Introduction

Procedural generation allow content creation at scales which are otherwise very time consuming, if not impossible, to create by hand. An example could be generation of the trees in a forest to provide larger diversity and save on resources for content creation, or using it to produce the finer details of fur.

For this project the focus will be on terrain generation, more specifically generation of a mountainous terrain. Terrain is often difficult to produce since it has a lot of little details which have been shaped over thousands of years from erosion.

Method

Noise

The terrain is generated based on noise. It uses simplex perlin noise as it is a well-known gradient noise function and an improvement over perlin noise. The noise function is used as the basis for generating terrain as the gradient nature of it can be iterpreted as peaks and valleys through the landscape.

The noise function has the property that it is deterministic. The same output is produced given the same input and the output is always in the range of -1 to 1. If the input is scaled by a factor s then the output will approach white noise in $\lim s \to \infty$. This is because adjecent samples will be similar, but as s grows a larger the correlation between the samples become smaller as the distance is greater.

Scaling is used to make a more diverse landscape since having $s = 1$ results in a pattern with very little features as shown in 1a.

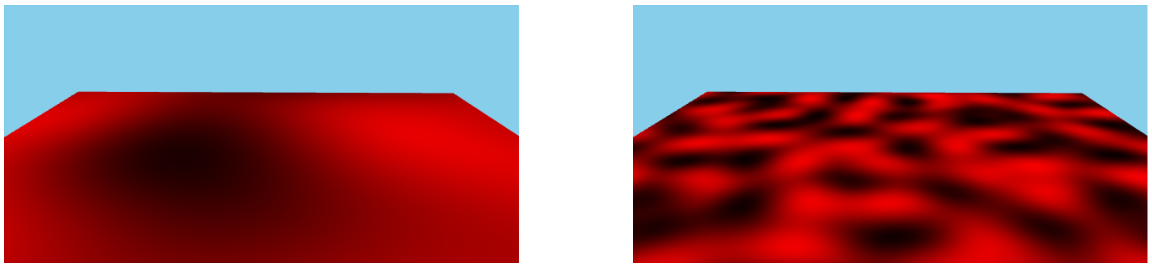
(a) Scale = 1 (b) Scale = 4

Figure 1: The simplex implementation at different scales illustrated on a quad

Fractional Brownian Motion

Simplex on its own does not generate very interesting results as shown in figure 2. Real mountains are not so smooth and have ridges and ueven terrain. To achieve a higher level of detail, we are making use of fractals. Fractals recursively add more and more detail to simple shapes in order to transform them into a more complex shape.

Specifically, we are using fBM(Fractional Brownian Motion) which samples a noise function multiple times, each times adding more detail[2]. The method takes four parameters: *p, H, lacunarity(l), octaves(θ)* where p is the sampling point, $H \in [0, 1]$ determines the ”smoothness” of the resulting curve with the most effect at $H = 1$. In $\lim H \to 0$ the result becomes more and more like white noise. lacunarity can be treated as a constant $l = 2$ and influences how much the influence of the next sample is changed by, with half the influence at each iteration if $l = 2 \wedge H = 1$. Finally, the octaves parameter determines the amount of samples done per result and thereby the underlying amount of details to be extracted. This is mostly a parameter that must be experimented with for good results. The higher the octaves the more details, but also higher the cost of each sample causing it to be a trade-off between quality and performance. The method used for calculating fBM is:

$$\sum_{n=0}^{n<\theta} noise(p * l^n) * l^{-H*n} \tag{1}$$

The result becomes more smooth than the underlying noise function because the sample is taken over a larger area($p * l^n$), evening out the result. This is because taking the average of multiple samples from a random distribution approaches the mean of the distribution as the amount of samples grow.

As mentioned earlier, parameter H effects the smoothness of the result. With each iteration, H dampens the influence of the the sample. This dampening can be thought of as a weighted mean across all samples. The effect of altering H with $l = 2$ is shown in figure 3. It seems that to get a good looking mountain range, we would need to have $H > 0.8$.

Generating Terrain

The base mesh for the terrain is a quad. By subdividing a quad multiple times into a grid, we can vary the height of each vertex vertex in the shader. We use snoise with a 2 dimensional input in order to traverse the terrain height across the quad.

