University of North Carolina Asheville
Journal of Undergraduate Research
Asheville, North Carolina
May 2022

Island Generator

Owen Feldmann
Department of Computer Science
The University of North Carolina Asheville
One University Heights
Asheville, North Carolina 28804 USA

Faculty Advisors: Dr. Marietta Camera, Dr. Adam Whitley

Abstract

Handcrafting game assets can be very time consuming, especially when variations are needed. Procedural generation allows many similar assets to be created in significantly less time than manually creating these assets. This project creates software for the procedural generation of a 3D island for use in games or animations. The island is split into multiple different biomes, each with their own generation rules and decorative props such as trees and bushes. The approach used for biome generation utilizes Voronoi Noise to define the position and size of biomes. A method of stitching together the harsh borders between these biomes is covered. The project also experiments with an algorithm to parametrically build feature points on the terrain, namely a volcano. A player character is included so that the user may observe the island from a person's perspective. The user has access to a settings menu that affects how the island is generated. These features allow users to quickly create interesting custom island models.

1. Introduction

Island Generator is a software tool that generates islands from a given seed and various other settings. Once an island is generated, it can easily be exported as a .fbx file and imported to 3d editing software or game engines. This allows users to quickly create a suitable environment for their games, animations, or renderings, while using the software tools they desire. This is especially useful for game development since multiple unique levels may be created by just changing the seed. Generating an island can take about a second, an enormous improvement to handcrafting an island model from scratch. Even if the generated island is used as a base and modified manually, significant amounts of time can be saved. Software generating a landscape also handles randomness more fairly than a human handcrafting a landscape. The human will generally have some bias when they try to select random options.

Another motivation is the opportunity to experiment with and implement terrain generation ideas the author has read about in the past. Island Generator is meant to be an expansion and improvement upon the scrope of a previous unnamed project which generated a small 2D map grid for a sandbox type game. The previous project would be similar to Island Generator using settings for single biome generation with no adjustments.

The default settings create an island that is split into various biomes, place props such as trees and rocks in these biomes, and generates a volcano. Each of these major settings may be toggled on or off and sub settings are included to control the generation process in finer detail. For example, settings for controlling the height and size of the volcano are available. The project demo[2] has a UI menu that allows the user to change the settings and experiment easily. More advanced changes, such as changing which props generate in which biomes, require changes to the project[3] in Unity Engine 2020.3.8f1. Further reading on the use of Unity Engine can be found at[11].

2. Literature Review

2.1. Mesh Generation

2.1.1. heightmaps

Heightmaps are one of the simplest methods for generating and displaying terrain. Heightmaps exist on a plane, often limited to a square or grid of points. Each of these points is then assigned a value, often between 0 and 1, which is the height at that point. The value of the points is determined by a function of the location on the heightmap. A major limitation is that each location can only have a single height assigned. This means that more complex terrain options such as caves and overhangs are impossible to represent[9].

Several options exist to define the height function[8]. The simplest is a constant function, which would result in flat terrain. The other extreme is to use a pure random function so that every point on the terrain has no relation to nearby points. This results in unnatural and spiky terrain. Natural terrain tends to be somewhat smooth. One solution to this is interpolated terrain, where only some points are randomly selected, and the rest are interpolated from them. Cubic interpolation creates smooth curves, rather than straight lines, between these random points. A similar technique is fractal landscapes, where each corner is seeded with a random value and the diamond square pattern is used to set the value of the remaining points based on the values of the determined points[1, 8].

Island Generator uses gradient, or noise[7] based heightmaps. Perlin noise was developed by Ken Perlin to generate procedural textures for computer graphics applications. Noise creates smooth curves between nearby points and can easily be translated or stretched without problems, making it a popular choice for generating heightmaps. Noise functions can map 1,2,3, or higher dimensional spaces to a single dimensional value.

