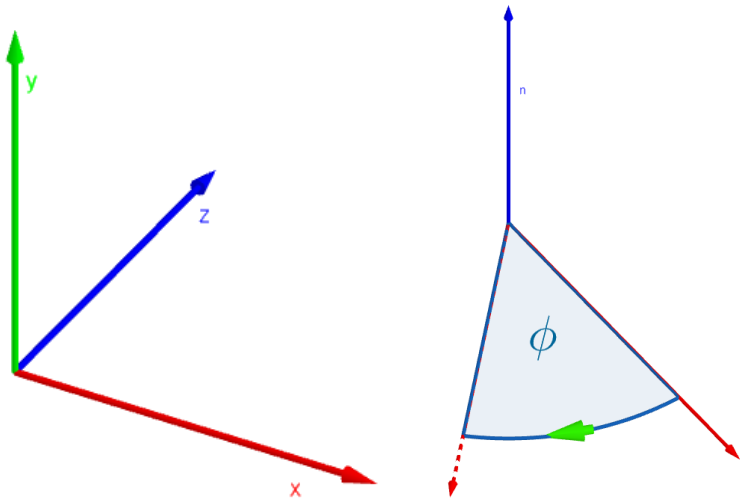


Figure 1: Unity coordinate system

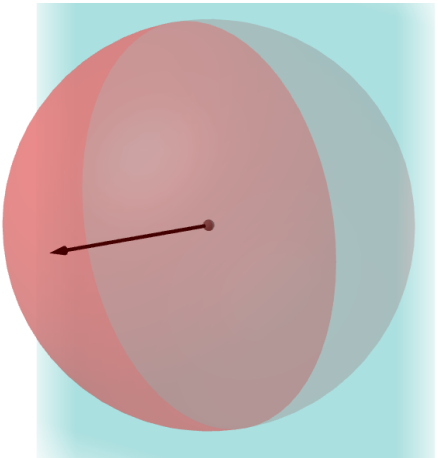
2.2 Sectional planes and great circles

Sphere can have any number of sectional planes, i. e. planes that have some non-empty intersection with the sphere. A subset of planes passing the center of the sphere will be called *sectional central planes* (Figure 2a). Any sectional central plane ρ is characterized by some non-zero normal vector $\mathbf{n}_\rho$ and for any point on the plane represented by their position vector $\mathbf{x}$ it holds that

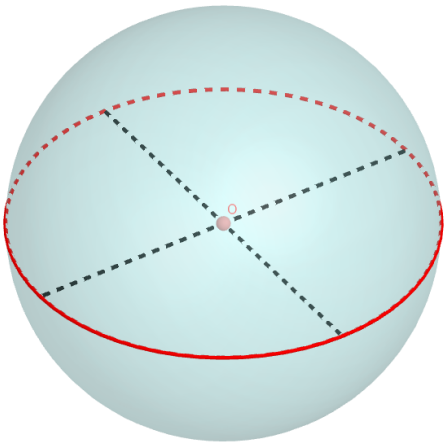
$$\mathbf{n}_\rho \cdot \mathbf{x} = 0$$

This is synonymous to the fact that any vector lying within a plane passing the origin is perpendicular to the normal vector of the plane. The dot product on the left side of the equality is also important because given a specific normal vector we can decide on which side is any vector $\mathbf{x}$ *outside* the plane – simply take the sign of the dot product, vector on the side of the normal vector will result in a positive dot product value with $\mathbf{n}_\rho$, negative otherwise.

An important object on the surface of a sphere is a great circle. It is any circle that shares its center and radius with the sphere (Figure 2b). It is also the intersection of a plane passing the center of the sphere with its surface.



(a) central plane

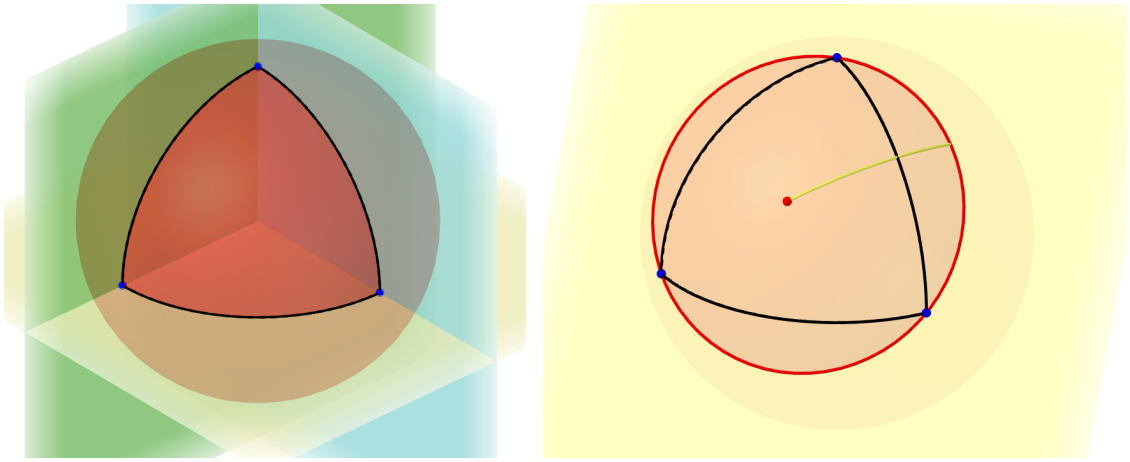


(b) great circle

Figure 2: Sphere section by plane

2.3 Spherical triangles

Any three points on the surface of a sphere that do not lie on a single great circle form a *spherical triangle* (Figure 3). This is the fundamental concept behind many of the computations in the project.



