Driftworld Tectonics 1.0: an overview

Adalbert Delong

2nd August 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 and support.

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, Geogebra projects, Blender and Dia. 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. Although the text follows almost a textbook-like mathematical structure, a lot of the formulations and conclusions lack correct proof. Some reasoning is made to carry a point, but meticulous readers are left to their own devices.

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.

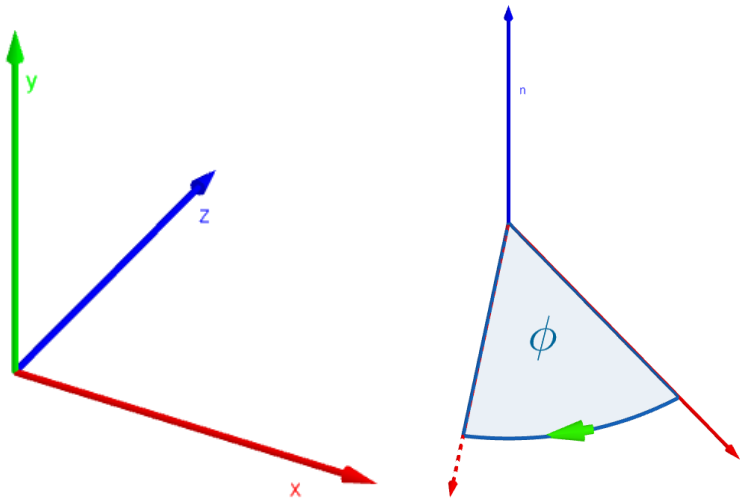


Figure 1: Unity coordinate system

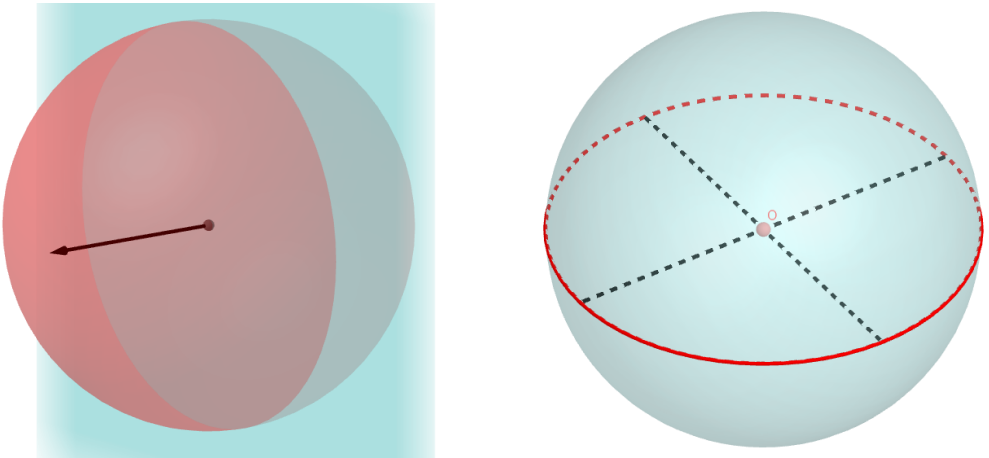
2.2 Sectional planes and great circles

Sphere can have any number of sectional planes, i. e. planes that have some non-empty intersection with the sphere. 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.

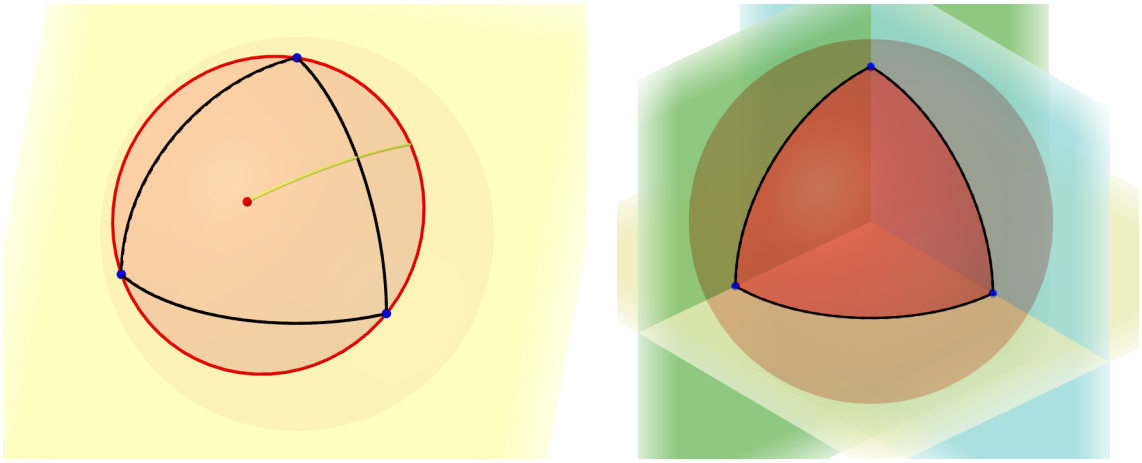


(a) central plane (b) great circle

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. However, strictly speaking, there are two triangles defined by such three points. The closure of the complement of any spherical triangle with respect to the sphere surface is also a spherical triangle, albeit one of the two is unintuitive as it is larger than half of the sphere surface area. To get around this, we construct somewhat narrower class of spherical triangles so that any three valid points define a triangle unambiguously.



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

Figure 3: Spherical triangle

We denote the surface of a unit sphere $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3 : ||\mathbf{x}|| = 1\}$. Given a triplet of three linearly independent point vectors $(\mathbf{a}, \mathbf{b}, \mathbf{c}) \in \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ (called *vertices*)[1], we can construct a vector $\mathbf{n}_\lambda$ normal to some sectional plane λ cutting off a spherical cap (Figure 3a):

$$\mathbf{n}_\lambda = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})$$

$$\forall \mathbf{x} \in \lambda : \mathbf{n}_\lambda \cdot \mathbf{x} + d_\lambda = 0$$

We can calculate d_λ by assigning e. g. $\mathbf{x} = \mathbf{a}$ and solving the plane equation with respect to d_λ, but that will not be neccessary. We impose a further requirement $\mathbf{n}_\lambda \cdot \mathbf{a} > 0$. This is not always true, in which case it can be ensured by swapping any two vertices in the triplet and recalculating $\mathbf{n}_\lambda$. This means that all three vertices and their circumcenter are all 'on one side' of the sphere. In Unity coordinate system, this also means that for an outside observer, the vertices are oriented *clockwise* on the sphere surface.

Spherical circumcircle l is a set of points on a sphere that has constant spherical distance from a single point $\mathbf{c}_\mathcal{T} \in \mathcal{S}$ called *circumcenter*. Equivalently, we can substitute dot product for distance:

$$\exists t \in \mathbb{R} : (\forall \mathbf{x} \in l : \mathbf{c}_\mathcal{T} \cdot \mathbf{x} = t)$$

Since $\mathbf{n}_\lambda \cdot \mathbf{a} = \mathbf{n}_\lambda \cdot \mathbf{b} = \mathbf{n}_\lambda \cdot \mathbf{c} > 0$, we know that some scalar multiple of $\mathbf{n}_\lambda$ is the circumcenter for the vertices. In fact, there is only one possible circumcircle for all three vertices, which is the intersection $\mathcal{S} \cap \lambda$. We easily find the circumcenter and the circumradius r_l as[2]

$$\mathbf{c}_\mathcal{T} = \frac{\mathbf{n}_\lambda}{||\mathbf{n}_\lambda||}, r_l = \arccos(\mathbf{c}_\mathcal{T} \cdot \mathbf{a})$$

[1] Linear independence of unit vectors is equivalent to the condition that the vectors do not lie on a single great circle.

[2] We have to keep in mind that on a unit sphere, central angle and spherical distance are identical, barring formal dimension.

Previous reasoning allows us now to test if some point $\mathbf{x} \in \mathcal{S}$ is inside a spherical triangle $\mathcal{T} \subset \mathcal{S}$ with clockwise-oriented vertices $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Geometrically speaking, a spherical triangle is a region bounded by three arcs of great circles [7]. 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 that always pass two of the vertices. Because of this, each one is divided by them into two arcs. There is only one triplet of arcs connected by the vertices that forms a meaningful region boundary on $\mathcal{S}$ (Figure 3b). There are two such regions but we already bypassed this problem by ensuring orientation. We test the point $\mathbf{x}$ against following condition:

$$\mathcal{T} = \{\mathbf{x} \in \mathcal{S} : \mathbf{n}_\rho \cdot \mathbf{x} \geq 0, \mathbf{n}_\sigma \cdot \mathbf{x} \geq 0, \mathbf{n}_\tau \cdot \mathbf{x} \geq 0\}$$

This simply tells us that $\mathbf{x}$ is inside the spherical triangle $\mathcal{T}$ when it is on the surface of the sphere and also on one specific side of all three planes ρ, σ, τ. This definition does not encompass all possible spherical triangles on a sphere, but it allows us to properly test the properties of any triangles used in reasonable spherical meshes.

