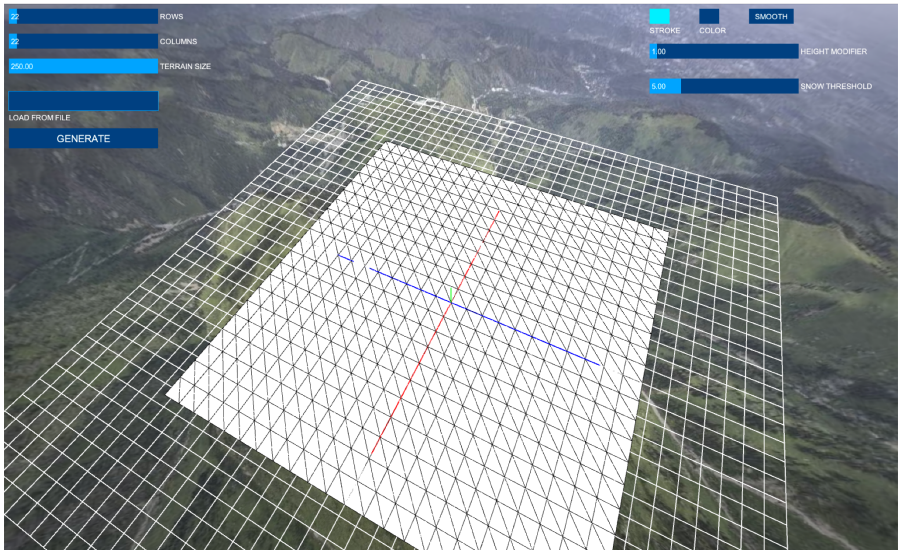
Procedural Mountain Generation

Cory Grossman
Nikhil Agarwal

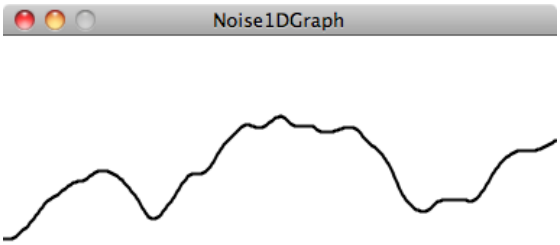
Compiling Guide

Implemented Functionality

1. **Plane Generation**
   - Create a grid of triangles based on the users desired amount of rows and columns
   - Triangles are indexed in vertex array
   - Each array has a x, y, z value



2. **Perlin Noise Algorithm**
   Perlin noise is a popular procedural generation algorithm used to generate things like textures and terrain procedurally, meaning without them being manually made by an artist or designer. Perlin noise has a more organic appearance because it produces a naturally ordered ("smooth") sequence of pseudo-random numbers. The graph below shows Perlin noise over time, with the x-axis representing time; note the smoothness of the curve.

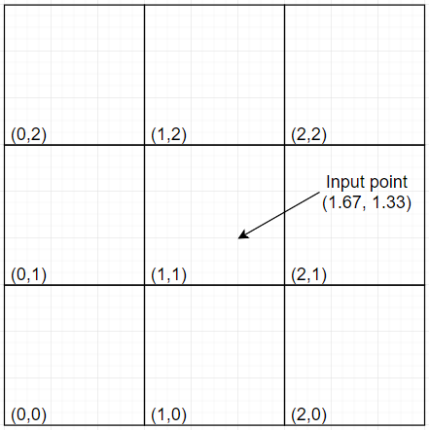


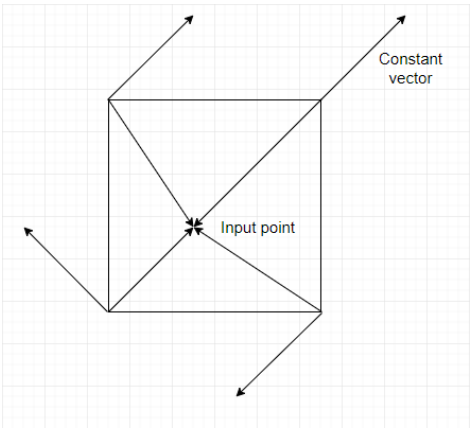
- Take floating point parameters as input and return a value in a certain range.
- Range: [sqrt(n)/2, sqrt(n)/2], where n is the dimension.
- For a 2D there would be 2 parameters x and y (positions). For texture generation x and y would be the coordinates of the pixels in the texture.
- Loop through every pixel in the texture calling the perlin noise function for each one.

