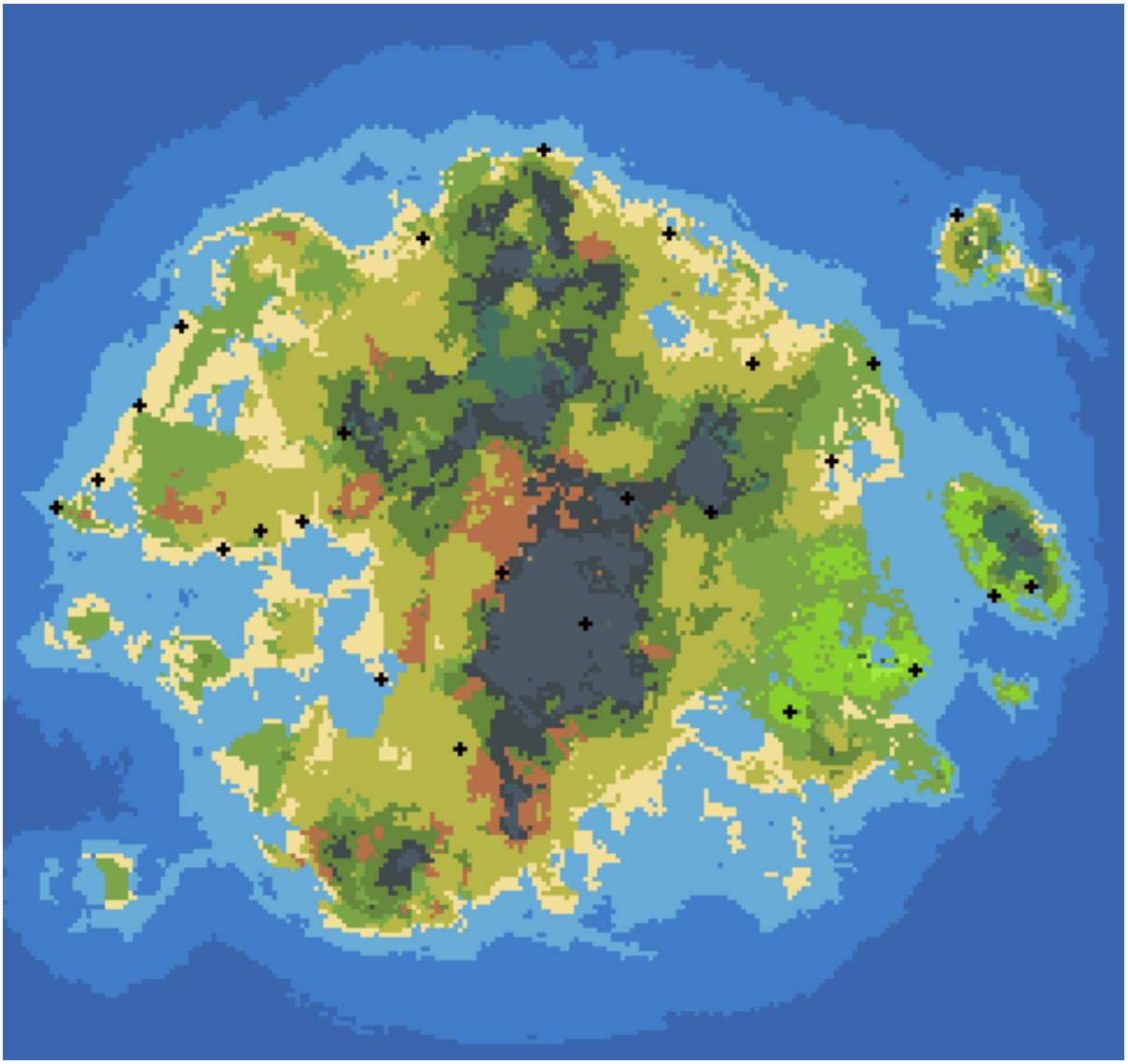
Fundamentals of Terrain Generation



1. **Why use terrain generation?**

   Terrain generation is useful for games/applications where you want natural looking terrain (caves, hills, rivers, biomes, etc.) that has a smooth/continuous feel, but is also random.

   - Side-view games with linear terrain (Line Rider, Hermit)
   - Games with caves (Stardew Valley, Terraria)
   - Games with different types of terrain in "biomes"
   - Map creating/editing applications
   - 2.5D or 3D graphics games (Minecraft)

2. **Case Study: Hermit**

Hermit is a 15-112 Term Project from a few years back that had an infinite procedurally-generated world. The world is a side-view, and the terrain is generated forever in a single direction. For this type of terrain, various algorithms can generate height values for each x-coordinate. Procedural-generation algorithms are ones that generate the terrain as the player moves along it, rather than generating the entire terrain the moment the game loads.