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 $U = (\mathbf{u}_i)_{i=0}^{N-1}, \mathbf{u}_i \in \mathcal{S}$. Samples are therefore three-dimensional normalized vectors. Driftworld uses spherical Fibonacci sampling [8]. To get the sequence U, another sequence $F = (\mathbf{f}_i)_{i=0}^{N-1}$ is first computed, using the following definition:

$$\mathbf{f}_i = (\phi_i, z_i), \phi_i \in [0, 2\pi), z_i \in (-1, 1),$$

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

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

$[x]$ 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 [8]. U is finally obtained by mapping $\mathbf{f}_i$ values to $\mathcal{S}$:

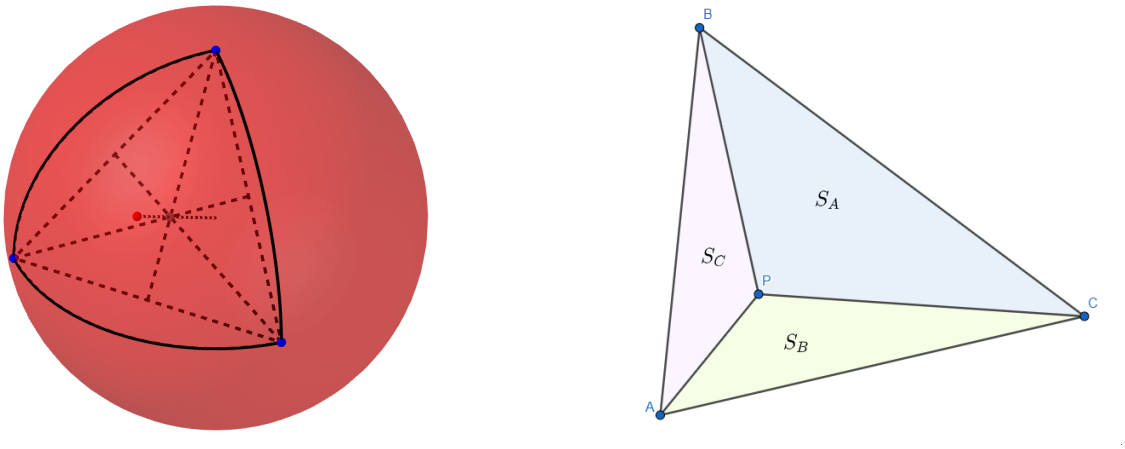
$$\mathbf{u}_i = (\sin(\arccos(z_i)) \cdot \cos\phi_i, z_i, \sin(\arccos(z_i)) \cdot \sin\phi_i)$$

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.

2.5 Centroids, data values and barycentric interpolation

Driftworld often makes computations for points inside sperical triangles - notably, it uses triangle centroids to evaluate triangle neighbours. Calculating these points is not a trivial procedure and although for a long time there have been methods to do so, it would be too resource-consuming when performed on a larger scale. To save computation time, we assume that all evaluated triangles are nearly planar, i. e. their *triangle excess* is negligible (Legendre's Theorem[3] [9]). We calculate the centroid of a triangle by simply normalizing the sum of its vertices (Figure 4a):

$$\mathbf{b}_{\mathcal{T}} = \frac{\mathbf{a}+\mathbf{b}+\mathbf{c}}{||\mathbf{a}+\mathbf{b}+\mathbf{c}||}$$



(a) sectional triangle centroid

(b) barycentric coordinates

Figure 4: Centroid geometry

To store crust data, we must assign values to points on the sphere. These values may be of different types or have different meaning. Formally, we denote a sequence of arbitrary sets $C = (V_i)_{i=0}^{n-1}$, where n is the number of different values assigned to a point and each V_i is a specific set of possible values. The system of all possible value combinations is then a cartesian product of these value sets:

$$V = \prod_{i=0}^{n-1} V_i$$

Stored data can then be defined as a map:

$$h : U \to V$$

$$h_i : U \to V_i$$

We store data only for sphere samples because of limited memory. Other values will be computed as needed using *barycentric interpolation* [10].

[3] This theorem is also known as Saccheri-Legendre theorem.

Now it comes to the following problem: how to compute values anywhere on $\mathcal{S}$? We are effectively looking for some domain extension, since $U \subset \mathcal{S}$:

$$h' : \mathcal{S} \to V, \forall \mathbf{u} \in U : h'(\mathbf{u}) = h(\mathbf{u})$$

$$h'_i : \mathcal{S} \to V_i, \forall \mathbf{u} \in U : h'_i(\mathbf{u}) = h_i(\mathbf{u})$$

Given some arbitrary point $\mathbf{x} \in \mathcal{S}$, we start with an assumption that $\mathbf{x}$ is found inside some spherical triangle $\mathcal{T}$ with negligible triangle excess and vertices $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\} \subset U$ for which we already know the values of h. We would like $h'(\mathbf{x})$ to be computed 'fairly', i. e. the closer $\mathbf{x}$ is to some vertex, the more influence the vertex value should have on $h'(\mathbf{x})$. A good start might be in analogy with a political voting system based on area. If the population is homogeneous, any region vote is weighted by its area and transitionally, by its population.

Let P be some point within a triangle ABC (Figure 4b). This point is represented by a point vector $\mathbf{p} \in \mathcal{S}$. As stated earlier, we assume all points lie nearly on the same plane. If we construct three triangles PBC, APC and APB, the triangle ABC will be divided into three regions, each corresponding to their oposite vertex of ABC. The closer P is to any of the vertices, the larger the corresponding triangle area is. Total area sum of the three triangles is equal to the area of ABC. Therefore, we can use these triangle areas as weights for interpolating values at P – we only need to find the respective areas of S_A, S_B, S_C and S_{ABC}. This can be done using cross product:

$$S_A = \frac{|(\mathbf{b} - \mathbf{p}) \times (\mathbf{c} - \mathbf{p})|}{2}$$

$$S_B = \frac{|(\mathbf{c} - \mathbf{p}) \times (\mathbf{a} - \mathbf{p})|}{2}$$

$$S_C = \frac{|(\mathbf{a} - \mathbf{p}) \times (\mathbf{b} - \mathbf{p})|}{2}$$

$$S_{ABC} = \frac{|(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})|}{2}$$

Since $S_A + S_B + S_C = S_{ABC}$, we can define normalized weights u, v, w called *barycentric coordinates*:

$$u = \frac{|(\mathbf{b} - \mathbf{p}) \times (\mathbf{c} - \mathbf{p})|}{|(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})|}$$

$$v = \frac{|(\mathbf{c} - \mathbf{p}) \times (\mathbf{a} - \mathbf{p})|}{|(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})|}$$

$$w = \frac{|(\mathbf{a} - \mathbf{p}) \times (\mathbf{b} - \mathbf{p})|}{|(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})|}$$

It is easy to confirm that $u + v + w = 1$.

There are basically two types of values interpolated in the project – real values and categories. Real value interpolation is straightforward:

$$h'_i(\mathbf{p}) = uh_i(\mathbf{a}) + vh_i(\mathbf{b}) + wh_i(\mathbf{c})$$

Categories are simply assigned to $\mathbf{p}$ according to the largest weight:

$$u = \max(\{u, v, w\}) \Rightarrow h'_j(\mathbf{p}) = h_j(\mathbf{a})$$

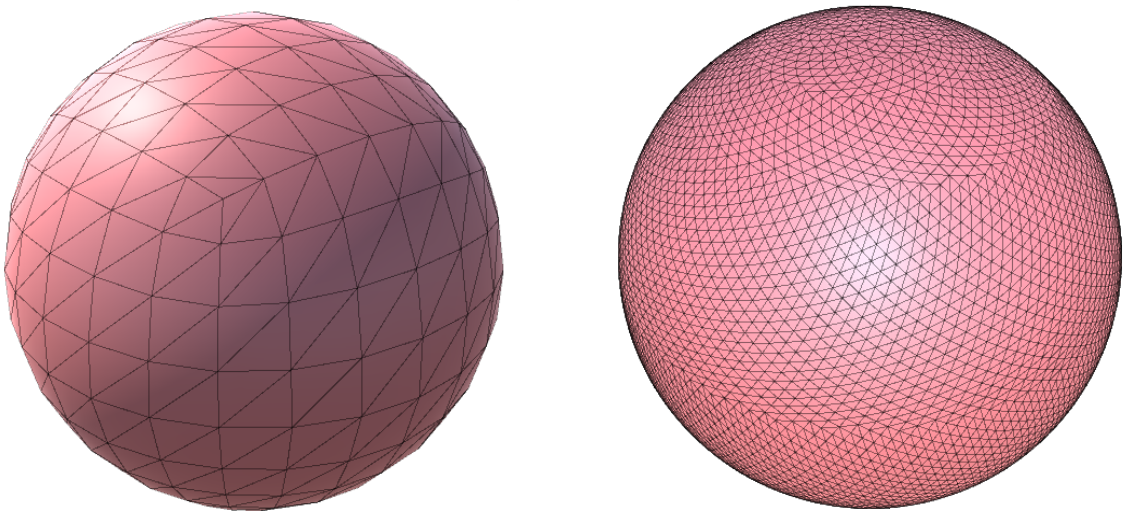
$$v = \max(\{u, v, w\}) \Rightarrow h'_j(\mathbf{p}) = h_j(\mathbf{b})$$

$$w = \max(\{u, v, w\}) \Rightarrow h'_j(\mathbf{p}) = h_j(\mathbf{c})$$

This effectively draws a Voronoi map according to categories [11].

2.6 Spherical mesh and Delaunay triangulation

There is a basic sphere mesh, provided by Unity (Figure 5a). It is like a detailed cubic mesh, projected onto a sphere. However, for finer terrain details, a much more detailed mesh is needed, preferably with uniform triangles (Figure 5b).



(a) Unity sphere mesh

(b) Delaunay mesh

Figure 5: Spherical meshes

Definition and description of a 3D mesh is way beyond the scope and purpose of this document. In this context, it is simply an approximation of the sphere surface. All samples U are vertices connected into triangles so that the whole sphere is covered by them without any gaps. For Driftworld, a set of prepared meshes is provided, created by *Delaunay triangulation*[4] [12].

When interpolating surface data such as elevation, it is important that reasonable samples are used for the interpolation. Calculating elevation in a mountain range from a triangle with vertices too far apart may result in meaningless artifacts. Furthermore, the earlier mentioned requirement that the triangles are nearly planar would be undermined by extreme spherical triangles which exhibit considerable excess. It stands to reason that triangles used in the mesh should be as regular as possible. This is the goal and result of a Delaunay triangulation. There is a number of algorithms performing the triangulation on a plane [13]. Since a sphere has a closed mesh, an adaptation is needed.