(a) planes intersections (b) circumcircle and sectional plane

Figure 3: Spherical triangle

Strictly speaking, there are two triangles defined by such three points. The complement of any spherical triangle with respect to the sphere surface is also a spherical triangle, albeit one of the two is unintuitive. To clarify, we try to formally construct a spherical triangle definition.

Geometrically speaking, a spherical triangle is a region bounded by three arcs of great circles [8]. We denote the surface of a unit sphere $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3 : ||\mathbf{x}|| = 1\}$. Given three linearly independent point vectors $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathcal{S}$ we calculate three normal vectors:

$$\mathbf{n}_\rho = \mathbf{a} \times \mathbf{b},$$

$$\mathbf{n}_\sigma = \mathbf{b} \times \mathbf{c},$$

$$\mathbf{n}_\tau = \mathbf{c} \times \mathbf{a}.$$

These vectors define planes ρ, σ, τ so that

$$\rho = \{\mathbf{x} \in \mathbb{R}^3 : \mathbf{n}_\rho \cdot \mathbf{x} = 0\}$$

$$\sigma = \{\mathbf{x} \in \mathbb{R}^3 : \mathbf{n}_\sigma \cdot \mathbf{x} = 0\}$$

$$\tau = \{\mathbf{x} \in \mathbb{R}^3 : \mathbf{n}_\tau \cdot \mathbf{x} = 0\}$$

intersections $\rho \cap \mathcal{S}, \sigma \cap \mathcal{S}, \tau \cap \mathcal{S}$ are then great circles p, s, t that always pass two of the points $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Because of this, each one is divided by them into two arcs. There is only one triplet of arcs connected by $\mathbf{a}, \mathbf{b}$ or $\mathbf{c}$ that forms a region boundary on $\mathcal{S}$ (Figure 3a). Problem is, there are two such regions. We would like our definition to be unambiguous. Because of the way we computed the normal vectors, there is always a subset of $\mathcal{S}$ so that the dot products of any of its elements with $\mathbf{n}_\rho, \mathbf{n}_\sigma$ and $\mathbf{n}_\tau$ all have the same sign.

2.4 Vertex sampling

Because of memory restrictions, sphere surface data is represented as a set of sampled points. We can identify these points as position vectors $\mathbf{u}_i$ from the global origin to sample points, resulting in

a sequence $U = (\mathbf{u}_i)_{i=0}^{N-1}, \mathbf{u}_i \in \mathbb{R}^3, ||\mathbf{u}_i|| = 1$. Samples are therefore three-dimensional normalized vectors. Driftworld uses spherical Fibonacci sampling [7]. To get the sequence U, another sequence $F = (\mathbf{f}_i)_{i=0}^{N-1}$ is first computed, using the following definition:

$$\mathbf{f}_i = (\phi_i, z_i), \phi_i \in \mathbb{R}, z_i \in \mathbb{R},$$

$$\phi_i = 2\pi \left[\frac{i}{\Phi} \right],$$

$$z_i = 1 - \frac{2i+1}{N}.$$

$[x]$ denotes the fractional part of x, Φ is the golden ratio $\Phi = \frac{\sqrt{5}+1}{2}$. The values of $\mathbf{f}_i$ actually lie on a spiral on the surface of a cylinder with the radius of 1 and the height of 2 [7]. U is finally obtained by mapping $\mathbf{f}_i$ values to a unit sphere:

$$\mathbf{u}_i = (\sin(\arccos(z_i)) \cdot \cos\phi_i, z_i, \sin(\arccos(z_i)) \cdot \sin\phi_i)$$

Note that this mapping reflects Unity's axes orientation and the first and the last samples of U do not fall exactly on the poles.

3 Simple tectonic model

4 Implementation & data model

4.1 Rendering

4.2 GPU Computing

4.3 Use

5 Performance & problems

6 Conclusion

6.1 Continuation of work

7 References

[1] Cortial, Y., Peytavie, A., Galin, E. and Guérin, E. (2019), Procedural Tectonic Planets. *Computer Graphics Forum*, 38: 1-11. https://doi.org/10.1111/cgf.13614

[2] D. Meister and J. Bittner, "Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 3, pp. 1345-1353, 1 March 2018, doi: 10.1109/TVCG.2017.2669983.

[3] Wikipedia contributors. (2022, March 20). List of games using procedural generation. In *Wikipedia, The Free Encyclopedia*. Retrieved 06:20, May 19, 2022, from https://en.wikipedia.org/w/index.php?title=List_of_games_using_procedural_generation&oldid=1078237416

[4] Brogan, J. (2016, October 5). *The Daggerfall Paradox*. SLATE. Retrieved May 19, 2022, from https://slate.com/technology/2016/10/the-paradox-of-procedurally-generated-video-games.html

[5] Keriew, *Augustus*, (2021), GitHub repository, https://github.com/Keriew/augustus

[6] Game Developer Conference [GDC]. (2019). *RimWorld: Contrarian, Ridiculous, and Impossible Game Design Methods* [Video]. YouTube. https://www.youtube.com/watch?v=VdqhHKjepiE

[7] Keinert, Benjamin & Innmann, Matthias & Sänger, Michael & Stamminger, Marc. (2015). Spherical Fibonacci Mapping. *ACM Transactions on Graphics*. 34. 1-7. 10.1145/2816795.2818131.

[8] Palmer, C. I., & Leigh, C. W. (1934). *Plane and spherical trigonometry*. New York: McGraw-Hill Book Company, Inc.

List of Figures

List of Tables

A Project parameters