Voxel Environments

Jacob Howell

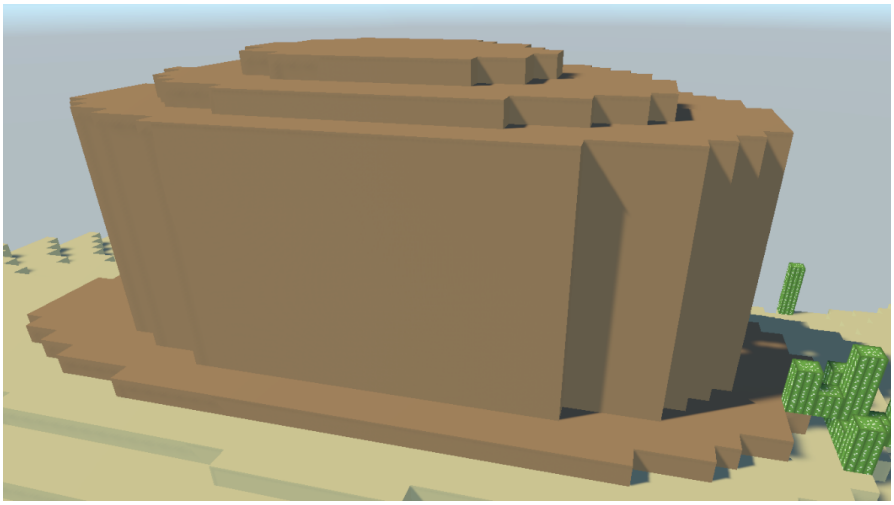
Abstract

The goal of this project was to build three procedurally generated voxel environments based on real world biomes. Within the contexts of this project, procedurally generated is defined as being created at runtime without direct interaction from the user and a voxel is defined as a three dimensional cube that does not have a static, or pre-runtime, model or mesh. The three chosen real world biomes are: a desert, a boreal forest, and a deciduous forest.

The project is available in completed form at: https://theaviator559.itch.io/voxel-environments .