Delaunay meshes for Driftworld use an algorithm which originally triangulates a set of random samples [14]. Because U is ordered, the initial tetrahedron is somewhat difficult to construct, especially because of the requirements imposed on a spherical triangle – in a large number of cases at least one triangle had a circumcircle larger than a great circle. For this reason, the initial structure was set to be a nearly regular octahedron with vertices assigned by a brute-force look-up. Other than that, the algorithm follows the article [14].

[4]The triangulation algorithm is not a part of Driftworld. The meshes were actually constructed in a separate tool written in C++ and then added as data files to Driftworld Tectonics repository.

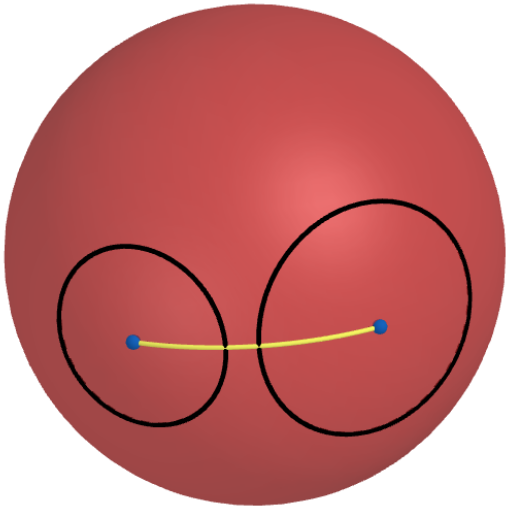
2.7 Collisions

The tectonic model in Driftworld computes many interactions on a regular basis and these computations must be as efficient as possible. There are two basic collisions used for evaluating interactions – a collision of two circles and a collision of two triangles. To clarify, in both cases we only need to answer the question whether the two objects have a non-empty intersection – not to fully classify the intersection. Algorithms for both collisions are fairly simple and the spherical geometry actually helps in the case of triangle collisions.

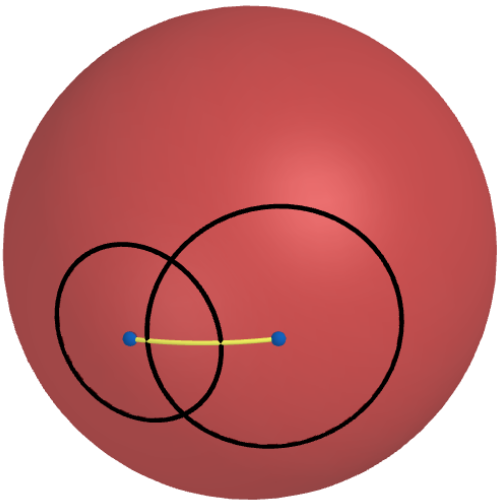
The relative position of two circles $k, l \subset \mathcal{S}$ is governed by several parameters. Each circle has a circumcenter $\mathbf{c} \in \mathcal{S}$ and a radius $r > 0$. We consider full circles to determine the intersection:

$$\forall \mathbf{x} \in \mathcal{S} : \arccos(\mathbf{x} \cdot \mathbf{c}_k) \leq r_k \Rightarrow \mathbf{x} \in k$$

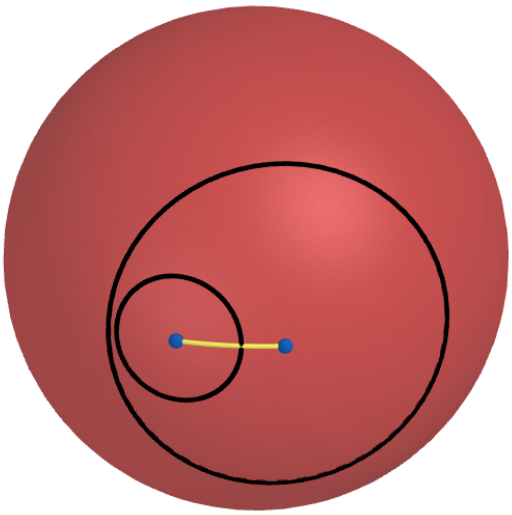
This means that there is only one case of relative circle position that has an empty intersection: disjoint circles. The case of one circle lying inside another has an intersection indentical to the inside circle. Three major cases of the relative positions of circles are seen in Figure 6.



(a) disjoint circles



(b) circles intersecting at two points



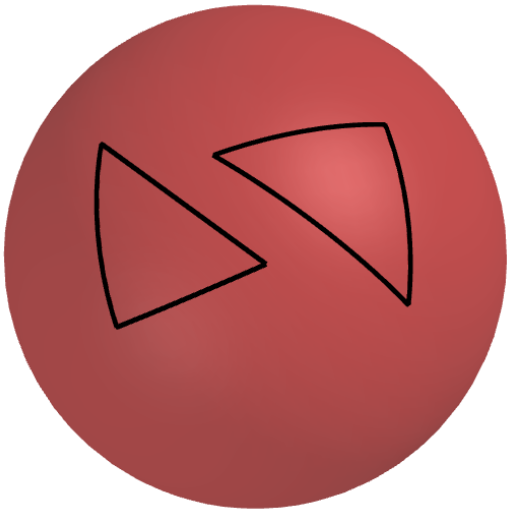
(c) circle lying inside another

Figure 6: Relative positions of two circles

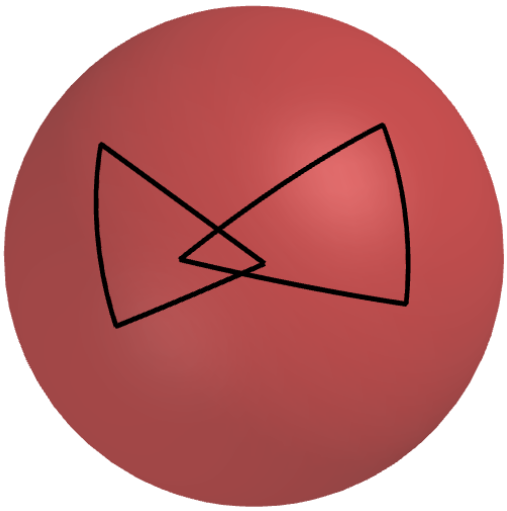
It is therefore easy to decide whether two circles collide or not - circles not colliding have a spherical distance larger than the sum of their radii:

$$k \cap l = \emptyset \iff \arccos(\mathbf{c}_k \cdot \mathbf{c}_l) > r_k + r_l$$

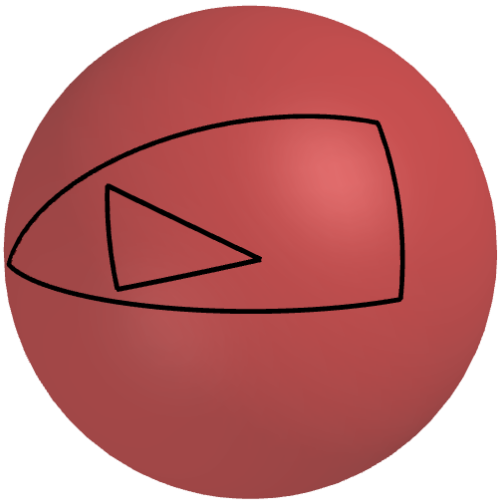
In case of triangles the collision is more complex. There are four major cases of the relative position of two spherical triangles - three similar to the case of circles and one specific (see Figure 7). The second and the third case can be resolved by determining whether any point of one triangle lies within the other (see subsection 2.3). The first and the fourth require a more thorough test.



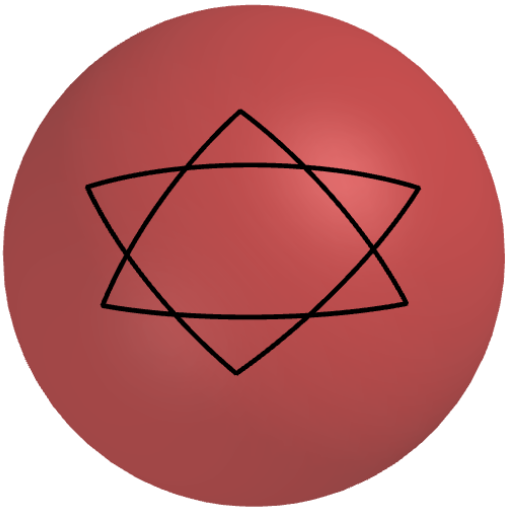
(a) disjoint triangles



(b) triangles intersecting at two points



(c) triangle lying inside another



(d) Star of David

Figure 7: Relative positions of two triangles

If neither of the two triangles contain a vertex of the other one, we have to decide if any two edges intersect. Consider two edges defined by pairs of non-identical vertices $(\mathbf{a}_1, \mathbf{a}_2)$ and $(\mathbf{b}_1, \mathbf{b}_2)$. These define planes within which lie their respective great circles. The planes have normal vectors $\mathbf{a}_1 \times \mathbf{a}_2$ and $\mathbf{b}_1 \times \mathbf{b}_2$. Since the planes are central, Any intersection must be along the vector $(\mathbf{a}_1 \times \mathbf{a}_2) \times (\mathbf{b}_1 \times \mathbf{b}_2)$. There are two intersections in $\mathcal{S}$ and we consider the one maximizing the dot product with $\mathbf{a}_1$ (same hemisphere). We denote such intersection $\mathbf{i}$. The final test simply decides if $\mathbf{i}$ lies on both segments or is somewhere else on the great circles. This can be done by testing dot products, as $\mathbf{i}$ must be closer to both vertices than the distance of the vertices for both segments:

$$(\mathbf{i} \cdot \mathbf{a}_1 > \mathbf{a}_1 \cdot \mathbf{a}_2) \wedge (\mathbf{i} \cdot \mathbf{a}_2 > \mathbf{a}_1 \cdot \mathbf{a}_2) \wedge (\mathbf{i} \cdot \mathbf{b}_1 > \mathbf{b}_1 \cdot \mathbf{b}_2) \wedge (\mathbf{i} \cdot \mathbf{b}_2 > \mathbf{b}_1 \cdot \mathbf{b}_2)$$

If any two segments of the two triangles intersect, the triangles must by the sign analysis have non-empty intersection and therefore collide. If all tests are negative, the triangles are disjoint.