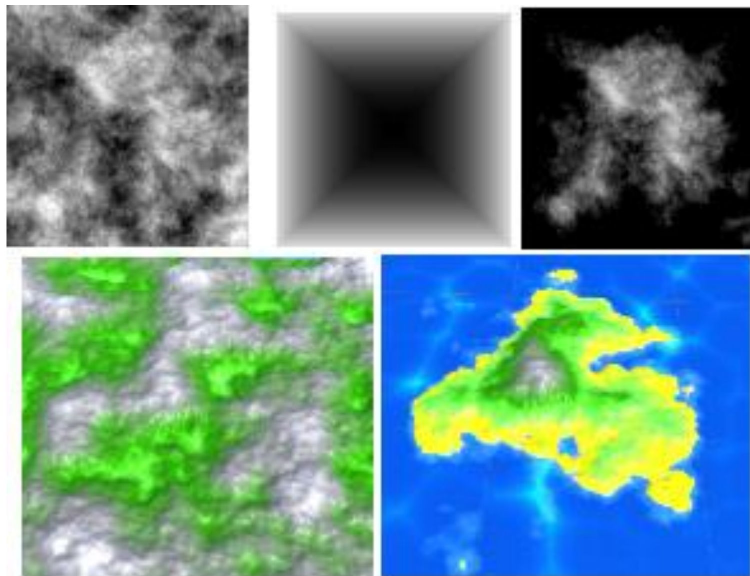

Figure 1. 2D noise examples

Figure 1 The top three images[10] from left to right show 2D noise, a square gradient, and the square gradient subtracted from the noise. White corresponds to the value 1, the highest elevation, and black corresponds to 0. The bottom left image is an angled aerial view of terrain generated by Island Generator. The bottom right image is the result of subtracting a circular gradient from this terrain.

2.1.2. marching cubes

An alternative to heightmap terrain is the marching cubes algorithm[9]. Marching cubes takes a scalar point field, usually three dimensional, and 'wraps' a mesh around the points that are determined to be inside the mesh. These points can

be either binary ‘in’ or ‘not in’, or determined by comparing a tolerance value to decimal values. An advantage over heightmaps is that marching cubes can represent overhangs and caves. This terrain can also be easily terraformed by a user during runtime by editing the scalar field and ‘rewrapping’ the mesh. This ease of modification makes marching cubes an excellent choice for sandbox games where voxels are not desirable. The drawback is the added complexity of the algorithm compared to the simplicity of heightmaps.

2.2. Biome Generation

2.2.1. voronoi noise

Cellular[4] or Worley noise functions require a list of points. When called with a position, the function returns the distance from that position to the closest point. This results in a smooth change in values. Using a single point in the center of a region can be used to create a circular gradient. Voronoi noise differs from Cellular noise by returning the closest point to the supplied position. This creates discrete regions which can be used to generate biomes. The downside to this method is that biome boundaries are harsh, with no transition between them. The biomes assigned to the points can also be arbitrary without relation to each other, such as a desert next to a snowy biome.

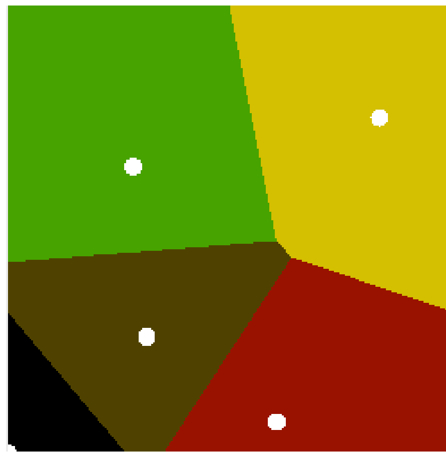

Figure 2. Voronoi noise example

Figure 2 A square colored with Voronoi noise using five points. Image generated by modifying a code snippet in[4]. If Cellular noise were used instead, a black and white image could be generated so that it is darker near each of the points and gradually becomes lighter farther away from these points.

2.2.2. moisture and temperature heightmaps

Another approach to biome generation is the layering of multiple heightmaps[10]. While a heightmap can represent the elevation at a point, moisture and/or temperature heightmaps can represent the climate. For example, a point that is at high elevation, cold, and has a high moisture value could be labeled a snowy mountain. If that point had low moisture, it could instead be a barren rocky mountain. This technique is easy to use since biome transitions are smooth and relate to nearby biomes.

2.3. Generated Terrain Shape

A major consideration when generating terrain is the overall shape of it. The simplest option is a plane, just a square, infinite or otherwise, of terrain. If a circular gradient is subtracted from the plane, an island shape is the result[10]. Heightmaps are effective tools for these first two shapes which are generated from a 2D base shape. Marching cubes is effective when using a 3D base shape, such as a cube. Tunnels could be carved through the cube to create caves, or a spherical gradient could be subtracted to create a planet[9].