3. **Case Study: Terraria**

Terraria is one of many games that has cave-like terrain structures from a side view or birds-eye view. In these kinds of games, a grid can be generated indicating whether each cell is stone or air. The goal of such algorithms is to make the grid randomly shaped, yet still feel continuous and curved like a natural cave structure.

4. **Case Study: WorldAnvil**

WorldAnvil is a world building application that includes a map creating/editing mode. Its possible to create a term project that also centers on a map building/editing application. Parts of it could be centered on the user drawing the terrain themselves, but parts of it could involve tools to randomly generate parts of the map. This could involve algorithms to generate biomes, features of the terrain, and the shape of the continent.

5. **Case Study: Minecraft**

Minecraft is a 3D graphics game that has many kinds of terrain generation. There are different biomes which are laid out in random configurations (jungles, deserts, mesas, oceans, plains, mushroom forests, mountains, etc.), underground caves/ravines, and above ground hills/rivers/plains. Each of those can be handled with a different algorithm (i.e. one for biomes, one for underground caves, one for above ground terrain-height).

6. **Algorithms:** (or at least, a subset of them)

1D Hill Generation:
A simple algorithm is to generate a bunch of random sinusodal functions in order to control the elevation of a hill. Another algorithm that is great for generating this kind of terrain is called **midpoint displacement**. It works by starting with random values at two points, calculating their average at their midpoint, adding some random noise, then repeating recursively for the midpoints between the starting points and the midpoint.

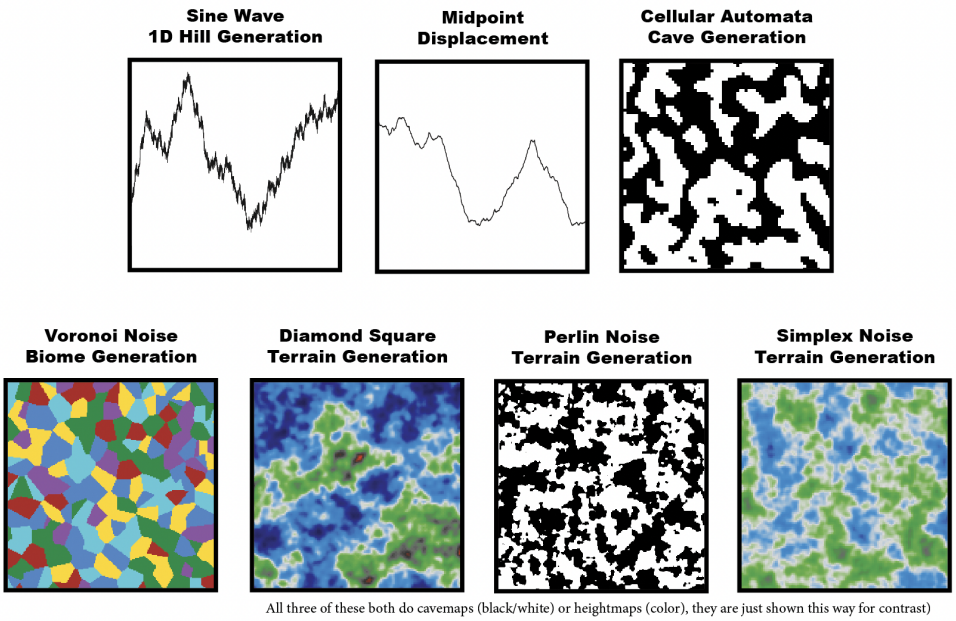
Cellular Automata Cave Generation:
The cellular automata approach is based on the idea of starting with a random grid of 0s (walls) and 1s (passages) then repeatedly applying a simple set of rules to each cell to obtain a new grid, repeating the process several times. These rules are often something like "If the cell is currently a passage, and if at least 6 of its neighbors are walls, then the cell will become a wall".

Voronoi Noise Biome Generation:
Voronoi Noise (aka Worley Noise) is based on randomly placing a bunch of "seeds" in the grid, each of which has a randomly chosen biome. Then, the biome of each cell in the grid is based on the closest seed.

Diamond Square Algorithm:
The Diamond Square algorithm generates terrain (i.e. hills) with a fractal approach. It starts with some random values that are spaced appart, then calculates the noise values in between by taking averages and adding some noise. The process is repeated many times until every cell has a noise value.

Perlin Noise Terrain Generation: (VERY HARD)
Perlin Noise is a complex algorithm for generating random terrain that is still smooth / continuous. Its quite difficult to get working, but the results are stunning. The best part is that this algorithm works well with **procedural** generation, meaning it create load part of the world, then when you move towards the edge, you can generate more terrain and it will seamlessly connect.