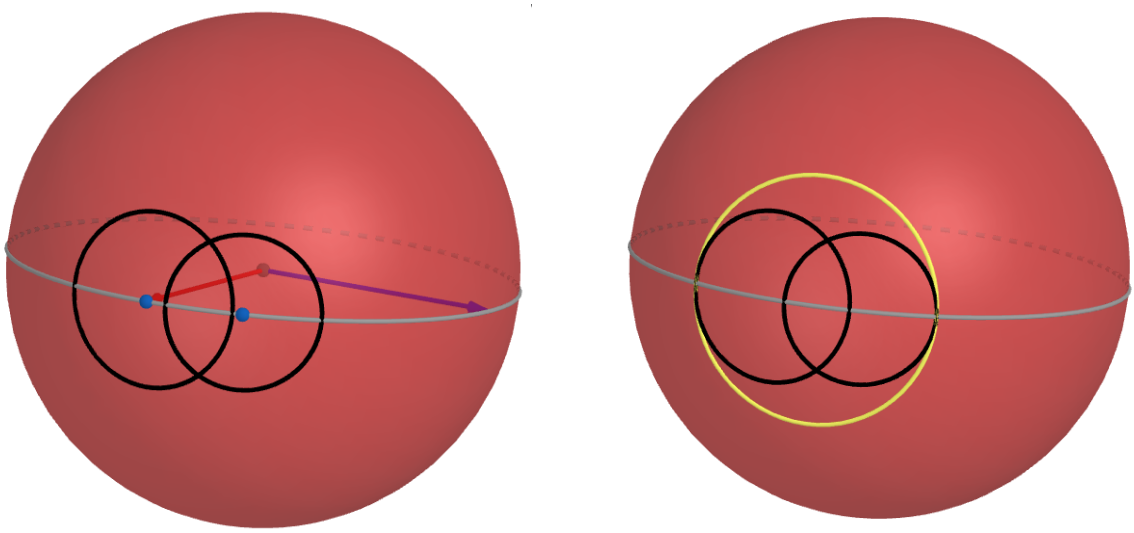
2.8 Merging of spherical circles

Because of the need to build bounding volume hiearchies (see subsection 4.3), there is a task of finding a suitable circle l which has the smallest area possible and which contains two other given circles m, n (see Figure 8b). The center of l must lie on a great circle L containing both centers of the circles. This means we have to find a center and radius of l so that it only touches either both of the two circles m, n or one in case one circle is within the other. The algorithm falls in at least one of the following cases: concentric circles, circles with opposite centers, one circle contained in the other, circles intersecting at two points or disjoint circles. The goal is to determine which case has the deciding influence on the position of the center $\mathbf{c}_l$ and the radius r_l except in the case of concentric circles, where l is found easily. In case of concentric circles the solution is:

$$\mathbf{c}_l = \mathbf{c}_m, r_l = \max(r_m, r_n)$$

Otherwise we find a suitable local basis in which we can easily parametrize all relevant points (Figure 8a). One circle center will be identical to the base vector:

$$\mathbf{e}'_x = \mathbf{c}_m$$



(a) local basis (b) circles merged

Figure 8: Merging of two circles

Computation of the second base vector $\mathbf{e}'_z$ depends on whether the circles are directly opposite, i. e. they have centers opposite on the sphere. If so, it is any vector perpendicular to $\mathbf{e}'_x$ since the centers lie on inifinitely many great circles. If not, it it can be found with a double cross product:

$$\mathbf{e}'_z = \frac{(\mathbf{c}_m \times \mathbf{c}_n) \times \mathbf{c}_m}{||(\mathbf{c}_m \times \mathbf{c}_n) \times \mathbf{c}_m||}$$

There are three variables we need to compute now. First is the distance d_{mn} between $\mathbf{c}_m$ and $\mathbf{c}_n$:

$$d_{mn} = \arccos(\mathbf{c}_m \cdot \mathbf{c}_n)$$

Second is the central angle $\Delta\phi$ of an arc running between $\mathbf{c}_m$ and the center of the merged circle. This allows us to compute the center as a linear combination of the base vectors $(\mathbf{e}'_x, \mathbf{e}'_z)$. Third is the actual radius r_l. However, we first need to see if one of the circles is contained within the other. If so, then one boundary is pushed by the encompassing circle:

$$-r_m > d_{mn} - r_n \Longrightarrow \Delta\phi = d_{mn}, r_l = r_n$$

$$r_m > d_{mn} + r_n \Longrightarrow \Delta\phi = 0, r_l = r_m$$

The first condition is for n encompassing m, the second is the other way around. If neither is true, the computation is slightly more difficult:

$$\Delta\phi = \frac{d_{mn} - r_m + r_n}{2}$$

$$r_l = \frac{r_m + r_n + d_{mn}}{2}$$

Finally, we compute the center of l[5]:

$$\mathbf{c}_l = \cos(\Delta\phi)\mathbf{e}'_x + \sin(\Delta\phi)\mathbf{e}'_z$$

2.9 Texture mapping

The system of overlays requires creating a texture every time the planet is to be rendered. Suppose we have a texture with resolution $w \times h$ that should have a color b_{ij} from some color space $\mathcal{C}$ assigned to each of its points:

$$i \in \mathcal{I} = \{0, 1, 2, ..., w-1\}$$

$$j \in \mathcal{J} = \{0, 1, 2, ..., h-1\}$$

$$b : \mathcal{I} \times \mathcal{J} \to \mathcal{C}$$

We want b_{ij} to reflect the data on the sphere. Let b' be a map from the sphere surface, representing an overlay:

$$b' : \mathcal{S} \to \mathcal{C}$$

We now have to find some map between the pixel indices and the sphere surface. This is very similar to the classical problem in cartography. Our choice will be an equirectangular projection because of its simplicity. We first compute the azimuthal and polar angles ϕ_i, θ_j from i, j:

$$\phi_i = 2\pi\frac{\left(i + \frac{1}{2}\right)}{w}$$

$$\theta_j = \pi(1 - \frac{j + \frac{1}{2}}{h})$$

This means the texture y coordinate increases *upwards* (to north) in the respective cylinder rectangle (Figure 9). From the angles it is simple to find the point on the sphere:

$$\mathbf{x}_{ij} = (\sin\theta_j \cos\phi_i, \cos\theta_j, \sin\theta_j \sin\phi_i)$$

Finally, we can compute b_{ij}:

$$b_{ij} = b'(\sin\theta_j \cos\phi_i, \cos\theta_j, \sin\theta_j \sin\phi_i)$$

[5]Note that the value of $\Delta\phi$ can be negative. This is the case of clockwise-oriented arc from $\mathbf{c}_m$ in the local basis.

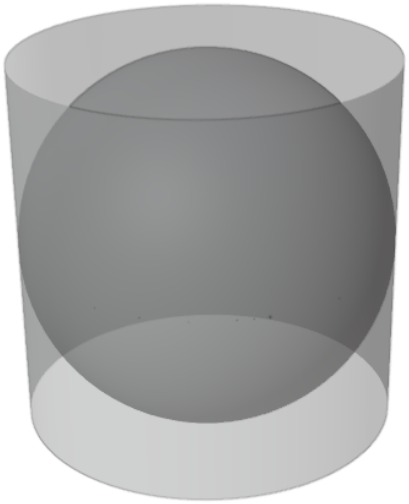


Figure 9: Cylinder rectangle around a sphere

This texture is subsequently used for rendering the surface, however, there is an inherent problem with uv mapping. Because of my insecurity about the math involved, the process will not be explained in this text and instead, the reader is encouraged to read a more educated article [16].

There is perhaps a better way to deal with texture mapping, which was discussed in a Unity forum thread [17]. Cubemaps are quite possibly a more logical choice, as the texture information density needlesly increases towards extreme values of the y coordinate.

2.10 Unit dimensions

Virtually all expressions throughout this section assumed a unit sphere for simplicity, but for rendering and parameter context some scaling is needed. For example, radius R of the planet might be given in kilometers [1]. Quantities such as this one describe planets in a physical metric space with a basic length unit of 1 m (and its metric prefixes). We consider the basic length unit in Unity scenes a 'Unity meter' u and the corresponding space as Unity space. Given standard radius of a planet (in order of 1000 km), rendering in 1:1 scale from metric to Unity space is impractical. We set a scale convention 1 u = 1000 km. Because of the geological time scale, our basic unit of time will be My (million years). Some useful conversions follow in table 1.

The planet radius expressed in Unity meters is used as a scaling parameter to transform quantities from Unity space to the unit sphere representation (denoted by $R_{\mathcal{U}}$). The simulation runs in the unit sphere representation and planet is rendered by scaling the vertices on the unit sphere by $R_{\mathcal{U}}$.

Example Planet with a radius $R = 6370$ km has a maximum plate speed[6] v_0' of 100 mm·y^{-1} and an average plate area $\mathcal{A}_0'$ of 25.5×10^6 km^2 . We want to transform these values to a unit sphere representation. The scaling parameter is $R_{\mathcal{U}} = 6.37$ u. Maximum plate speed is transformed as:

$$v_0 = \frac{v_0'}{R_{\mathcal{U}}} = \frac{100 \text{ mm} \cdot \text{y}^{-1}}{6370 \text{ km}} = \frac{0.1 \text{ u} \cdot \text{My}^{-1}}{6.37 \text{ u}} \approx 0.0157 \text{ My}^{-1}$$

[6] Because Driftworld Tectonics uses the unit sphere values, we will reference quantities declared in the original article [1] as primed.