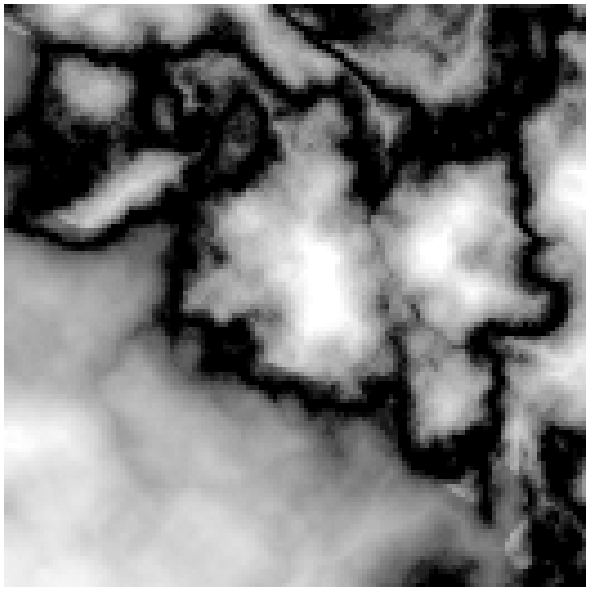
As it can be seen in the above figure, each pixel don't just have a random color, instead they follow a smooth transition from pixel to pixel and the texture don't look random at the end. That is because Perlin noise has a property that if 2 inputs are near each other, the results of the noise function will be near each other too.

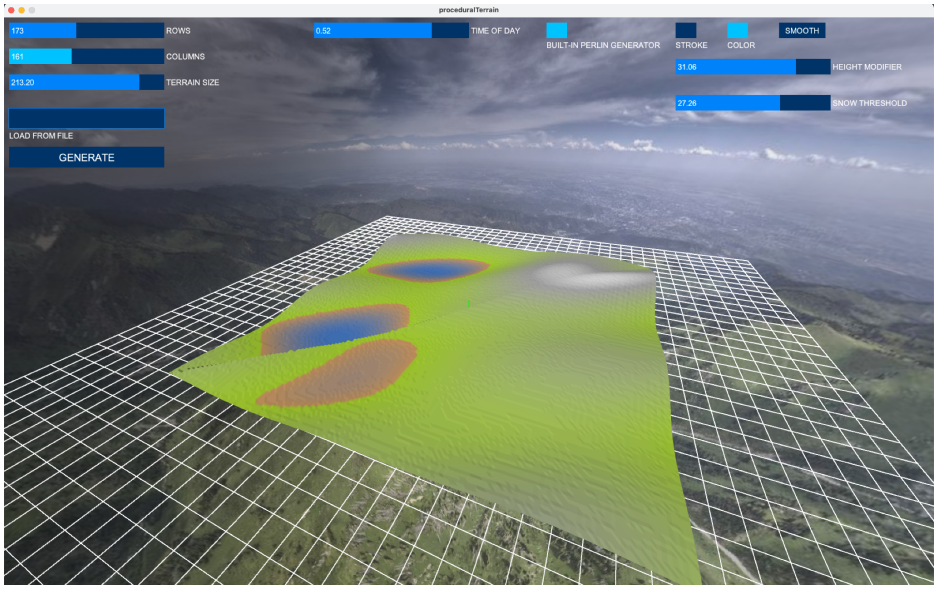
The inputs are considered to be on an integer grid. Each floating point input lies within a square of this grid. For each of the 4 corners of that square, we generate a value. Then we interpolate between those 4 values and we have a final result. The difference between Perlin noise and value noise is how those 4 values are obtained. Where value noise uses a pseudo-random number generator, Perlin noise does a dot product between 2 vectors.

The first vector is the one pointing from the grid point (the corners) to the input point. The other vector is a constant vector assigned to each grid point. That one must always be the same for the same grid point, but it can change if you change the seed of the algorithm .

