The concept behind terrain:

- When creating a computer game, we want to be able to traverse a 3D world, in which we want many numerous aspects to change all at the same time all located simultaneously at the same coordinates, some of these characteristics of the world include but are not limited to:
    - The type of ground of the platform
    - The height of the platform
    - I was a nice smooth ground without holes
    - The steepness of the ground
    - Where specific objects from the games library start
    - We want these objects to me able to move location and change real-time values
    - We want the graphics to be maximised for quality
    - My cache isn't large enough to generate the entire infinite world at once
    - We want to be able to interact with the world as we are close to entities
    - I might want to generate a world to my own configured specifications
    - We might want a day-night cycle with a period
    - We may want world events randomly selected based on environment
    - Crouch or prone when you want to
    - I want to edit what keys on my keyboard do what and my sister wants to do the same for hers on her account
    - That was a good world - my cousin wants to play a fresh version
    - I want the density of trees to clear up into flowers on the same ground
    - The projection of light from the sun should project shadows consistently
    - It's a shame when the world generation won't generate at the distance I want
    - That one block of water shouldn't be flying over there.
    - The acceleration should tend towards the floor increasing proportionally as I jump off the but not affect my lateral velocity at all.
    - When I go low enough down, after a certain, I want certain objects to varyingly spawn but always in fair clusters
- Additionally, it is worth noting that when we have played for a sufficient amount of hours, we want to be able to return to the exact setup of the world without it taking up excessive space on our harddrive. We don't want to render millions of the exact same high resolution textures taking up TBs of storage.

Thankfully, there exists an optimal solution to satisfy all of the complicated criteria I layed out above all at the same time.

When we generate the exact world we want at the start, the precise set of our configuration of those values on the sliders are each input as parameters into the algorithm that underpins the game engine that constitutes a function of an appropriate modelling of relevant combinations to functions containing various degrees of each. The resulting world generation is consistent, continuous, appearing appropriately random and you can save to a small world file.

Our terrain initiates by beginning the generation from an origin location at the 0 time counter, which is logically the x-y coordinate pair of (0,0). When the loading screen is displayed, the result of our rendered world is being systematically assembled. The algorithm generates a grid consisting of all the possible combinations of pairs of coordinates bounded by a function dependent on your reference point (initialising at the origin but regularly updated as the sum of ticks hits specific values) and ranging in a function with correlation to difference in magnitude of the render distance stored from our settings continuous beginning at our most recently cached x-y.pair with some additional combination of our resolution to map the world we see.

Every x and y of the coordinate set is mapped through as some systematic function stemming from our reference.

—-----------

Terrain is split up into different areas and then put back together.

1. Biome type
2. The height of the ground at any given point
3. Cave structure

—------

Midpoint Displacement

Purpose: primarily 1D generation to create a **2D landscape** but also for graphical applications (i.e. rivers/ravines/paths).

Procedure:

1. Select 2 random points
2. calculate the midpoint
3. Randomly displace the midpoint within a set range of values, either in the direction of an axis or in the direction perpendicular to the line itself.
4. Iteratively/recursively repeat steps (1-3) between every connected point until the segments widths are equal to or less than 2 pixels.

Variables:

- The range of possible displacement values
    - Initial - how high/steep?
    - Consecutive - smoothness optimisation

Information Source

Landscape generation using midpoint displacement – Bites of code (wordpress.com)

—-------------

Cellular Automata

Purpose: start with a specific order of initial conditions (seed) from which rules are procedurally followed at each given discrete time (ticks) to result in the exact same construction for any given tick value, upon recreation (i.e. each iteration is a pure function of the previous). Useful for **cave generation**.

Procedure:

1. Use the seed to create a grid containing a unique set of live coordinates/cells.
2. Iterate the current set through an algorithm to create a new set x times such that:

   a. A live cell remains if 2 or 3 adjacent cells are live
   b. A dead cell becomes live if strictly 3 adjacent cells are live
   c. All other live/dead cells are now dead
3. (optional) Floodfill - fill all the secondary living areas as appropriate by setting nodes and filling adjacent cells one after another until the classification changes

Variables:
- Seed - initial state classification
- Birth/death limits - algorithm progression
- Total ticks - # iterations

Information source:
Conway's Game of Life - Wikipedia

—----
Voronoi/Worley Noise
Purpose: **biome generation.**

Procedure:
1. Randomly distribute a set of feature points, 1 in each cell of a 2D grid of set resolution, usually done as procedural noise to mimic a poisson disc.
2. Find the direction vector of each given location to its the nearest feature point
3. Gradient or colour the 2D space at each coordinate based on the magnitude of its direction vector.