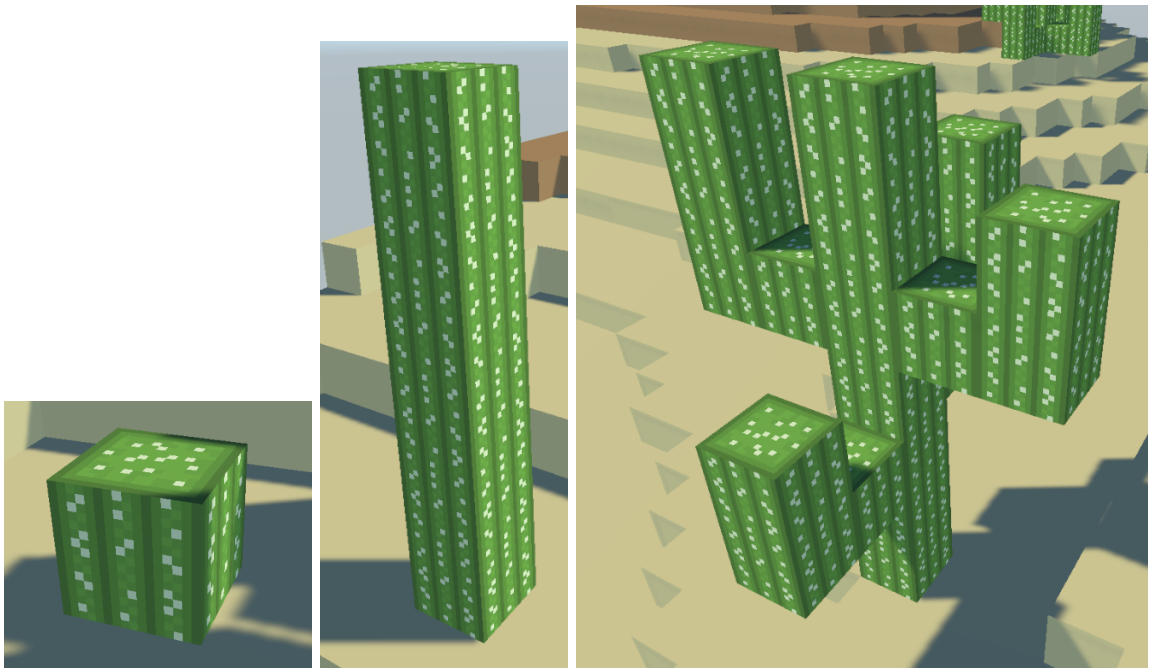
Overview of each biome

The desert biome can be characterized by a severe lack of moisture and humidity as well as a history of being an ancient seabed. These characteristics result in vast sand dunes, towering mesas, playas or salt flats, and a variety of cactus species. Mesas are tall outcroppings made of a reddish stone(1) while playas are flat areas of condensed salt(2). The implemented types of cacti are a standard pillar cactus, barrel cactus, and saguaro cactus(3).



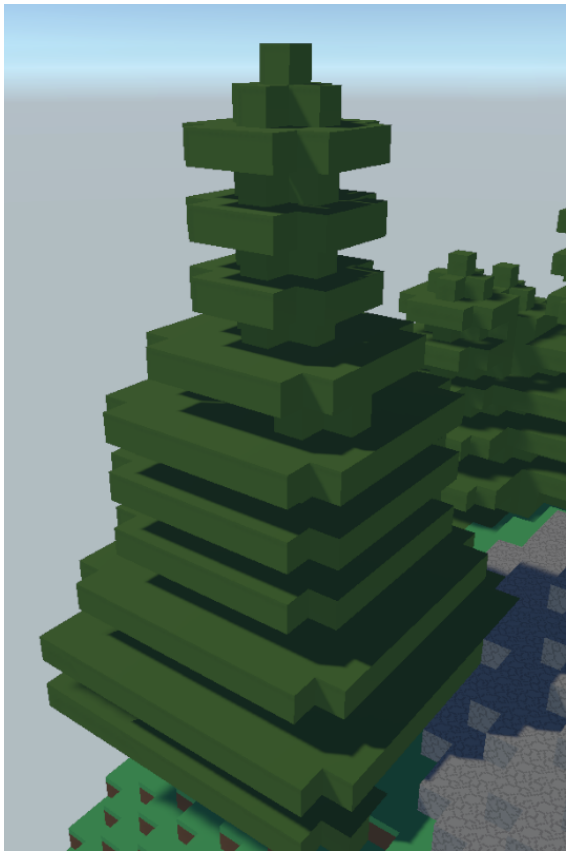
(1)

(2)

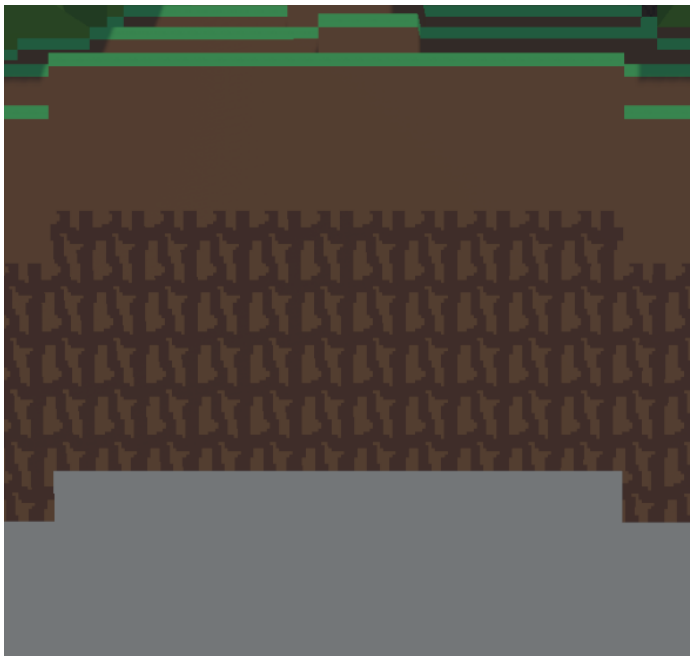


(3)