2.4. Generated Terrain Scale

The scale of the generated terrain must be considered. An infinite plane of terrain can not be loaded all at once. It must be split into chunks and dynamically generated and loaded. It may be desirable to chunk finite terrain too. A planet is finitely large, but may be too large to load all at once. There is also no point in loading the far side of the planet, which is not visible to the viewer. A limitation in Unity[11] is that 3D meshes only render about 65,000 vertices. This barely fits a 256 by 256 square grid of vertices. While an island is finite in size, it can not exceed this size without dealing with the complexity added by chunking.

2.5. Generated Feature Points

Once terrain has been generated, there are options to modify it. Parametrically generating feature points[5] such as mountains or craters can add points of interest to terrain so that it looks less consistent and thus more interesting and potentially realistic. There are several algorithms to generate feature points, but they tend to be implemented with iterative techniques that layer several small adjustments to approximate the desired outcome.

2.6. Low-poly Terrain

Low-poly[6] is an art style which uses a minimal amount of polygon in 3D models. The advantage of using this art style is that fewer triangles are needed to be generated. This decreases the time required to generate the terrain and allows the software that uses the generated terrain to run more smoothly. This also reduces the time needed for a 3D modeler to handcraft models that don't visually clash with the terrain. A low-poly character can be created much more quickly than a realistic character.

3. Project Description

3.1. Specifications

Since there are several conflicting possibilities for terrain generation, some must be selected to match the project. Island Generator uses heightmaps generated with Perlin noise and a circular gradient to create the basic island shape. This shape is what is visible when biome generation is disabled. When biome generation is enabled, all land above sea level is flattened down to sea level before biomes are generated. This results in the heightmap determining what areas are land, and generating the underwater terrain only. Heightmaps were chosen over the marching cubes algorithm for their simplicity. The limitation of being unable to generate caves and overhangs is deemed trivial for this project.

Biome generation is accomplished using Voronoi noise instead of moisture and temperature maps. This option was chosen with the goal of generating discrete and unique biomes, each with different generation rules. Another reason this option was chosen is to experiment with blending the borders between discrete biomes. The blending algorithm will be discussed in Section 3.2.1. Inside each of the biome regions, terrain is generated using layers of Perlin noise heightmaps masked with the corresponding region. The values of Perlin noise, which range from 0 to 1, are mapped to new values by using an animation curve. This allows terrain other than basic rolling hills, such as plateaus and spiky mountains, to be generated. Vertex colors are similarly mapped from the noise values, but use a gradient instead.

Island Generator clearly uses an island shaped generation technique. This option was chosen for the finite nature of the shape, limiting the need for dynamic generation techniques. This is further limited by using only one chunk. While this limits the scale of the island, a reasonably large island can fit inside a single chunk. The island shape also prevents questions about how to deal with the land at the edge of the world.

To experiment with modifying the island after generation, a volcano is generated as a feature point. The generation algorithm is further discussed in Section 3.2.2. Two final options for modifying the island mesh are smooth terracing and vertex jiggling. Smooth terracing drops each vertex down the the closest terrace step. The walls of the terraces are angled instead of vertical. Making vertical terrace walls requires the mesh to be modified to contain twice as many vertices, which is problematic since the maximum island size already contains as many vertices as Unity allows. Vertice jiggling just moves each vertex to a nearby position. This interrupts the smoothness of the terrain to make it appear more natural. While some smoothness appears natural, too much smoothness starts to appear unnatural.

The final step is to place props such as rocks and trees on the island. This algorithm is examined in Section 3.2.3.

Most of the generation process outlined could be completed in a single step where each vertice only needs to be visited once. Instead, the project completes each step sequentially. This allows for the process to be animated in a manner which helps the user understand how the island is generated, instead of just producing an island with no explanation. A series of screenshots showing this animation process is shown in Figure 3.

Once the island finishes generating, a simple pill shaped player character is added. When the user clicks on the island, the player will attempt to walk to the clicked point. This is implemented with a Unity AI NavMesh. The primary goal of the player is to give the user a sense of the scale of the island, which can be hard to understand from a static top-down view.

A user interface(UI) is also included. This is primarily useful for the web hosted demo and users who are less involved in customizing their generation settings. The UI gives incomplete access to the settings of the project, focusing on general settings such as toggles and particularly useful options such as changing the size of the island, setting a seed, etc. Some of the more complex settings such as changing biome generation or prop placement settings require direct changes in the Unity editor. The UI also contains a menu that explains the settings provided, an overlay which names each step of the generation process when animating, and a menu that explains how to export the island as a fbx file.

