

Driftworld 1.0: an overview

Adalbert Delong

19th May 2022

Driftworld is simply 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.

Project Driftworld is licensed under a [Creative Commons Attribution 4.0 International License](#).

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 hierarchy 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 L^AT_EX 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 unforeseen mistakes may and probably do remain.

I would like to ask anyone using Driftworld 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	4
1.1	Motivation	4
2	Spherical geometry & topology	5
3	Simple tectonic model	5
4	Implementation & data model	5
4.1	Rendering	5
4.2	GPU Computing	5
4.3	Use	5
5	Performance & problems	5
6	Conclusion	5
6.1	Continuation of work	5
7	References	6
	List of Figures	7
	List of Tables	7
A	Project parameters	8

1 Introduction

Driftworld is a Unity project implemented 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 is 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 necessary 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, 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

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>

List of Figures

List of Tables

A Project parameters