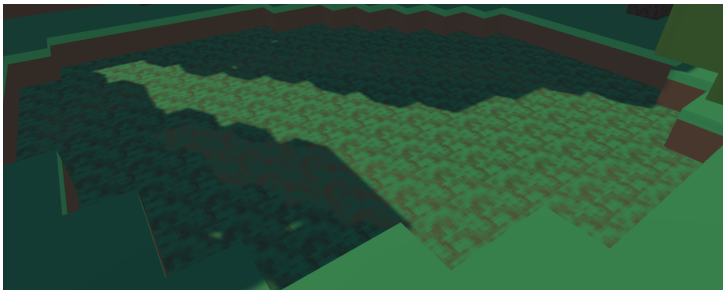
The taiga biome is characterized by higher altitudes, high humidity, and low temperatures. These characteristics result in larger fir trees, shallow dirt and deep permafrost, shallow grassy bogs known as muskegs, and a visible tree line on larger hills. The fir trees(4) grow noticeably larger due to the relative abundance of soil nutrition and the fact that they do not shed their leaves allowing for year round growth. The shallow layers of dirt and deep permafrost(5) result from the combination of high humidity and low temperatures. The muskegs(6) form from moisture that is not able to be absorbed by the ground. The tree line(7) is a result of the altitude reaching a point where the available oxygen is too low to support much, if any vegetation.



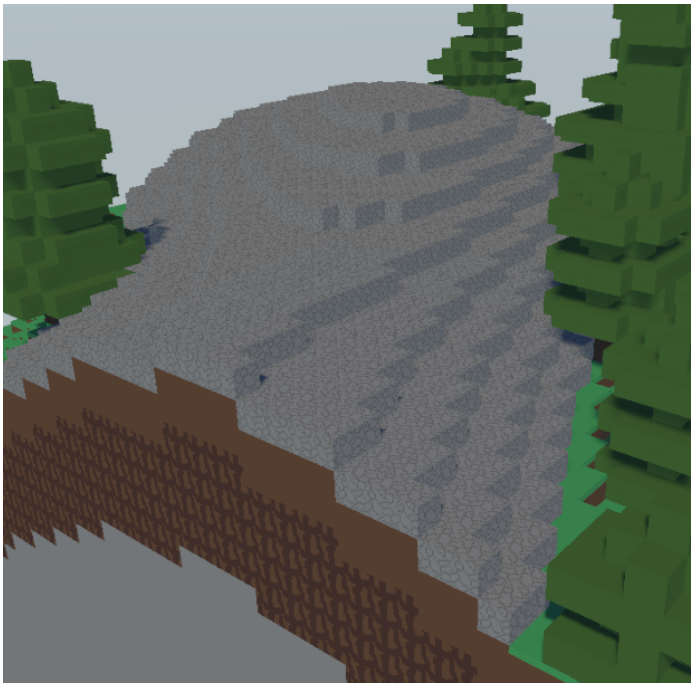
(4)



(5)

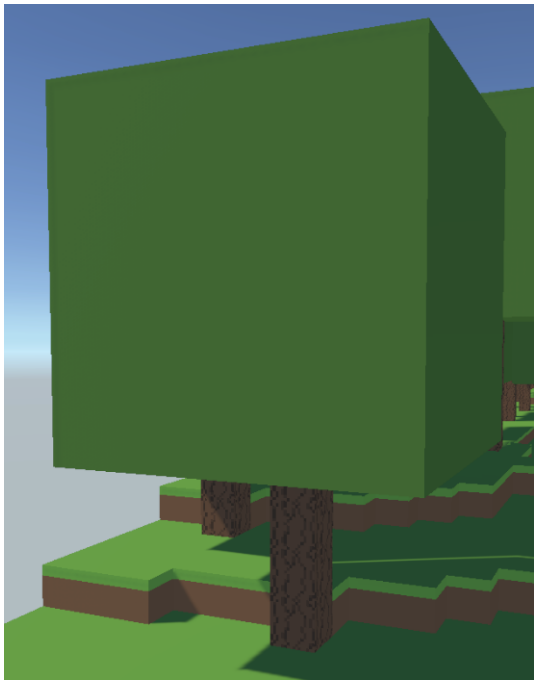


(6)



(7)

The deciduous forest biome is characterized by moderate temperatures and humidity, allowing life to flourish. These characteristics result in rolling hills and a large amount of oak trees. The oak trees(8) get placed using a custom forest growth model that is unique to the deciduous forest biome implementation.