Exporting the island as a fbx file packages both the island mesh, and all the props placed on the island. Fbx is a standard 3D model file format and is an effective choice when transferring models between applications. An island created in Unity can be exported to Unreal Engine, or to Blender where it can be edited manually and exported as any of several different file formats. Due to limitations of the software package used to create the fbx file, the process must be completed during runtime in the Unity editor. This also breaks the vertex color shader material that colors the island. This shader can easily be recreated in the destination software.

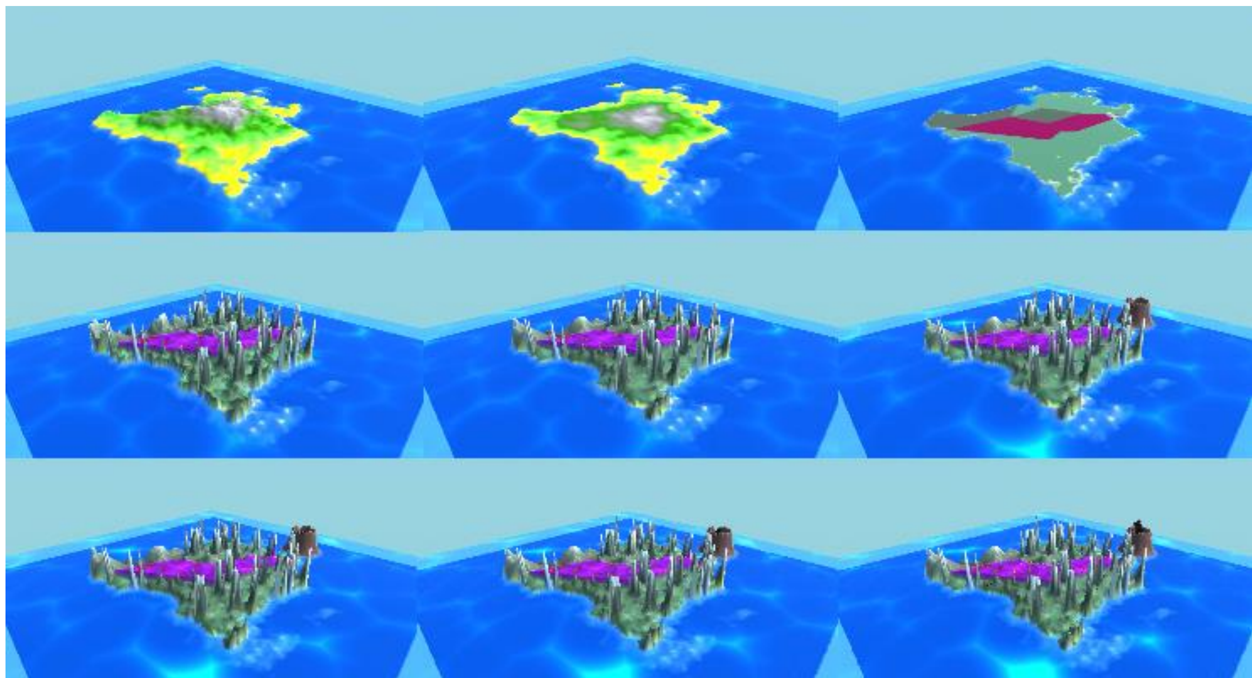

Figure 3. Island generation step by step process

Figure 3 From left to right, top to bottom the images show each step in the island generation process that is shown when animating. First is the creation of the island shape. Then it is flattened and split into biome regions. Note that the biomes on this island are plateau, mountains, and spiky mountains. These regions then have terrain generated followed by biome blending. Next a volcano is generated. The last three frames are difficult to see the differences in the small aerial view images in this Figure. First is the smooth terracing followed by vertex jiggling. Finally props are added.

3.2. Algorithm Design

3.2.1. biome blending

Just using Voronoi noise to generate biomes creates harsh boundaries. An example would be a sudden vertical cliff when a mountain is generated on the edge of a mountain biome placed next to a plains biome. Cliffs like these tend to be jarring and unnatural. Therefore, these boundaries need to be smoothed. There are two aspects to consider: vertex smoothing, and color smoothing. Vertex smoothing affects the shape of the terrain and cleans up the example cliff. Color smoothing blends the colors between biomes. An example would be the sudden change between desert sand yellow and plains grass green.