Metric unit	Unity conversion
1 km	0.001 u
1 km^{-1}	1000 u^{-1}
1 $\text{mm}\cdot\text{y}^{-1}$	10^{-3} $\text{u}\cdot\text{My}^{-1}$

Table 1: Unit conversion table

This allows us to identify any surface speed with an angular speed. The average area has two length dimensions, so to transform to a unit sphere, the expression is:

$$\mathcal{A}_0 = \frac{\mathcal{A}'_0}{R_{\mathcal{U}}^2} = \frac{25.5 \times 10^6 \text{ km}^2}{40576900 \text{ km}^2} = \frac{25.5 \text{ u}^2}{40.5769 \text{ u}^2} \approx 0.628$$

Note that the value is fraction-scaled to 4π, which is the unit sphere surface area.

2.11 Vector noise on mesh

Given a Delauney triangulation of a sphere, Driftworld uses a specific type of noise to randomize simple linear borders, such as between two tectonic plates. Because only whole mesh triangles are assigned to plates, a single noise value is assigned to each triangle. Because the goal of the randomizer is to shift border directions, the noise values are vectors. A single value $\mathbf{m}$ is obtained by taking a random vector $\mathbf{s} \in \mathcal{S}$ and projecting it onto the tangent plane of the triangle centroid $\mathbf{c}$[7]:

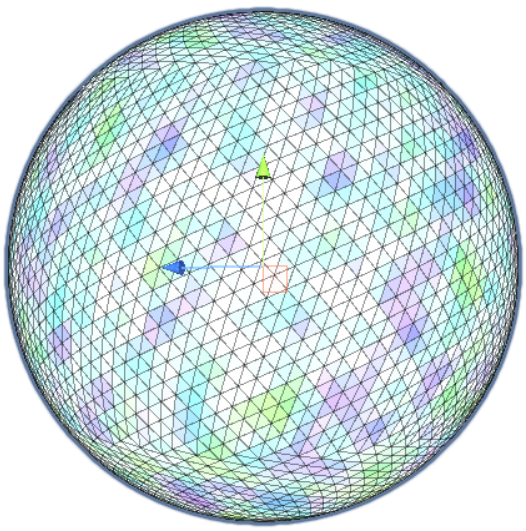
$$\mathbf{m} = \mathbf{s} - (\mathbf{s} \cdot \mathbf{c})\mathbf{c}$$

This way, the obtained random vectors should be uniformly distributed. Note that the vector lengths are from the interval $[0, 1]$ and completely random. To avoid some of the border jitter, the noise should be low-frequency. Every triangle has three neighbours along its edges. This means that for every triangle we can average its noise vector with the neighbour noise vectors to filter higher frequency noise. We only need to ensure that the result is again within the tangent plane by projecting either the contributing vectors or the result itself. This summation can be repeated several time to adjust the filtering.

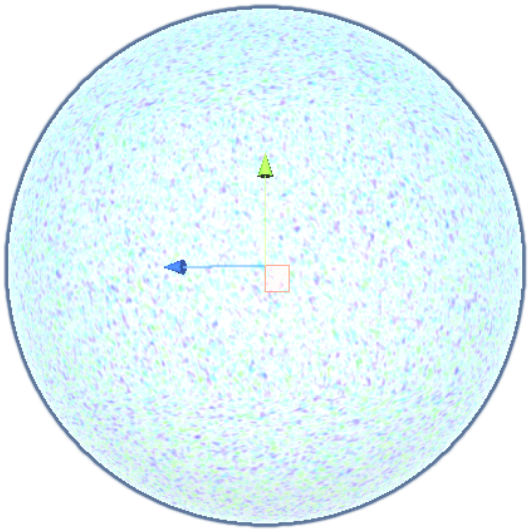
The resulting low-frequency vector noise is driven by the number of averaging iterations. This is basically a 'smearing' of the triangle noise up to a certain radius. The total number of triangles in a mesh influences the resulting noise pattern (detail) on the sphere. In Figure 10 the same number of iterations were used for meshes with 10,000 (10a) and 500,000 (10b) triangles. It is clearly seen that the the granulation is finer on the sphere with more detailed mesh. Also, there is a noise pattern change along the lines where the mesh pattern changes. This is partially because each triangle is colored whole with a single color and the triangles are not perfectly regular.

It might be a good idea to revisit the concept of a vector noise on the sphere some time in the future. The concept is experimental and it is unclear if it has some unforeseen consequences for the plate border interactions.

[7] This part is similar to the grid vector assignment when calculating 2D Perlin noise [18], although with additional projection.



(a) low poly with mesh - whole triangles are coloured



(b) high poly without mesh - borders of mesh patterns visible

Figure 10: Vector noise representation - hue represents relative direction, saturation the length of a vector, value is set to constant 1

3 Tectonic model

We now introduce the detailed description of the tectonic model used to create the planet crust. We discuss differences and similarities with respect to the original article and adapt the mechanisms to our unit sphere representation. Parameters that drive the model are summarized at the end of the section in Table 2. It should be stated that all parameters will be optimized for the number of vertex samples N equal to 500,000.

The simplest description of the algorithm is that it creates some random crust partitioning into plates. These plates have randomized drifting parameters for moving. Then a certain number of tectonic steps is performed with a time step length δt of 2 My and the result is a basic crust of the simulated planet. Between automated tectonic steps, user can force a few specific interactions (plate rifting, terrain smoothing) and change various global parameters to influence the simulation. The planet radius is set to 6370 km = 6.37 u.

3.1 Workflow

Basic crust is generated from a Delaunay triangulation of a unit sphere. The fresh crust is just a set of triangulated vertices, while each vertex is assigned some default crust data h. Each tectonic step consists of several substeps in a sequence. This sequence is firmly set because of implementation context and is depicted in Figure 11. Short descriptions for the substeps follow.

Continental collisions If the plate drift would result in an overlap of two different continental (above sea level) areas of crust belonging to different plates, a continental collision is triggered, resulting in a massive uplift of the lighter plate. The two plates in question are then merged into one. This is different from the orignal algorithm, which only attaches connected continental areas, called *terranes*.

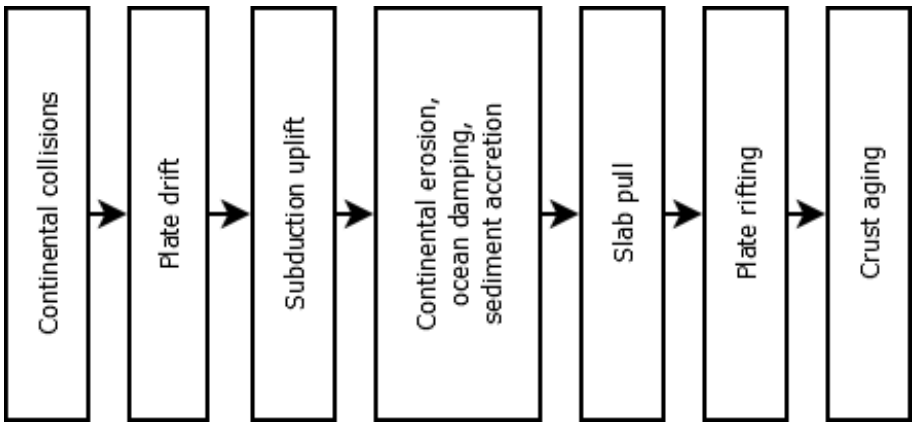


Figure 11: Tectonic step structure

Plate drift Rotation transform of each plate along the sphere surface is updated by its respective angular speed.

Subduction uplift Overlapping areas of different plates cause uplift in the lighter plates. This process is known as *subduction*. Note that overlapping continental areas have already been dealt with so this case should never occur during this step.

Continental erosion, oceanic damping, sediment accretion This step updates elevation values to simulate erosion of crust above sea level, lowering of the ocean bottom for underwater crust and sediment accretion for the oceanic crust below average ocean depth. This is questionable, as the original algorithm probably detects trench area for sediment filling.

Slab pull Subducting parts of plates tend to pull the plate towards them, altering the surface rotation. All vertices in subduction zones contribute to the rotation vectors of their plates.

Plate rifting Every tectonic step the largest plate has a chance to rift apart. Along some random linear border within the plate the vertices are assigned to two new plates with diverging velocities.

Crust aging Age of every crust vertex is updated by the length of the tectonic step.

3.2 Crust & plates

The crust is defined as a set of surface vertices U, obtained by Fibonacci sampling. Its size is the number of samples N. Each vertex $\mathbf{u}_i \in U$ is assigned crust point data h_i. The tectonic plate system is an equivalence system $\mathcal{P}$ of U (so that every crust point belongs exactly to one plate). These equivalence classes should be connected through the mesh, but it is not a strict requirement (although it subtracts from the realism). Plates are the individual equivalence classes $\mathcal{P}_i \in \mathcal{P}$. If all vertices of a mesh triangle belong to a plate $\mathcal{P}_i$, the triangle is also said to belong to $\mathcal{P}_i$. A triangle only has three neighbours, each sharing one edge. If a triangle belonging to $\mathcal{P}_i$ has a neighbouring triangle which does not, it is called a *border triangle*.

Each plate is also assigned: a centroid $\mathbf{c}_i$, rotation axis $\mathbf{w}_i$, plate angular speed ω_i and a transform q_i. The centroid is a vector calculated as the normalized sum of all vector representations of vertices belonging to the plate. If it cannot be normalized, a random vector is assigned. The rotation axis is a unit vector along the axis around which the plate drifts. The plate angular speed is self-explanatory. The transform is a quaternion representation of the relative rotation of the plate with respect to the original position. This is because the simulation does not actually move the plate vertices, only adjusts

the plate transform to correctly calculate interactions. Moving the vertices would introduce serious problems with rendering.