(8)

Generation logic for each biome

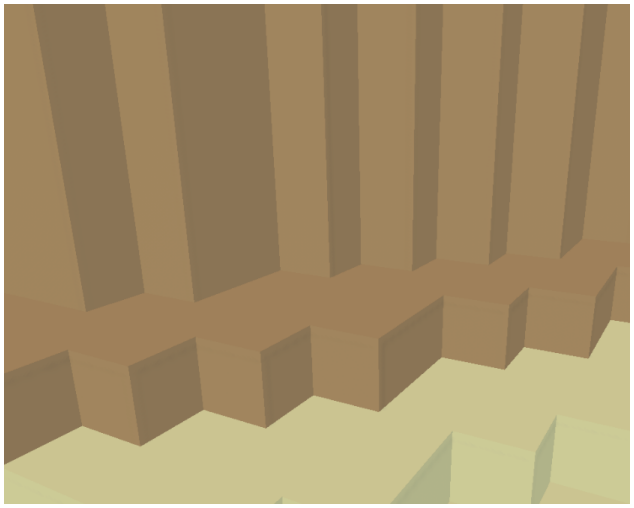
Desert

The first step in generating the desert environment is to build the height map using values determined by the terrain weight, terrain scale, solid ground height, and seed control variables(9). The height map is used to determine the vertical height of the surface and at what height to place cacti. It is also used to determine the minimum height of mesas(10) as well as the upper and lower bounds for playas(11). Any point in the height map that is above the minimum mesa height has the mesa height offset control variable added to it and once that is complete for every point the height map, the minimum mesa height is reduced by the mesa bleed off control variable to help ground the mesas in the environment(12). Once the height map is constructed the biome is responsible for two more tasks: reporting the type of voxel at any given position, and determining where to place cacti.

$$TerrainWeight \times PerlinNoise(Seed, TerrainScale) + SolidGroundHeight \quad (9)$$

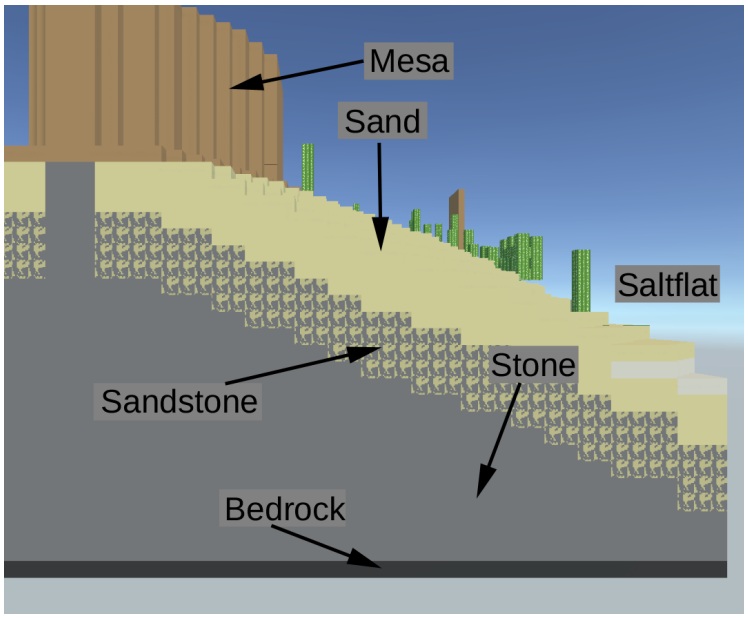
$$MinMesaHeight = MaxHeight - (MaxHeight \times MesaChance) \quad (10)$$

$$MaxPlayaHeight = MinHeight - (MinHeight \times PlayaChance) \text{ \&}$$
$$MinPlayaHeight = MinHeight \quad (11)$$