The solution used by this project is a selective averaging function that is applied to all vertices, not just boundary vertices. This makes the interior of biomes look slightly better. The averaging function must be selective so that it does not destroy unique terrain such as plateaus or spiky mountains. The function accesses each vertex and checks the surrounding vertices within a square radius. There are two parts of this function, for vertex height and color respectively. The sea is not included in averages since it would drop terrain below sea level and color land blue. Vertices in plateau biomes are also not targeted by the averaging function in order to preserve the unique terrain shape and color. Vertices outside of, but near plateau biomes exclude the vertices inside the plateau biome to maintain their cliff faces. This also prevents the red and purple colors from spilling out. Most of the terrain in spiky mountain biomes can be averaged safely since it is a similar color and height to other terrain. The spiky mountains themselves must be excluded from the averaging to preserve their shape and prevent their lighter color from spreading.

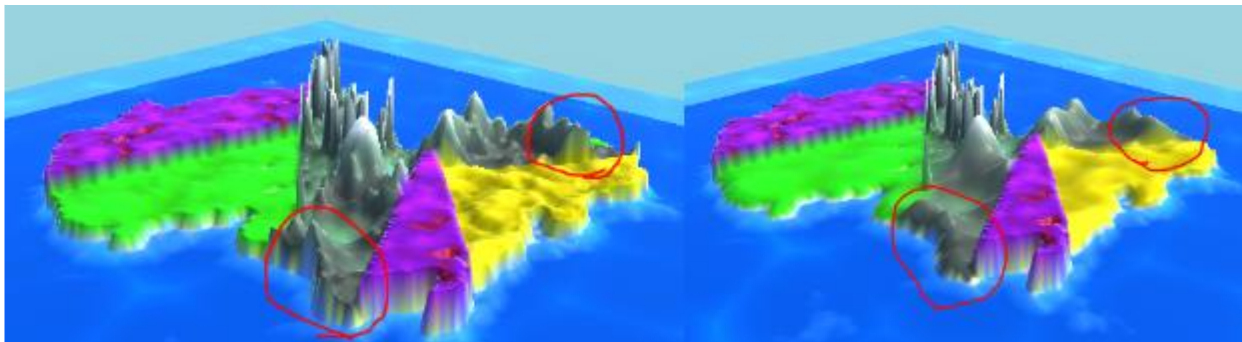

Figure 4. Biome blended cliffs

Figure 4 The image on the left shows the original island and the image on the right shows the same island after the biome blending algorithm has passed over it. Note that the sharp cliff faces circled appear more smooth and natural. Also note that the coastline flows more naturally into the sea and that the spiky mountains and plateau are not destroyed.

3.2.2. volcano generation

The volcano generation algorithm takes inspiration from the iterative nature of the algorithms used to generate feature points in[5]. The start of the algorithm creates a foundation for the volcano by averaging all vertices within the affected region with sea level a couple times. These vertices are all colored brown. Creating this foundation minimizes some visual problems created when the algorithm 'lifts' up the terrain. For example, if the volcano generates on the edge of a plateau, half the volcano is the plateau's height taller than the other half, with a cliff dividing it. Another example is the volcano's rim generating under a spiky mountain, lifting it up so that its peak is twice as tall as it should be.

The shape of the volcano rim is approximated using a circle. Each iteration places a spread circle centered at a random point on the rim circle. Then an iteration circle is placed at a random point on the spread circle. All vertices contained within the iteration circle have their elevation increased slightly. Every iteration, the radius of both the spread circle and iteration circles are slightly decreased. This causes early iterations to build a wider base which then tapers inwards during later iterations. Vertices within the rim circle and below a specified height are colored an orange

color to appear as lava. A visual effect system is placed inside the volcano to create a smoke effect at the end of the algorithm.