3.3 Crust data

Crust data values included so far are: elevation, crust thickness, orogeny type and crust age. All crust points are strictly represented by unit vectors. The elevation values are information stored separately. Oceanic crust are all crust points with negative elevation, continental crust points have a non-negative elevation. Crust thickness is a placeholder information for potential future updates. Orogeny type is represented by three categories: *None*, *Andean* and *Himalayan*. The Andean type is a crust point elevated above the ocean level by subduction, the Himalayan type is a crust point that was influenced by continental collision. The None type is reserved for crust points not yet elevated by continental collision nor elevated above the ocean level by subduction. It does not exist in the original article, as the orogeny type is reserved for continental crust. The crust age is simple the time passed from the creation of the crust point. The original model also uses fold direction, which is not yet implemented, as I do not properly understand its purpose and mechanics.

The default crust point data for new points depends on whether the new point is continental or not. The only new continental points are created during the first partitioning (see Subsection 3.4). Initial elevation is $z_{0t} = -0.004$ u for oceanic crust and $z_{0c} = 0.001$ u for continental crust. Crust thickness is always calculated from a basic crust thickness value $e_0 = 0.01$ u as $e = e_0 + z$, where z is the crust elevation. The initial orogeny type is None for all new oceanic crust points and Andean for the initial continental crust. Initial crust age is universally equal to 0.

3.4 Plate initialization

Partitioning of the crust into the initial set of plates is governed by two parameters: the number of initial plates $N_{\mathcal{P}}$ and the probability of an initial plate being continental p_C. The initial number of plates is 30 and the probability of an initial plate to be continental is 0 for testing purposes. Before partitioning, vector noise is assigned to each triangle on the mesh with a noise averaging iterations parameter n_{sm} of 4.

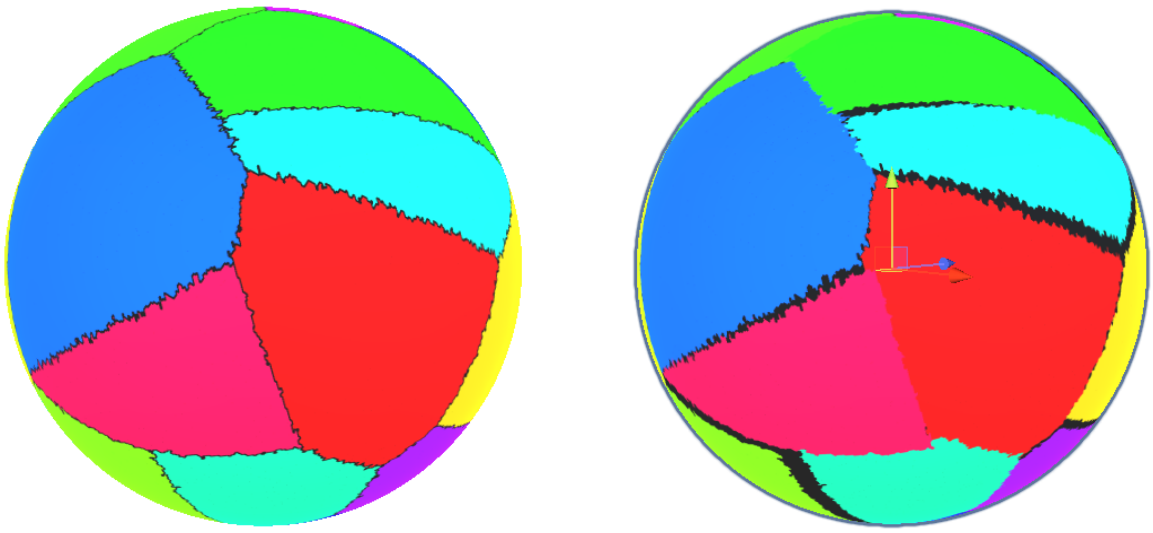
At first, $N_{\mathcal{P}}$ number of random points $\mathbf{c}$ (future *centroids* of the plates) is distributed on the surface. Then all initial crust vertices are assigned to these points by their shortest distance on a unit sphere:

$$d(\mathbf{x}, \mathbf{c}) = \arccos(\mathbf{x} \cdot \mathbf{c})$$

All points that have the shortest distance to a certain centroid point belong to a single plate. This plate inherints the points and the centroid. When all points are distributed to their plates, each plate is then assigned a random rotation axis $\mathbf{w}$ and random non-negative angular speed ω. The maximum plate angular speed is $v_0 = 0.0157$ My^{-1}. The symbol is unchanged to correspond with the original quantity. Finally, each plate is assigned a quaternion identity transform q. All crust points are assigned default data according to their plate. The probability of continental crust is evaluated on the plate level, so the initial plates all have the same elevation.

This kind of initialization basically creates a Voronoi diagram with perfectly straight lines (up to the triangle resolution). To simulate more realistic plate boundaries, vector noise is used. First we look

for the triangles which have vertices from exactly two different plates. For each of these triangles we try to roll for probability equal to its noise vector magnitude. If the probability succeeds, we compute three dot products between the noise vector and each vector from the triangle barycenter to the vertex. The vertex with the maximum dot product is assigned to the plate of the vertex with the minimum dot product. This shifts the borders of the plates and is repeated $n_{\text{vb}} = 4$ times. This parameter is the number of Voronoi border shift iterations. An example of the resulting partitioning can be seen in Figure 12a.



(a) initial plates

(b) plates drifting

Figure 12: Crust partitioning

3.5 Plate overlaps

Tectonic interactions require the concept of plate density. For example, denser oceanic plates are subducted under continental plates. Because our model does not have a clear designation of a plate as oceanic or continental (any plate can have oceanic or continental crust), we evaluate plates by a weighted sum of their vertices. Each plate is assigned a score equal to:

$$\text{score} = 100 \times \text{number of continental crust points} - \text{number of oceanic crust points}$$

The plates are then ranked by the highest score. The rank decides which plate 'goes under' when two plates overlap (the one with a lower score). This actually creates an irreflexive, antisymmetric and transitive relation on the set of plates.

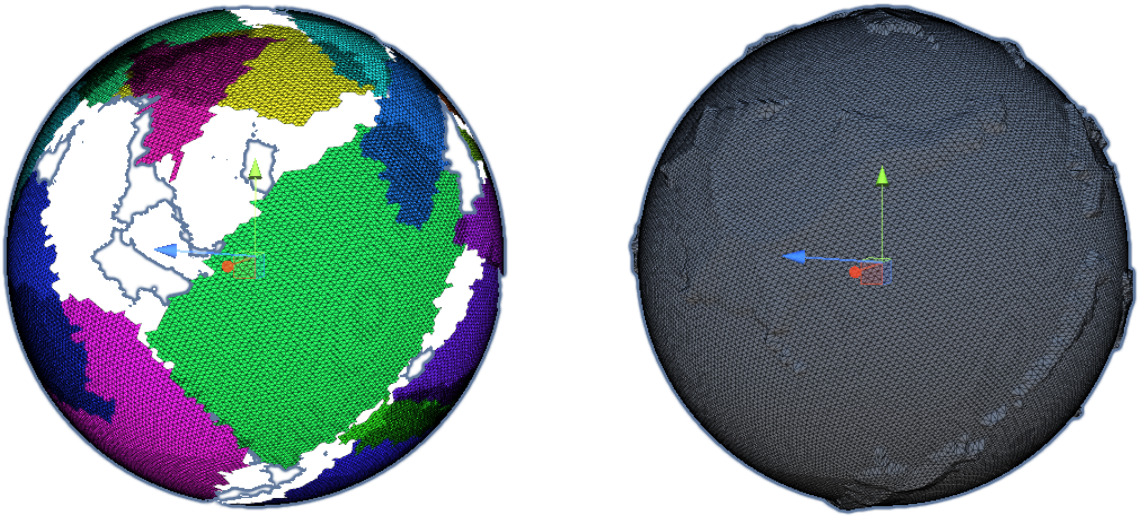
This ranking system is a gross simplification. As many simplifications, though, it makes certain decisions and calculations much easier. The plate ranks have to be recalculated every time an interaction requires them, as the scores change both with crust elevation and changes in crust point assignments to plates. An example might be when we need to know which of the overlapping plates defines a crust point elevation on the surface.

3.6 Plate drift

The main reason for tectonic interactions is the tectonic drift. Plates move constantly, causing collisions, subduction etc. To model the drift of a plate, during every tectonic step the plate transform is adjusted by multiplication of the transform by a quaternion representing a rotation around the axis $\mathbf{w}$ by the angle of $\Delta\phi = \omega\delta t$. This keeps the information about current crust points locations. The result of a one step drift from the initial position can be seen in Figure 12b. We can see here that the plates move as individual rigid bodies.

3.7 Oceanic crust generation & crust resampling

Moving rigid plates necessarily create gaps on the surface. In reality, this 'empty' space is filled with new crust drifting from oceanic ridges between the plates (see Figure 13). We use the original model, only slightly simplified. We can interpolate crust data at any point on the surface which is not in any triangle belonging to a plate. We compute two distances to two nearest plates (nearest vertices belonging to two different plates) d_1 and d_2 (1 being the absolute shortest) and assume that the point is approximately on the direct line between the nearest points. We also assume that the ridge is directly in the middle of the line. This is not true in reality, but makes it simple to use the original algorithm easily. We compute the ridge and plate elevation contributions and combine them as per Cortial et al.