(12)

The biome is responsible for reporting the type of voxel to place at any given position in the generated area based on the vertical height of the position. If the vertical height is equal to zero, then the voxel type at that position is set to bedrock. If the vertical height is greater than the corresponding value in the height map, then the voxel type is set to air. If the vertical height is greater than or equal to the minimum mesa height, then the voxel type is set to mesa. If the vertical height is between the upper and lower bounds for the playas and there is no solid voxel two units above the given position, then the voxel type at the given position is set to salt flat. If the vertical height is between the surface height and the upper soil depth, then the voxel type is set to sand. If the height is between the upper soil depth and the middle soil depth, the voxel type is set to sandstone. If all of the above are false, then the voxel type is set to stone(13).



(13)

In order for a cactus to be placed at a given position, it must pass a couple of checks. The trivial first check is that the position is at the surface height. The second is the overall cactus zone threshold while the third is the cactus placement threshold(14, 15). These threshold checks serve to clump the cacti together rather than scatter them over the generated area.

$$PerlinNoise(position, -seed, treeZoneScale) > treeZoneThreshold \quad (14)$$

$$PerlinNoise(position, -seed, treePlacementScale) > treePlacementThreshold \quad (15)$$

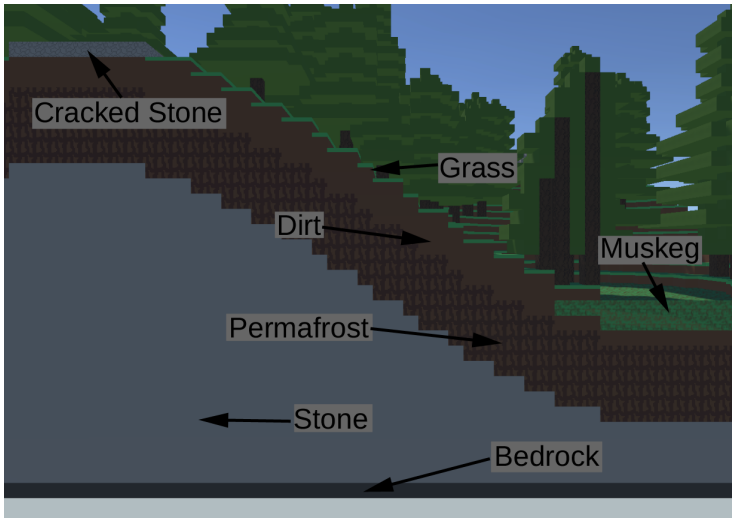
Taiga

The generation logic for the taiga, or boreal forest, biome follows similar logic to the desert biome with some minor changes. The height map is populated using the same formula as the desert biome. The height map is also used to determine the tree line height(16) and the upper and lower bounds for muskegs(17). Once the height map is constructed the biome is responsible for two more tasks: reporting the type of voxel at any given position, and determining where to place fir trees.

$$TreeLineHeight = MaxHeight - (MaxHeight \times TreeLinePercentage) \quad (16)$$

$$MaxMuskegHeight = MinHeight - (MinHeight \times MuskegChance) \text{ \& } MinMuskegHeight = MinHeight \quad (17)$$

The biome is responsible for reporting the type of voxel to place at any given position in the generated area based on the vertical height of the position. The bedrock and air position rules are the same as in the desert biome. If the vertical height is between tree line height and ground height, then the voxel type is set to cracked stone. If the vertical height is between the upper and lower bounds for the muskegs and there is no solid voxel two units above the given position, then the voxel type at the given position is set to a structural placeholder for muskegs, this is explained in the Voxel Types section. If the vertical height is between the surface height and the upper soil depth, then the voxel type is set to taiga dirt. If the height is between the upper soil depth and the middle soil depth, the voxel type is set to permafrost. If all of the above are false, then the voxel type is set to stone(18).