Shadowmaps

Shadowmapping is used to have the mountain cast shadows. Shadowmapping is required for self-shadowing, which is required since the mountains are part of the terrain and the terrain is based on a single quad. The shadowmap is stored using bitshifting for heigher precision and sampled with antialiasing.

The bitshifting is required because the mountains span a large area with mountains that are both close and far from the light source. The extra precision helps with making the shadows more precise.

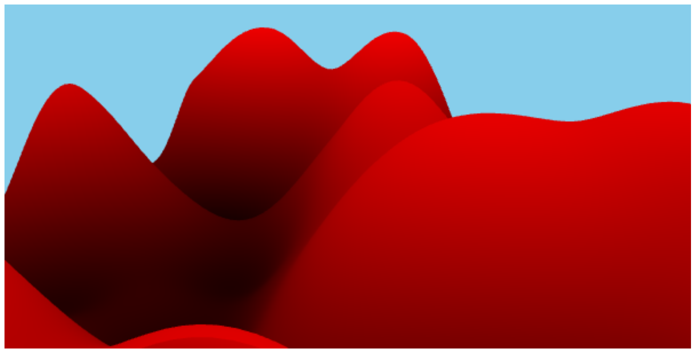

Figure 2: Terrain based on simplex using scale 2

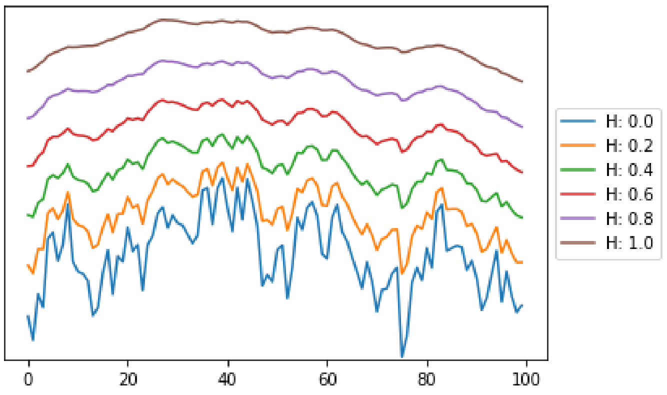

Figure 3: The influence of the H value on the smoothness of the output

The light is placed at an angle to the terrain, causing magnification due to projection on the shadowmap. Percentage-closer filtering[4] is used to combat the jaggilation of shadows caused by the magnification.

Lighting

The project includes lighting even though lighting was not a part of the initial agreement for the project. It was initially thought to be difficult to compute the normals for the generated terrain, but we stumbled upon[5] which is a technique based on Central Differences[3] that can be used to calculate the normals directly in the shader. It is used in mathematics to approximate the derivatives at a point. In 3D the approximation of normal $\hat{X}$ would be calculated as shown in equation 2. x and y are the coordinates in the terrain to calculate the normal at, k is the height, h is some small offset and f is the noise function used to compute k.

$$\hat{X} = \frac{X}{|X|}, X = norm(x, y, k) = \begin{pmatrix} f(x-h, y) - f(x+h, y) \\ f(x, y-h) - f(x, y+h) \\ 2 * k \end{pmatrix} \tag{2}$$

It works by finding the derivative in both the y and x axis in noise space and then using 2 * k for the z axis. The result is a normal because f produces a ”height” value for both x, y. The resulting output is an approximation of a normal pointing ”upwards”, based on the local trend for height in noise space around x and y.

Only ambient and diffuse lighting[1] is used in this assignment as the metallic look of specular lighting is not relevant for mountains. It could have been interesting to add reflections to the icecaps on the summits, but that it outside of the scope at this time. The diffuse lighting will be calculated using the normals generated using the method covered in previous paragraph. Only directional light will be used since it is supposed to simulate light from the sun.

Implementation

In this project the Z axis is considered upwards instead of Y(that is, Z denotes the height of the terrain). This is because the original quad was placed in the XY plane and it evolved from there. This change has no impact on the functionality of the program and even proved to be beneficial in some cases.