Simplex Noise Terrain Generation: (VERY VERY HARD)
Simplex noise is a variant of Perlin noise which uses triangles instead of a square grid for the gradient vector locations. Its best saved for additional features, since its extremely difficult to implement.



All three of these both do cavemaps (black/white) or heightmaps (color), they are just shown this way for contrast)

7. **Example: Cellular Automata Cave Generation**

There are many possible rules for the cellular automata algorithm. Below is one example:

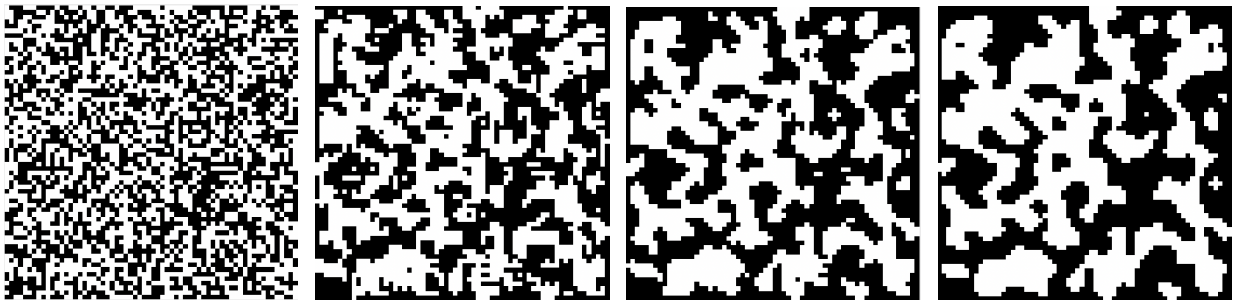
Suppose $w_i(r, c) = 0$ if cell (r, c) is a wall and 1 if it is a passage at time i.
Suppose $n_i(r, c) =$ the sum of $w_i(r', c')$ for each neighbor (r', c') of (r, c).
Suppose that for each (r, c), $w_0(r, c)$ is randomly chosen to be 0 or 1.

One iteration of the automata can be described by the formula below:

$$w_{i+1}(r,c) = \begin{cases} 1 & w_i(r,c) = 1 \ \& \ n_i(r,c) \geq 4 \\ 1 & w_i(r,c) = 0 \ \& \ n_i(r,c) \geq 5 \\ 0 & otherwise \end{cases}$$

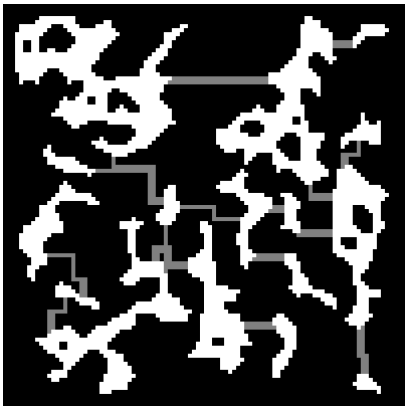
In English: if the cell is a passage and at least 4 of its neighbors are passages, then the cell is a passage in the next iteration. If a cell is a wall and at least 5 of its neighbors are passages, then it is a passage in the next iteration. Otherwise, it is a wall in the next iteration.

Repeat this algorithm a few times (say, 3 to 7) and you get an organic-looking cave. You can also tinker with other factors like the constants 4 and 5 in the formula above. Below is an example of the process starting from a random grid and applying the procedure 3 times.



Afterwards, you can always add other augmentations, such as:

- Using the floodfill algorithm to identify the regions within the cave (perhaps delete any region that is too small, thus removing "air bubbles").
- Connecting disconnected regions using an algorithm like Prim or Kruskal for minimum spanning trees and carving passages between each component based on which nodes the algorithm connects. (There is an example of this below)
- Instead of just 0 for walls and 1 for passages, what if there were multiple kinds of walls (i.e. dirt vs stone vs granite)?



8. **Example: Perlin Noise Terrain Generation**

The steps for Perlin's noise (the 2D version) are as follows:

(a) Associate certain points in the grid with a random gradient vector (i.e. 2 random numbers (x, y) such that $\sqrt{x^2 + y^2} = 1$)

(b) For each cell in the grid, find the four nearest gradient vectors.