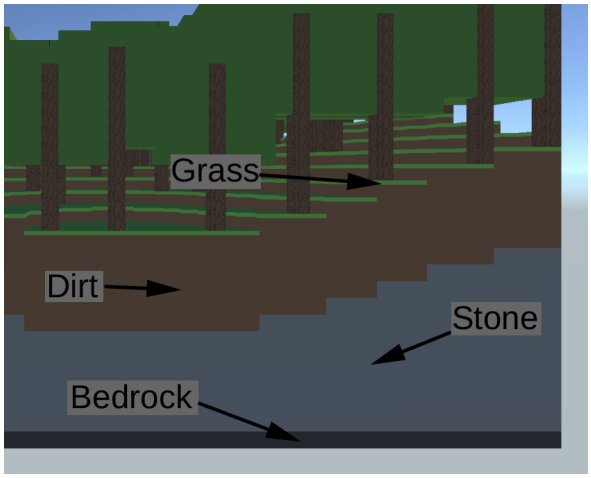


(18)

In order for a fir tree to be placed at a given position, it must pass a couple of checks. The trivial first check is that the position is at the surface height. The second is the overall fir tree zone threshold while the third is the tree placement threshold(14, 15). These threshold checks serve to evenly space the fir trees over the generated area.

Deciduous Forest

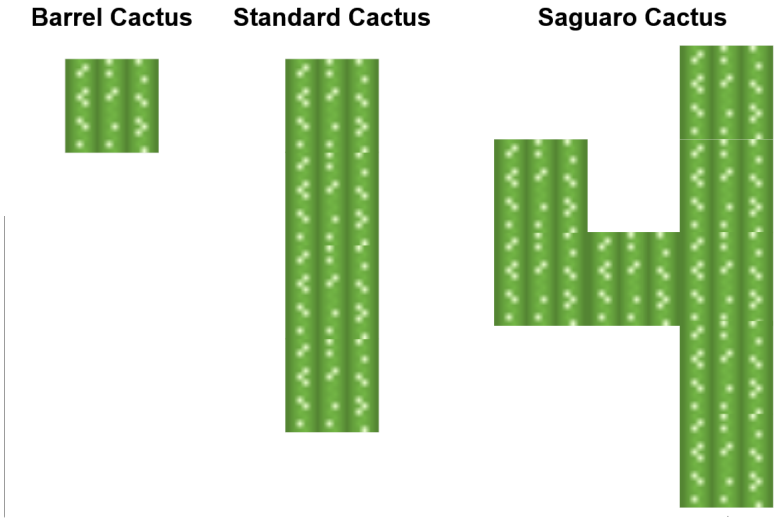
The generation logic for the deciduous forest biome is simpler than the other two biomes since most of the work went into the forest growth model. The height map is populated using the same formula as the other biomes. Once that is done, the biome creates the biomass map, seeds it with an initial number of trees, and then populates the forest using the forest growth model. The biome is also responsible for reporting the type of voxel to place at any given position in the generated area based on the vertical height of the position. The bedrock and air position rules are the same as in the other biomes. If the position is at ground height, the voxel type is set to grass. If the vertical height is between ground height and middle soil depth, then the voxel type is set to dirt. If all of the above are false, then the voxel type is set to stone(19).



(19)

Tree and cactus construction

There are three types of cacti that can be placed in the desert biome: the standard pillar cactus, the barrel cactus, and the saguaro cactus(20). The type of cactus that is placed is based on the placement chance for each type. The standard pillar cactus builds itself up vertically from directly above ground level by placing one cactus voxel on top of another up to between minimum and maximum cactus height. The saguaro cactus builds itself up like the pillar cactus and then adds between one and four arms that branch out of the main structure.

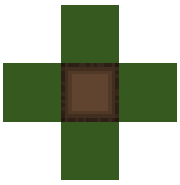


(20)

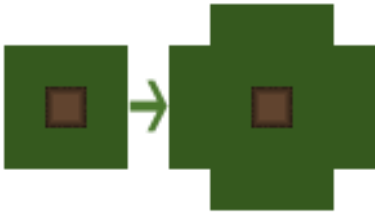
The fir trees(21) are constructed in a similar manner. First the center trunk is built up from ground height to between the minimum and maximum tree height, followed by a single fir leaf voxel on top of the trunk. This is then followed by ‘growing’ the leaves downward in an

alternating pattern of placing a layer of leaves directly next to the trunk in the cardinal directions(22) and placing a circular layer of leaves around the trunk that expands based on a provided growth value(23).