(a) diverging plates

(b) generated ridges

Figure 13: Oceanic crust generation

The ridge function profile uses three parameters: the highest oceanic ridge elevation z_r, the abyssal plains elevation z_a and the oceanic ridge elevation falloff σ_r. The values of these parameters are:

$$z_r = -1 \text{ km} = -0.001 \text{ u}$$

$$z_a = -6 \text{ km} = -0.006 \text{ u}$$

$$\sigma_r = 0.05 \text{ u}$$

The ridge function profile is a function of a variable d_Γ which is the distance to the oceanic ridge. The function profile is chosen as:

$$z_\Gamma = (z_r - z_a)2^{\frac{d_\Gamma}{\sigma_r}} + z_a$$

The original function profile is not specifically described and may be more complex. Because of our assumptions we can calculate the ridge function profile variable that is the distance to the ridge as:

$$d_\Gamma = \frac{d_2 - d_1}{2}$$

The orogeny type is universally filled as None for points in the surface voids. The crust age is computed as the scaling parameter $\alpha = \frac{d_\Gamma}{d_\Gamma + d_1}$ multiplied by the total time for which the plates have been diverging (since last they were in close contact). This makes the new oceanic crust gradually older the further it is from the ridge. Finally, any new oceanic crust point is assigned to the nearest plate.

For simulation running for many steps it is vital that we periodically resample the surface to fill the gaps made by diverging plates and to resolve overlapping plates. As per Cortial et al., it is recommended to resample the surface every 10th-60th step. Our model forces resampling on several occasions, namely continental collision. The resampling simply means to interpolate surface data onto the original mesh from the current surface data defined on a mesh broken by drifting plates and interactions. For each initial mesh vertex, we test if it is found on a plate with the highest rank possible. If so, we perform barycentric interpolation from a triangle within which the vertex currently resides. If no plate is found, we create a new oceanic crust point. When all crust point data is assigned, the original mesh with the interpolated crust point data becomes the new crust model. Because the algorithm remembers to which plates the crust points belong, the new crust points are reassigned to their respective plates and the plate transforms are reset to identity.

3.8 Continental collisions

The quaternion transform mechanics have both advantages and disadvantages. The main advantage is that it allows for faster computations and relatively simple data management. The main disadvantage is that it does not allow for simple crust point reassignment to another plate if the plates' respective transforms are not identities. For this reason, whenever a point is assigned to another plate, a resample must be performed in advance. This is the reason why continental collisions are evaluated as the very first in a tectonic step. We use a predictive detection, that is we first simulate a one-step drift and then detect whether two continental triangles (triangles with all-continental points) from two different plates intersect. This is sufficient condition for triggering a continental collision. This varies significantly from Cortial et al. because we do not require the plates to overlap over a specific distance (the 300 km and more originally). This may create unwanted tectonic artifacts and should be addressed in the future. Note that the simulated drift does not actually move the plates, it only adjusts transforms for the collision tests.

First we test all continental triangles for continental collisions with all other plates and store all collision triggers. If any such trigger occurs, we first flag all vertices in the triggering triangles and then we search for all connected groups of continental vertices that contain the flagged vertices and belong to a single plate. These groups are called *terranes*. It should be noted here that it is theoretically possible for one triangle to be colliding with two different plates. When constructing terranes, we only accept the first matching plate in order. This order is arbitrary and only depends on the order in which the plates are stored in the memory. Multi-collision for terranes is therefore ignored, as it is a very rare occurence. The way terranes are constructed allow for a vertex to be flagged as

colliding with a plate with which it was not flagged as colliding beforehand, simply because it belongs to a single terrane from a vertex which does. Although this is not based on reality, it does make the algorithm unambiguous and hopefully does not negatively affect the simulation. If a continental collision occured, we now perform a one-step drift for all plates.

We now have a set of terranes belonging to certain plates. Each terrane has an assigned plate with which it collides. All terranes have a radius of influence inside which they cause collision uplift. This radius depends on several parameters. The first is the global maximum collision distance r_c, the second is the relative plate speed v_{ij} at the terrane centroid, the third is the maximum plate speed v_0, the fourth is the area of the terrane $\mathcal{A}_i$ and the fifth is the average initial plate area $\mathcal{A}_0$. The global maximum collision distance is a model parameter, which needs to be scaled to the unit sphere:

$$r_c = \frac{4200 \text{ km}}{6.37 \text{ u}} = \frac{4.2 \text{ u}}{6.37 \text{ u}} \approx 0,659$$

The vector velocity of a plate $\mathcal{P}_i$ at a certain point $\mathbf{q} \in \mathcal{S}$ is equal to:

$$\mathbf{s}_i(\mathbf{q}) = \omega_i \mathbf{w}_i \times \mathbf{q}$$

Relative speed of two plates $\mathcal{P}_i, \mathcal{P}_j$ at a point $\mathbf{q}$ is:

$$v_{ij}(\mathbf{q}) = ||\mathbf{s}_i(\mathbf{q}) - \mathbf{s}_j(\mathbf{q})|| = ||\omega_i \mathbf{w}_i \times \mathbf{q} - \omega_j \mathbf{w}_j \times \mathbf{q}|| = ||(\omega_i \mathbf{w}_i - \omega_j \mathbf{w}_j) \times \mathbf{q}||$$

The radius of influence is computed as follows:

$$r = r_c \sqrt{\frac{v_{ij}}{v_0}} \frac{\mathcal{A}_i}{\mathcal{A}_0} \approx r_c \sqrt{\frac{v_{ij}}{v_0}} N_i \frac{N_\mathcal{P}}{N} = r_c \sqrt{\frac{v_{ij}}{v_0}} \frac{N_i}{N_0}$$

The ratio of terrane area to the average initial area is approximately the same as the ratio of the number of terrane vertices to the average number of vertices in an initial plate, provided the number of samples is sufficiently high.

This range applies to area beyond the border of the terrane. A vertex on a plate in which the terrane collides is affected by the collision uplift if its distance d from the nearest vertex of the terrane is less or equal to r.

If a crust point is affected by a continental collision, this collision causes an immediate uplift based on a discrete collision coefficient $\Delta_c = 1.3 \times 10^{-5} \text{ km}^{-1} = 1.3 \times 10^{-2} \text{ u}^{-1}$. The uplift is is computed as follows:

$$\Delta z_i = \Delta_c \mathcal{A}_i \left(1 - \left(\frac{d}{r}\right)^2\right)^2 \approx 4\pi R^2 \frac{N_i}{N_\mathcal{P}} \Delta_c \left(1 - \left(\frac{d}{r}\right)^2\right)^2, d \in [0, r]$$

This value is added to the elevation value of the crust point. If a crust point was affected by a continental collision, its orogeny type becomes Himalayan. Finally, after all collisions have been resolved, we resample the whole crust, the vertices of all colliding terranes are reassigned to their new plates and the whole crust is resampled again.

3.9 Subduction uplift

The process of subduction takes place when two plates come into contact and continental collision is not triggered. Continental collisions should already be dealt with when subduction is evaluated. If

the plates did not drift during a continental collision event, a one-step plate drift is performed before testing subduction events.

Subduction test begins with determining all subduction front points. Given a border triangle $\mathcal{T}_i$ on a plate $\mathcal{P}_i$, we search all other plates for triangles intersecting with $\mathcal{T}_i$. For each other plate, we try to find at least one and take the first. If such a triangle $\mathcal{T}_j$ is found on a plate $\mathcal{P}_j$, we store a new subduction front point at the the centroid location $\mathbf{c}_i$ of $\mathcal{T}_i$ assigned to $\mathcal{P}_i$ with the subduction uplift source $\mathcal{P}_j$ and an elevation value equal to the mean elevation value of the vertices in $\mathcal{T}_j$. We try to find a subduction front point for all border triangles on all plates.

Subduction front point information is then used to calculate the potential subduction uplift for all crust vertices. For each crust vertex $\mathbf{p}_i$ and each of its possible subduction uplift source plates, we find the smallest distance d_j to a subduction front point assigned to its plate with a given subduction source plate $\mathcal{P}_j$. There is a maximum distance allowed for subduction, called *maximum subduction distance* r_s. Our model value is:

$$r_s = \frac{1800 \text{ km}}{6.37 \text{ u}} = \frac{1.8 \text{ u}}{6.37 \text{ u}} \approx 0.283$$

If the subduction uplift source plate $\mathcal{P}_j$ has a lower overlap score $d_j < r_s$, we calculate the subduction uplift contribution, otherwise the contribution is set to 0.[8] The uplift contribution $\tilde{u}_j$ from $\mathcal{P}_j$ has three parts: distance transfer, speed transfer and height transfer:

$$\tilde{u}_j = u_0 u_{\text{dt}} u_{\text{st}} u_{\text{ht}}$$

$u_0 = 0.6 \text{ mm} \cdot \text{y}^{-1} = 0.0006 \text{ u} \cdot \text{My}^{-1}$ is called *base subduction uplift*. The distance transfer is a parametrized cubic function:[9]

$$u_{\text{dt}}(d_j) = \frac{\frac{d_j^3}{3} - \frac{(r_c+r_s)d_j^2}{2} + r_c r_s d_j + \frac{r_s^3}{6} - \frac{r_s^2 r_c}{2}}{\frac{r_s^3 - r_c^3}{6} + \frac{r_c^2 r_s - r_s^2 r_c}{2}}$$

$r_c = 0.1$ is the *subduction control distance* and it is the distance in which the transfer has maximum. The other transfer functions are computed as follows:

$$u_{\text{st}}(v_{ij}) = \frac{v_{ij}(\mathbf{p}_i)}{v_0}, u_{\text{ht}}(z_j) = \left(\frac{z_j - z_t}{z_c - z_t}\right)^2$$

v_{ij} is the relative speed plate at $\mathbf{p}_i$ and z_j is the elevation value of the subduction front point.