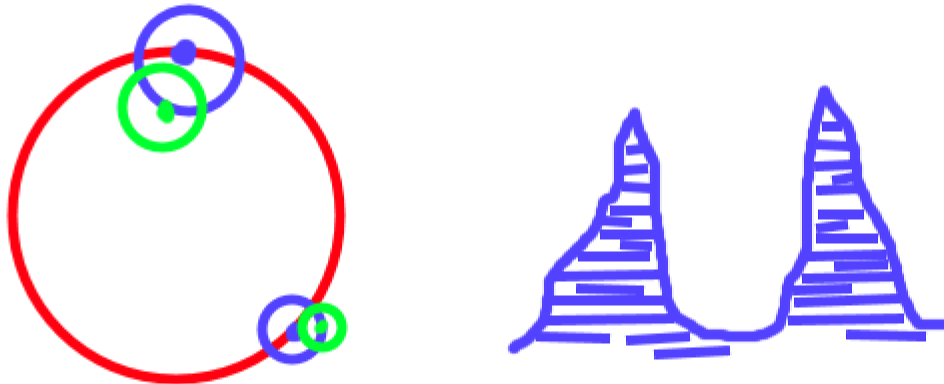

Figure 5. Volcano generation algorithm

Figure 5 The red circle is the rim of the volcano, the blue circle is the spread circle, and the green circle is the iteration circle. Just the first and last iterations are drawn, with the first one being the larger circle. The right shows a cross-sectional view through the volcano that shows how it is elevated by repeatedly stacking circles. The decrease in circle size results in tapering near the rim.

3.2.3. prop placement

There are several approaches to prop placement. Two options are to use random placement or be clustering by using a tolerance value with a noise heightmap. Island Generator uses the clustering option for the appearance. The algorithm used has a set amount of attempts to place props. Each attempt selects a random location on the island and checks if it is valid. Invalid positions are in the sea, on the volcano, or too close to other props. If the location is valid, all props that can be placed in that biome have their tolerance settings compared to their noise functions at that location. A random prop that passes this check is then placed. If an attempt fails at any point, no prop is placed at that location and the attempt ends. Fewer props will be placed than attempts made. The clustering on 'noise hills' groups similar props together as shown in Figure 6. This looks more natural than purely random placement.

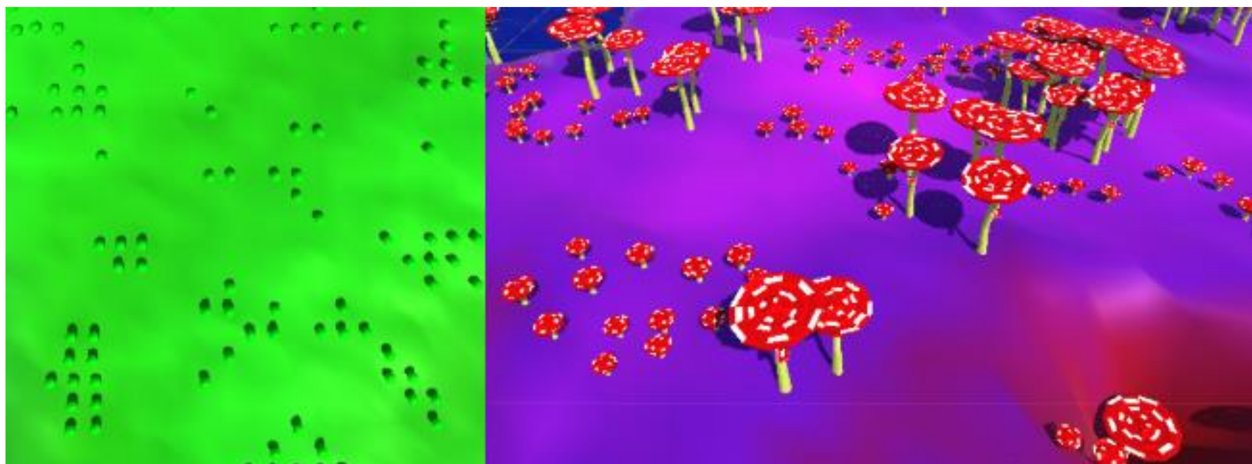

Figure 6. Prop clustering

Figure 6 Left shows the clustering of bushes on 'noise hills' in a plains biome. Bushes are placed at integer value positions to clearly show clustering. The right shows clustering of mushrooms on a plateau biome. Note that there

are two types of mushrooms and that they form clusters independently of each other. Props are ordinarily placed at floating point value positions and are scaled and rotated randomly for variety as shown on the right.

4. Results

Overall, the development of Island Generator proceeded smoothly. The basic goal of generating the island mesh was completed early, leaving time for stretch goals and feature creep involving visual polishing and the UI. The intended use of generating an island works nicely with the default settings as well as most combinations of settings. Some particularly nice island views are shown in Figure 7 and several islands generated with varying settings are shown in Figure 8. Experimenting with some of the settings also creates some other non-intentend possibilities as a result of not checking the value of some settings.