(21)

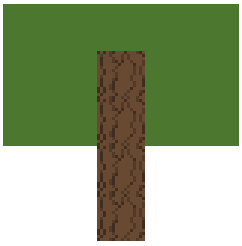


(22)



(23)

The oak trees(24) are constructed similarly to the fir trees. First the trunk is built up from the ground to between minimum and maximum tree height and a single leaf voxel is placed on top of the trunk. From there, the rest of the leaves are placed around the trunk in a simple cube.

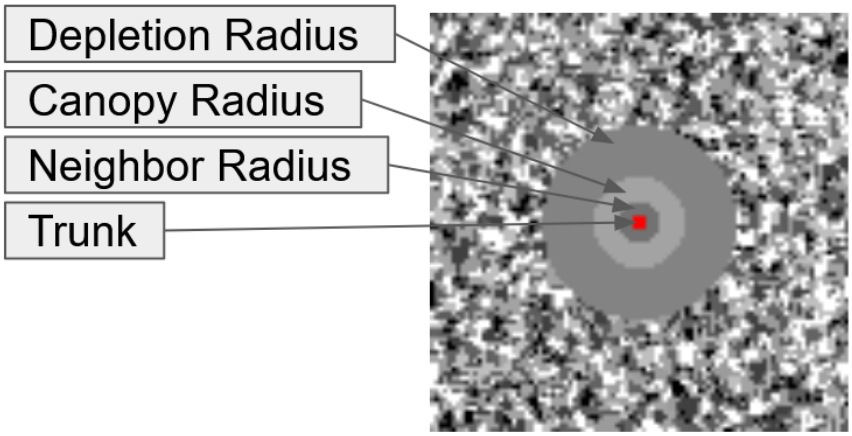


(24)

Forest growth model

The forest growth model runs off of a generated biomass map, which represents the soil fertility at any given position in the map. The biomass map is filled with random values between 0 and 100 and the growth model then places a number of initial trees to jump start the model.

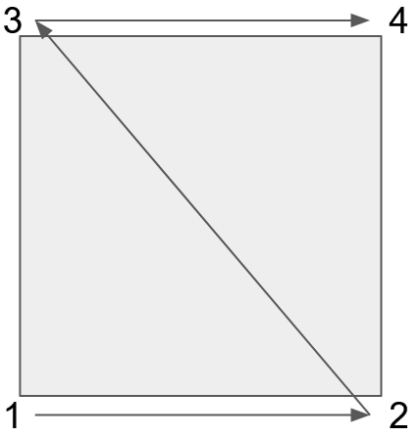
When a tree is placed, the values in the biomass map are updated; the position where the tree is placed is set to -1 to mark a tree trunk for later use, positions within the neighbor radius have their value reduced by a large amount, positions within the canopy radius have their values reduced by a moderate amount, and positions within the depletion radius have their values reduced by a light amount(25). The various radii serve to discourage but not entirely prevent trees from clumping together to mimic how real world forests expand and grow.



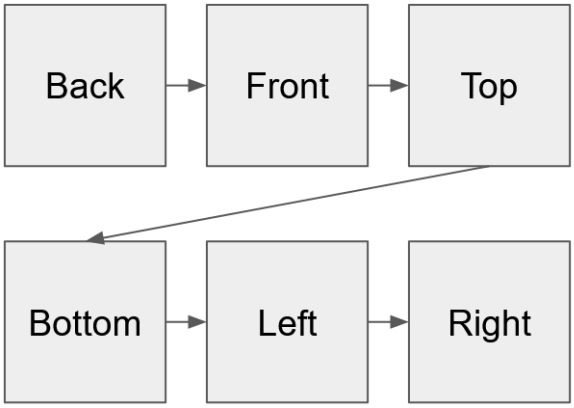
(25)