When all uplift contributions $\tilde{u}_j$ have been calculated at $\mathbf{p}_i$ for all possible subduction uplift source plates $\mathcal{P}_j$, the vertex elevation z_i is increased by:

$$\Delta z_i = \delta t \sum_j \tilde{u}_j$$

The subduction event possibly differs from Cortial et al. in that multiple contributions are possible at once and also the height transfer is carried from the subduction front as opposed to the original expression (denoted $z_i(\mathbf{p})$ in the article). This algorithm, however, presented no observed artifacts so far. It should be noted, however, that terranes on a mostly oceanic plate get subducted under an oceanic plate with a higher overlap score – this, apparently, is not very realistic.

[8] We also set the contribution to 0 if there is no subduction front point for a given subduction uplift source plate.

[9] Cortial et al. only mention piece-wise cubic function, this is an example implementation.

3.10 Continental erosion

The continental crust erosion adjusts the elevation values of all continental crust points:

$$\Delta z = -\frac{z}{z_c}\epsilon_c \delta t$$

$\epsilon_c = 3 \times 10^{-2}$ mm $\cdot$ y^{-1} $= 3 \times 10^{-5}$ u $\cdot$ My^{-1} is the *continental erosion parameter*.

3.11 Oceanic damping

The oceanic damping adjusts the elevation values of all oceanic crust points:

$$\Delta z = -\left(1 - \frac{z}{z_t}\right)\epsilon_o \delta t$$

$\epsilon_o = 4 \times 10^{-2}$ mm $\cdot$ y^{-1} $= 4 \times 10^{-5}$ u $\cdot$ My^{-1} is the *oceanic damping parameter*.

3.12 Sediment accretion

The sediment accretion adjusts the elevation values of all oceanic crust points below the average ocean depth $\bar{z}_o = -0.004$ u:

$$\Delta z = \epsilon_t \delta t$$

$\epsilon_t = 3 \times 10^{-1}$ mm $\cdot$ y^{-1} $= 3 \times 10^{-4}$ u $\cdot$ My^{-1} is the *sediment accretion parameter*.

3.13 Slab pull

All vertices of a plate $\mathcal{P}_i$ in a subduction front of another plate $\mathcal{P}_j$ alter its rotation axis $\mathbf{w}_i$. In our model, this means we have to flag all vertices that are found inside any triangle belonging to another plate with a higher overlap score. Again, we assume no continental collision is taking place since all such events have been resolved earlier in the tectonic step. We group all vertices by the plates they belong to and then we compute slab pull contributions for each plate. Given a plate $\mathcal{P}_i$ with a rotation axis $\mathbf{w}_i$ and a centroid $\mathbf{c}_i$, the slab pull adjustment for n flagged points (each at a location $\mathbf{q}_k$) is computed as follows:

$$\delta \mathbf{w}_i = \epsilon' \sum_{k=0}^{n-1} \frac{\mathbf{c}_i \times \mathbf{q}_k}{||\mathbf{c}_i \times \mathbf{q}_k||}\delta t$$

ϵ' is a perturbation coefficient scaling the influence of the slab pull. Because the summation vector varies wildly with the total number of mesh vertices, we need to find a parameter somewhat independent of N. We chose $\epsilon' = \epsilon\frac{N_\mathcal{P}}{N}$ and call the new parameter $\epsilon = 1$ *slab pull perturbation parameter*. After the adjustments are computed for all plates, they are added as vectors to their respective axes and the results are normalized to 1.

3.14 Plate rifting

Continental collisions inevitably create unproportionally large tectonic plates. This leads to decreased tectonic activity and created continents sink through erosion into the ocean. To counter that, there is a chance for plates to rift. Every tectonic step Driftworld checks probability to rift the largest plate.[10] The probability for a plate $\mathcal{P}_j$ to rift is $p_i = \lambda_i e^{-\lambda_i}$. Driftworld computes the parameter λ_i differently as:

$$\lambda_i = \lambda_0 \frac{\mathcal{A}_i}{\mathcal{A}_0} \delta t = \lambda_0 \frac{N_i N_{\mathcal{P}}}{N} \delta t$$

$\lambda_0 = 0.1$ My^{-1} is the *average rifting frequency.*

If the rifting event is triggered, the plate is divided into two plates by two random centroids within the plate according to the closest vertex distance. Finally, vector noise is applied to the shared border.

3.15 Crust aging

All crust points age is simply incremented by δt.

Symbol	Description	Original value	Model value
N	Number of mesh vertices	-	500,000
δt	Tectonic time step	2 My	2 My
R	Planet radius	6,378 km	6.37 u
z_{0t}	Initial oceanic elevation	-	-0.004 u
z_{0c}	Initial continental elevation	-	0.001 u
e_0	Basic crust thickness	-	0.01 u
$N_{\mathcal{P}}$	Initial number of plates	40	30
p_C	Initial continental plate probability	0.3	0
v_0	Maximum plate speed	100 mm·y^{-1}	0.0157 My^{-1}
n_{sm}	Noise averaging iterations	-	4
n_{vb}	Voronoi border shift iterations	-	4
z_r	Highest oceanic ridge elevation	-1 km	-0.001 u
z_a	Abyssal plains elevation	-6 km	-0.006 u
σ_r	Oceanic ridge elevation falloff	-	0.05
r_c	Maximum global collision distance	4200 km	0.659
Δ_c	Discrete collision coefficient	1.3×10^{-5} km^{-1}	1.3×10^{-2} u^{-1}
r_s	Maximum subduction distance	1800 km	0.283
u_0	Base subduction uplift	0.6 mm $\cdot$ y^{-1}	0.0006 u $\cdot$ My^{-1}
r_c	Subduction control distance	-	0.1
ϵ_c	Continental erosion parameter	3×10^{-2} mm $\cdot$ y^{-1}	3×10^{-5} u $\cdot$ My^{-1}
ϵ_o	Oceanic damping parameter	4×10^{-2} mm $\cdot$ y^{-1}	4×10^{-5} u $\cdot$ My^{-1}
$\bar{z}_o$	Average ocean depth	-	-0.004 u
ϵ_t	Sediment accretion parameter	3×10^{-1} mm $\cdot$ y^{-1}	3×10^{-4} u $\cdot$ My^{-1}
ϵ	Slab pull perturbation parameter	-	1
λ_0	Average rifting frequency	-	0.1 My^{-1}

Table 2: Model parameters summary

[10] The original article does not mention the mechanism specifically, but it probably checks all plates.

4 Implementation & data model

4.1 Rendering

4.2 GPU Computing

4.3 Bounding volume hiearchy

When dealing with interactions of objects consisting of many triangles, such as parts of a mesh or meshes, testing every triangle against each other is very inefficient, leading to algorithmic complexity of $\mathcal{O}(n^2)$. For this, Driftworld implements *bounding volume hiearchy* (BVH) over the triangles of the sphere mesh [15]. Bounding volume (BV) is a simple object that contains a primitive or a group of primitives. Its purpose is to simplify collision tests – for this, bounding volumes are structured in a hiearchy. The most common BVH is a tree – its leaves are bounding volumes of individual primitives. Parent nodes of such a tree are bounding volumes created by convenient merging of their children node BVs. The root of the tree is then some whole object. For instance, it could be a bounding volume of a mesh, while the leaves would be the singular triangles or their bounding volumes. There are multiple types of various bounding volumes and multiple types of constructed BV trees.

For our purpose, we construct a BVH for the triangles inside some region of the sphere triangulation. Our bounding volume is a spherical circle, the tree is a binary tree of merged circles. This choice makes sense because the triangles only move on the surface of the sphere and therefore we do not need spatial bounding volumes. Mathematically

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

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

[9] Todhunter, Isaac. (1886). *Spherical Trigonometry: For the Use of Colleges and Schools* (5th ed.). London: Macmillan and Co.

[10] *Ray Tracing: Rendering a triangle*. Scratchapixel 2.0. Retrieved June 6, 2022, from https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates

[11] Pokojski, Wojciech & Pokojska, Paulina. (2018). Voronoi diagrams – inventor, method, applications. *Polish Cartographical Review*. 50. 141-150. 10.2478/pcr-2018-0009.

[12] Delaunay, Boris. (1934). Sur la sphere vide. A la memoire de Georges Voronoi. *Bulletin de l'Academie des Sciences de l'URSS. Classe des sciences mathematiques et naturelles*, Issue 6, pp. 793–800.

[13] Guibas, L. J., Knuth, D. E., & Sharir, M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7(1-6), 381–413. doi:10.1007/bf01758770

[14] Ma, Yingdong & Chen, Qian. (2010). Fast Delaunay Triangulation and Voronoi Diagram Generation on the Sphere. *2010 Second World Congress on Software Engineering*, pp. 187-190, doi: 10.1109/WCSE.2010.136

[15] Sulaiman, Hamzah & Bade, Abdullah. (2012). *Bounding Volume Hierarchies for Collision Detection*. 10.5772/35555.

[16] Golus, Ben. (2021). *Distinctive Derivative Differences*. Retrieved June 15, 2022, from https://bgolus.medium.com/distinctive-derivative-differences-cce38d36797b#85c9

[17] (2020). *Texture repeats itself within single border triangles*. Unity forum thread. Retrieved June 15, 2022, from https://forum.unity.com/threads/texture-repeats-itself-within-single-border-triangles.1029907/

[18] Wikipedia contributors. (2022, May 31). Perlin noise. In *Wikipedia, The Free Encyclopedia*. Retrieved 20:26, July 28, 2022, from https://en.wikipedia.org/w/index.php?title=Perlin_noise&oldid=1090721154

List of Figures

List of Tables

A Project parameters

Table 3 contains all relevant parameters and constants used in the project. It is sectioned into the tectonic model and the implementation parameters to navigate more easily. If available, the original values from Cortial et al. are included for reference.

Symbol	**Description**	**Original value**	**Adapted value**
bla	bla	bla	bla
bla	bla	bla	bla

Table 3: Project parameters overview