For example, the falloff rate setting multiplies the circle gradient which enforces the island shape. A larger positive falloff rate results in a smaller island. When this value is zero, no gradient is subtracted, creating a basic plain of terrain. This is more obvious when biome generation is disabled. Negative falloff rate values instead adds a circular gradient, which results in a valley/crater shape. An unfortunate result of this setting choice is that Unity calculates the concavity to be inside the collider created for the island mesh. This results in some props being placed on a surface that doesn't exist. This can be ignored by disabling prop generation. While such valleys are an unintended result, they are interesting enough to consider not checking the values for some settings input fields.

Figure 7. Picturesque views

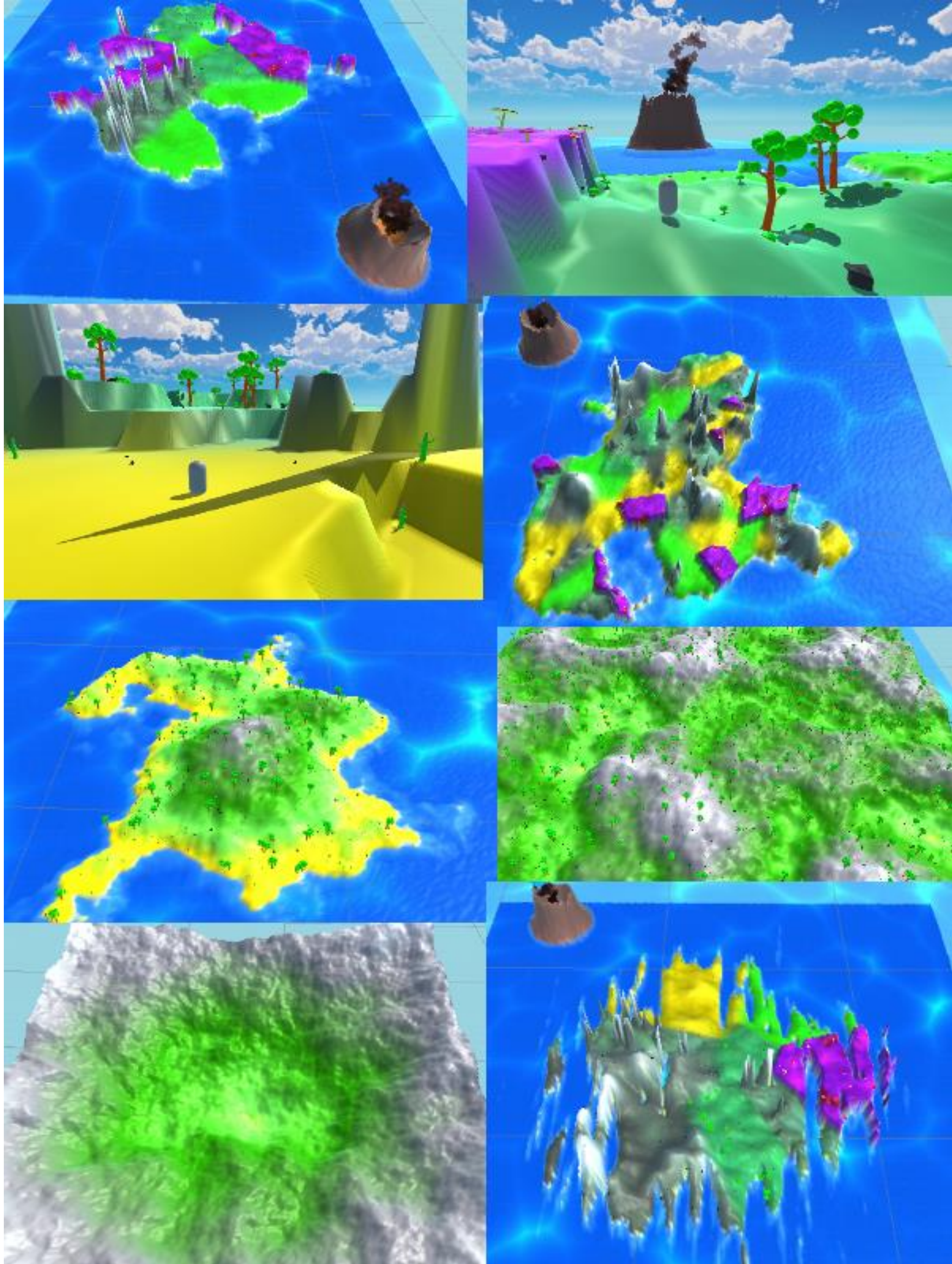

Figure 8. Various islands

Figure 8. The top two images are an aerial and a ground view of the same island. Below is a ground view of a terraced island and a view of an island with 100 biomes placed. Next is an island with biome generation disabled, which is next to an 'island' with the same settings, but the falloff rate set to 0. Below is another 'island' with a negative falloff

rate that results in a valley/crater shape. The last island scales the noise that shapes the island differently in the X and Z directions.