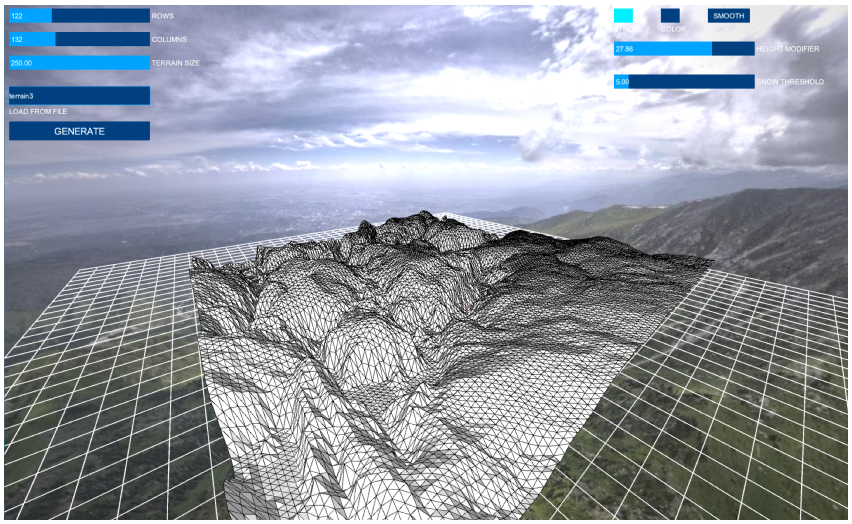

(0,2)
(1,2)
(2,2)
Input point
(1.67, 1.33)
(0,1)
(1,1)
(2,1)
(0,0)
(1,0)
(2,0)


Constant
vector
Input point

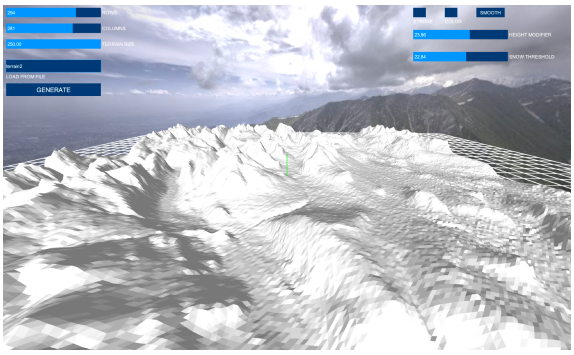


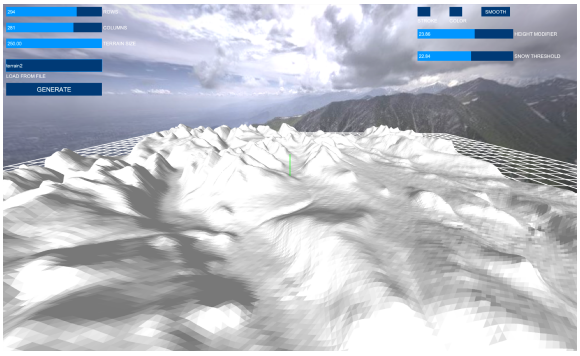

proceduralTerrain
173
ROWS
161
COLUMNS
213.20
TERRAIN SIZE
LOAD FROM FILE
GENERATE
0.52
TIME OF DAY
BUILT-IN PERLIN GENERATOR
STROKE
COLOR
SMOOTH
31.06
HEIGHT MODIFIER
27.26
SNOW THRESHOLD

3. **Height Maps**
    - A pre-determined image is loaded in the program
    - The height value are gathered from the red value of the pixel.
    - Piels of the image are mapped onto the amount of rows and columns.