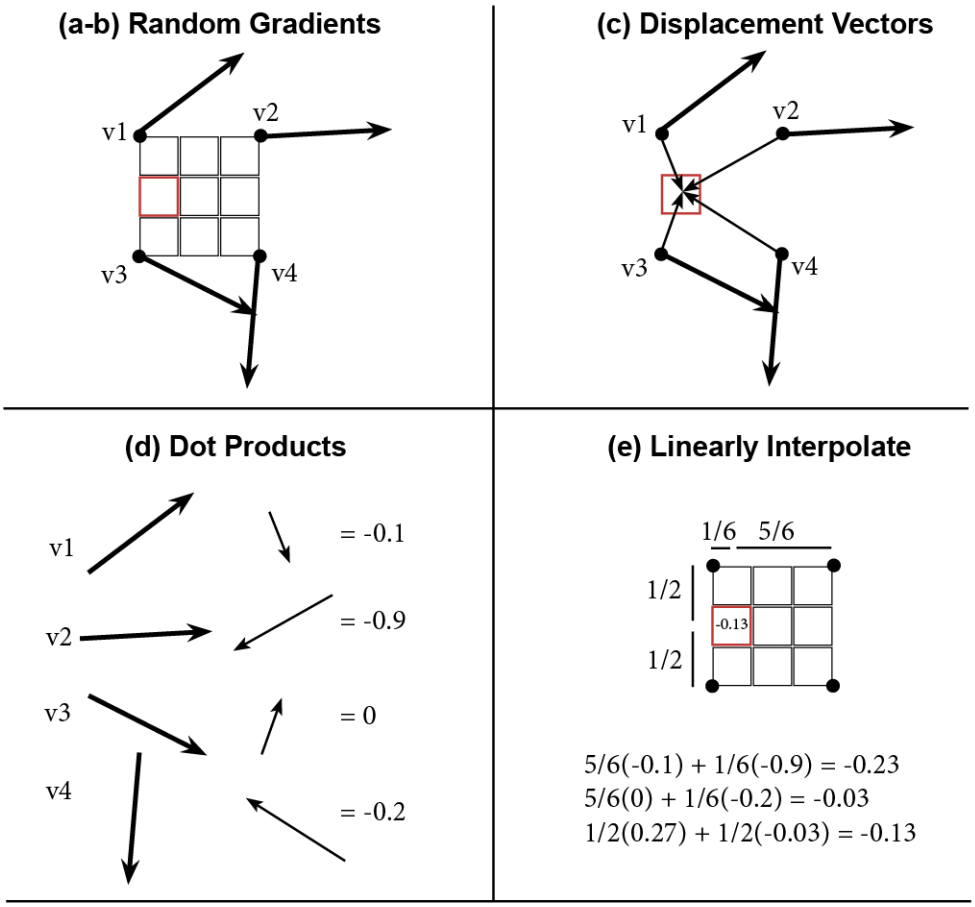
(c) Find the "displacement vector" from that point to the lattice points of those four vectors.

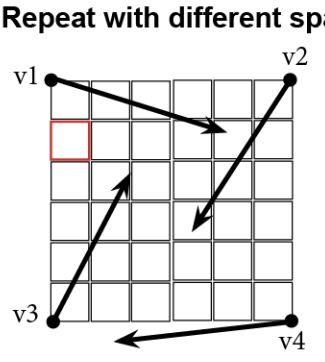
(d) For each of the four, take the dot product of the gradient vector and the displacement vector.

(e) Linearly interpolate the four values (i.e. take a weighted average based on which of the four is closest)

(f) **(Optional):** repeat steps (a)-(e) several times with different gradient vectors that are spaced out differently, adding up all the values.

(g) Use the values for each cell in the grid as the height or color of your terrain.





9. **Where to start looking**

Always try to avoid looking at code. Many good sources include code in other languages which could look quite similar or quite different to your eventual implementation. Also, there are countless variations you could add to these algorithms (multiple rounds of Perlin, different rules for cellular automata, taking weighted averages with Voronoi), and adding your own twists will help to differentiate your code from any code online.

You must cite all sources, even the ones below!

- 1D Terrain Generation:
  A guide on [**midpoint displacement**].
  A guide on [**1D Terrain Generation**] via interpolating random data.

- Cellular Automata:
  The wiki page for [**Conway's Game of Life**], an extremely famous cellular automata.
  A website that explains [**cellular automata for cave generation.**]
  Another article [**cellular automata cave generation**] and its second part which explains one aproach for [**linking unconnected caves**].
  The wiki page for the [**floodfill algorithm.**]

- Voronoi/Worley Noise:
  The wikipedia pages for [**Voronoi diagrams**] and [**Worley noise**].
  A guide on implementing [**Voronoi Noise**] with some variations.

- Diamond Square Algorithm:
  The [**wikipedia page**] for the Diamond Square algorithm.
  A guide explaining the [**Diamond Square Algorithm**], as well as several concepts that build up to it (i.e. midpoint displacement) which may be useful for similar types of terrain generation.

- Perlin and Simplex Noise:
  The wikipedia pages for [**Perlin Noise**] and [**Simplex Noise**].
  A guide explaining [**Perlin Noise**]
  A guide explaining [**Simplex Noise**]