5. Future Work

Island Generator is currently done as it is, but there are many aspects that could be modified or improved. One feature creep idea is the ability to import or export the settings defined in the UI. This feature idea could be convenient, but it is limited by what settings are represented in the UI and does not include more advanced changes such as changes to biome or prop placement settings. While the settings menu could be expanded to include more options, any serious users would want to modify settings in the Unity editor, which has a better UI. An UI related feature that was cut was a menu to export the island as a fbx. Due to various difficulties experienced when attempting to implement this, this feature was limited to using a package in the Unity editor instead.

One possible change to the island generation is to use chunking to generate a larger island. While the current maximum island size is adequate, the option to generate larger islands could be useful. Another change to generation would be to rework the system to use the marching cubes algorithm. While options such as overhangs and caves are not crucial to the project, they could make the island terrain more interesting. This change would be more about the learning process of how to implement this algorithm. More changes such as options for other mesh modifiers could be added. This would be in the same vein as the smooth terracing and vertex jiggling options.

Prop variety is also another area of improvement. Currently there is only one tree model used. This is visually monotonous, but somewhat mitigated by rotating or slightly scaling every prop differently. Prop modeling was never a major aspect of the original scope of the project. Prop related goals focused on placement, not modeling.

One final proposed change is the biome blending algorithm. The Voronoi noise aspect of the biome generation algorithm could be changed to return the two closest biome points to a given location. If the distance to each of these biome points is within some range, then the terrain at that location could be interpolated between the two terrains generated by each of the biome points. This process would ideally replace the current biome blending algorithm. One area that would require extra attention in this proposed algorithm is the boundaries between land and sea since sharp cliffs are not eliminated by the proposed algorithm as it is.

6. References

1. Brown, Adam. “The Maths of Fractal Landscapes and Procedural Landscape Generation.” Fractal Landscapes, 2002, https://www.fractal-landscapes.co.uk/maths.html.
2. Feldmann, Owen. “Island Generator Demo.” 2022, https://erawlam.itch.io/island-generator.
3. Feldmann, Owen “Island Generator Repository.” 2022, https://github.com/OwenFeldmann/Island-Generator-Capstone-Project
4. Gonzalez Vivo, Patricio, and Jen Lowe. The Book of Shaders, 2015, https://thebookofshaders.com/12/.
5. Kamal, K. Raiyan, and Yusuf Sarwar Uddin. “Parametrically Controlled Terrain Generation.” Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia - GRAPHITE '07, 1 Dec. 2007, pp. 17–23, https://doi.org/10.1145/1321261.1321264.
6. O'Caoimh, Josh. “Advantages and Disadvantages of Low Poly in Game Design.” Josh O'Caoimh, 2021, https://joshocaoimh.com/advantages-and-disadvantages-of-low-poly-in-game-design.
7. Perlin, Ken. “An image synthesizer.” ACM SIGGRAPH Computer Graphics Volume 19, Issue 3, Jul. 1985, pp. 287-296, https://doi.org/10.1145/325334.325247.
8. Shaker, Noor, et al. “Chapter 4 Fractals, Noise and Agents with Applications to Landscapes.” Procedural Content Generation in Games: A Textbook and an Overview of Current Research, 2016, http://pcgbook.com/wp-content/uploads/chapter04.pdf.
9. Sin, Zackary P. T., and Peter H. F. Ng. “Planetary Marching Cubes: A Marching Cubes Algorithm for Spherical Space.” Proceedings of the 2018 the 2nd International Conference on Video and Image Processing, 29 Dec. 2018, pp. 89–94. ACM Digital Library, https://doi.org/10.1145/3301506.3301522.
10. Travall. “Procedural 2D Island Generation — Noise Functions.” Medium, Medium, 26 Oct. 2018, https://medium.com/@travall/procedural-2d-island-generation-noise-functions-13976bddeaf9.
11. “Unity User Manual 2020.3 (LTS).” Unity Documentation, Unity Technologies, 2022, https://docs.unity3d.com/Manual/index.html.