Rendering and Texturing

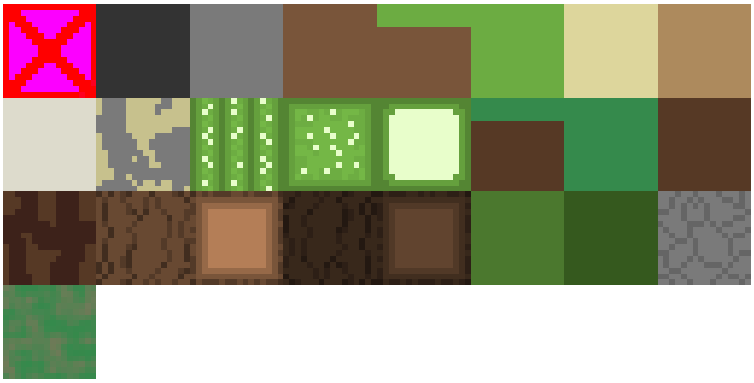
Optimization problems prevent voxel environments from being constructed from individual primitive cubes, resulting in the need for custom meshes. In this implementation, the generated area is split up into a number of ‘chunks’ which are responsible for creating, maintaining, and rendering the created mesh. The mesh creation process can be broken down into checking if a given side of a voxel is visible and then rendering it following a predetermined pattern(26, 27). Texturing the created mesh involves a process similar to creating an individual side of the voxel and requires using a texture atlas(28) which contains all necessary textures in a single image. The any given voxel has a specific type and at least one corresponding texture which is determined by the user and the ‘human readable’ texture number. The texture number is then used to determine the exact coordinates of the required texture within the texture atlas and those coordinates are passed into the mesh creation process and applied to the finished mesh.



(26)



(27)



(28)

Voxel Types and Explanation

To facilitate visually distinct voxels and more flexible generation logic, a voxel type system was developed. Each type of voxel can have a unique texture for each side of the cube as well as toggle for whether it should be rendered and whether it should render its neighbors. Due to their nature of being transparent, transparent blocks, specifically the muskeg block,

needs to be rendered after all other blocks. This results in needing a temporary placeholder block to allow all the non-transparent blocks to be generated and once done, the placeholders will be replaced with the intended transparent blocks.

NAME	TEXTURES*	Render Self?	Render Neighbors?	NOTES
Null	0 for all faces	Yes	No	Safety, Error catch block
Structure Placeholder	0 for all faces	No	Yes	Placeholder for transparent blocks
Air	0 for all faces	No	Yes	Empty block with no dedicated texture
Bedrock	1 for all faces	Yes	No	Bottom of the world, end of generation block
Stone	2 for all faces	Yes	No	Standard subsurface block
Dirt	3 for all faces	Yes	No	Deciduous forest subsurface block
Grass	4,4,5,2,4,4	Yes	No	Deciduous forest surface block
Sand	6 for all sides	Yes	No	Desert surface block
Sandstone	9 for all sides	Yes	No	Desert subsurface block
Mesa	7 for all sides	Yes	No	Desert Mesa feature block
Saltflat	8 for all sides	Yes	No	Desert Playa feature block
Taiga Dirt	15 for all sides	Yes	No	Taiga subsurface block

Taiga Grass	13,13,14,15,13,13	Yes	No	Taiga surface block
Permafrost	16 for all sides	Yes	No	Taiga subsurface block
Cactus	10,10,11,12,10,10	Yes	No	Cactus feature block
Oak Log	17,17,18,18,17,17	Yes	No	Oak Tree feature block
Fir Log	19,19,20,20,19,19	Yes	No	Fir Tree feature block
Oak Leaves	21 for all sides	Yes	No	Oak Tree feature block
Fir Leaves	22 for all sides	Yes	No	Fir Tree feature block
Cracked Stone	23 for all sides	Yes	No	Taiga tree line feature block
Muskeg	24 for all sides	Yes	Yes	Semi transparent, Taiga muskeg feature block

* Textures are listed in the order: Back, Front, Top, Bottom, Left, Right