Variables:
- Resolution of the grid

Information source:
Worley noise - Wikipedia

—---------
Diamond-Square
Purpose: generate **3D heightmaps** from a 2D grid.

Procedure:
1. From a 2D square array (size $2^n + 1$) set the 4 corners with initial values.
2. Diamond step - for the midpoint of the each square of coordinates, average the 4 corner values with some additional random value. If an edge, use a 4th value from the other side of the array (wrap).
3. Square step - for each diamond, set its value to be average of its 4 corners plus some additional random value. If an edge, use a 4th value from the other side of the array (wrap).
4. Repeat steps (2-4) but multiplying the random value by (2^{-h}, $0<h<1$)

Variables:
- Array size n - size of the map

- Initial values
- h - roughness of the terrain (low being rougher)

Information source:
Diamond-square algorithm - Wikipedia

—-----
Perlin/Simplex Noise

Purpose: generating **terrain textures** for any number of dimensions (Perlin) but particularly between 2-5 (Simplex) where computational complexity is reduced

Procedure breakdown:

1. Coordinate Skewing
2. Simplicial Division
3. Gradient Selection
4. Kernal Summation

Procedure:

1. Input coordinates are transformed using $x' = x + (x + y + \cdots) \cdot F,$
$$F = \frac{\sqrt{n+1}-1}{n}.$$
2. The resulting coordinates determine which skewed unit hypercube cell the input point lies in, (x_b' = floor(x'), y_b' = floor(y'), ...), and its internal coordinates ($x_i' = x' - x_b'$, $y_i' = y' - y_b'$, ...).
3. Assort the values of the internal coordinates, descending from the maximum to determine which skewed simplex the point lies in.
4. The resulting simplex is composed of the vertices corresponding to an ordered edge traversal from (0, 0, ..., 0) to (1, 1, ..., 1), of which there are *n*! possibilities, each of which corresponds to a single permutation of the coordinate.
5. Each simplex vertex is added back to the skewed hypercube's base coordinate and hashed into a pseudo-random gradient direction.
6. The contribution from each of the $n + 1$ vertices of the simplex is factored in by a summation of radially symmetric kernels centered around each vertex. First, the unskewed coordinate of each of the vertices is determined using the inverse formula
$$x = x' - (x' + y' + \cdots) \cdot G, \; G = \frac{1 - 1/\sqrt{n+1}}{n}.$$
7. This point is subtracted from the input coordinate to obtain the unskewed displacement vector.
8. Compute the extrapolated gradient value using a dot product.
9. Determine d^2, the squared distance to the point.
10. Each vertex's summed kernel contribution is determined using the equation
$$\left(\max(0, r^2 - d^2)\right)^4 \cdot \left(\langle \Delta x, \Delta y, \ldots \rangle \cdot \langle \operatorname{grad} x, \operatorname{grad} y, \ldots \rangle\right),$$

Variables:

- r^2 is usually set to either 0.5 or 0.6

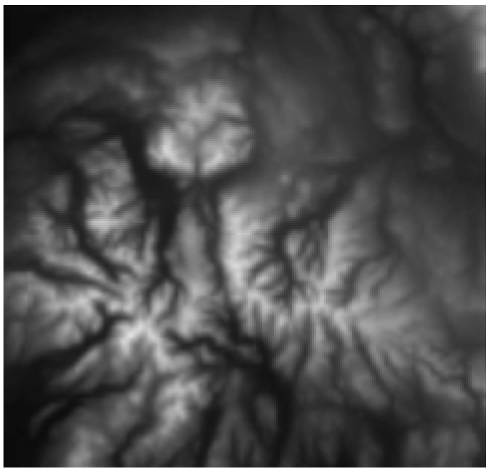
- Set of input coordinates

Information source:
[Simplex noise - Wikipedia]

—-------------
[Defra Survey Data Download]
[Defra Spatial Data Download]
Nz26ne
Longitude
-1.6306
Latitude
54.9678

[Coverage map viewfinderpanoramas.org]

—----------



Data (include river) - pca (vary the pca) - invert to original data

How much detail is lost? %