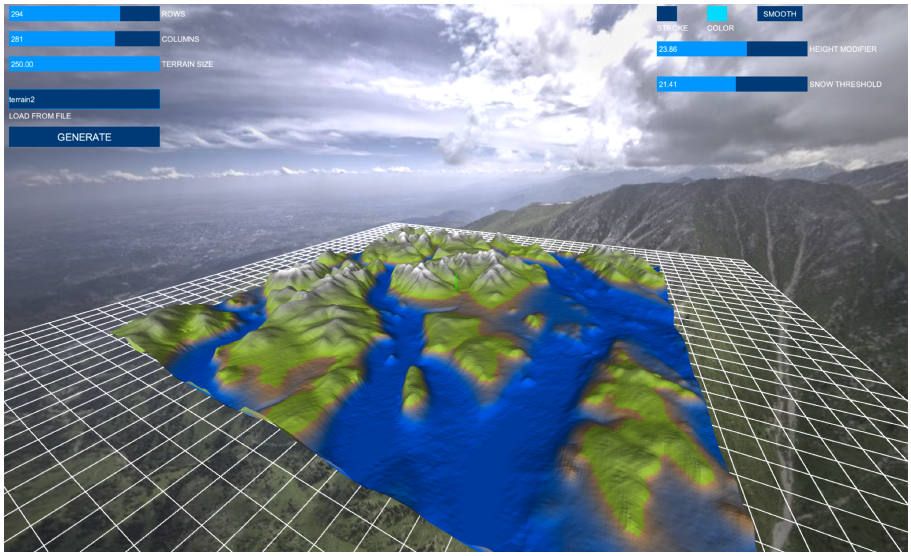


4. **Plane Smothing**
    - The smoothing algorithm works by averaging put the height on each vertex based on its neighboring vertex heights. This makes each vertex closer in height value to its neighboring vertices.



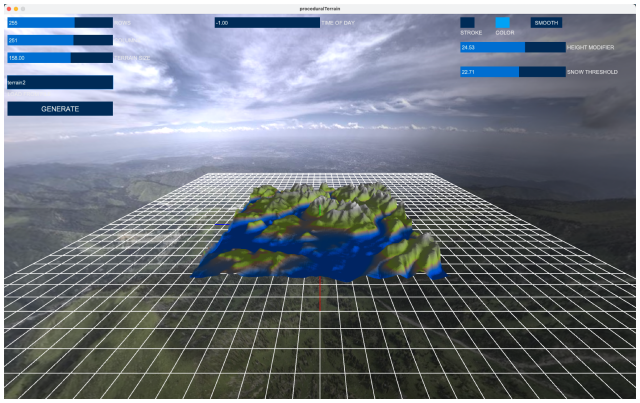


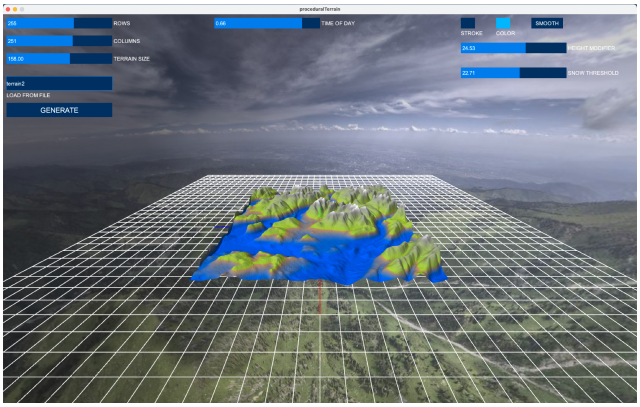
5. **Plane Texturing**
    - Textures are generated based on a vertex's height value.
    - If a vertex is at or above a snow threshold value, then the vertex is coles white.
    - Ther is a slider that the user can influence at what height the snow should be at.
    - Colors are interpolated between each colors height value for a blended appearance.



6. Environment Lighting

- An ambient of strength of 0.47 is used to fill a baseline illumination of the environment.
- A directional light of a strength 0.8 is used in the direction that the user can specify in the gui to act as a sun casting directional light on the environment.
- An hdri is loaded in to provid a background. However, the luminance values from it are not used in lighting the environment.





Program Logic

When first running the program, the user is met with an environment with a flat plane acting as the 3D grid. The user decides how detailed the terrain is by choosing the amount of rows and columns for the initialized plane. The user then decides to either load in a texture file (“terrain1.png”, “terrain2.png”, etc) or toggle the “Use Built-In Perlin Noise” to generate and load in an image texture. Once the user clicks the “GENERATE” button or presses the enter key, the program builds the plane of triangles and loads in the height values from the texture image into the y values of the triangle vertices. The user then can decide if they want to display triangle stroke, display environment color, move the vertices higher or lower based off a height modifier, and change the height at which snow appears. From there the user can also change the direction of light by using a slider. This emulates the rotation of directional sunlight throughout the day.

Acknowledgement

Built-In Perlin Noise: https://github.com/warmwaffles/Noise
Java Processing Documentation: https://processing.org/reference
ControlP5 Gui Library: https://www.sojamo.de/libraries/controlP5/
Cory Grossman Fall 2021 Intro to DAS Final Project