The overall flow of the program is as follows:

1. Setup WebGL, compile shaders and initialize shader parameters
2. Generate a 100x100 grid of quads
3. Create a framebuffer for shadowmapping
4. Render in a loop:
   (a) (Re)calculate view matrix based on rotation slider
   (b) Draw shadowmap using a camera with the position and direction of the light source
   (c) Draw the scene using the shadowmap produced

The terrain generation has primarily been generated in the vertex shader for realtime rendering. The shader contains the following steps:

1. Sample the fbm noise using x, y
2. Set the z-coordinate to be the result of step 1 and update the vertex position ($\mu = [x, y, fbm(x, y)]^T$)
3. Generate two noise samples for the fragment shader (more on that later)
4. Apply $P * MV * \mu$ and $lightMVP * \mu$ for gl_Position and shadow mapping, respectively
5. Estimate the normal at x, y using equation 2 for the fragment shader

Generating the grid

The grid is created using the function `CreatePlane(n)` which creates a $n \times n$ grid ranging from $(0, 0)$ to $(1, 1)$ in object space. This is done by placing vertices spaced $\frac{1}{n}$ apart on the XY plane. The plane is built using four indices for each subquad, which saves 2 vertices for each. It can be improved by reusing the vertices from adjecent subquads.

The plane is stored in an object containing the vertices and indices. This object can be passed onto the function `Commit` which goes through the object and creates a glBuffer for every attribute within. The data is uploaded as static data because the mesh is not changed since the terrain is generated in the shader.

Once the plane has been committed, another function `Bind` is used to ensure that the glBuffer is bound to the correct attribute pointers. The method also takes in an argument for which attribute pointers to bind to. This allow the same binding code to be used for both scene rendering and shadowmapping.

Rendering the scene

Shadowmap

The scene is first rendered from the position of the light in order to generate the shadowmap. The shadowmap is rendered in the same direction as the directional lightsource to simulate that the shadows are caused by it. The projection matrix has a FOV of 45 in order to see the whole terrain. The actual rendering is handled by the method `DrawShadowMap` which sets the uniform variables for the shader. After that, the frame buffer is attached and rendering performed. This produces a texture which can be used for rendering shadows in the scene. This is done by storing the value of the Z buffer for each pixel rendered, in the texture.

The Z buffer has a 10-bit precision, but the texture only support colors, each which only have a 8-bit precision. We have made use of bit-shifting to store the full 10-bit precision in the texture by distributing the 10-bit precision of the Z-depth across the colors by making use of their combined storage space, allowing the full 10-bit precision to be reconstructed from the texture.

Rendering the terrain

The vertex shader generates the terrain in object space. The generated quad's coordinate space ranges from $[0; 1]$ for both x and y. The terrain is generated in the range $[0; 1]$ as well, causing the full terrain to have nicely defined bounds. Had we instead generated the terrain in world space, transforms would not behave as expected as they could change the positions used from the XY plane to sample the coordinate space and hence some details might change. Instead, we finish the object by adjusting Z before we apply the MV matrix. This approach has the upsides: the bounds of the terrain model is well defined, the basis transforms are well defined and transforming the object is as expected. However, this approach also have the following downsides: One cannot use translation to traverse the terrain. This is not an issue for this project, but one might expect that moving the camera in one direction would show more of the landscape in that direction. That is after-all one of the benefits of procedural generation, but with this approach it will simply just offset the terrain.

The terrain itself is generated by multiplying a scaling parameter provided through a uniform variable with the XY-vertex coordinates in object space. The scaling parameter can be modified using a slider on the webpage to increase the mountain density. The scaled vertex coordinate is then used as input to the fBm noise function along with the H parameter as described previously. The H parameter can also be changed using a slider on the webpage and controls how smooth the terrain is. The other parameters, octaves and lacunarity has been fixed to the values 7 and 2. The value for octaves was identified through testing.

The fBm implementation is as following:

```
        float pnoisenorm(vec2 point)
        {
            return snoise(point) * 0.5 + 0.5;
        }
        float fbm(vec2 point, float H)
        {
            float result = 0.0;
            float max = 0.0;
            for(int i=0;i<octaves;++i)
            {
                float dampen = pow(lacunarity, -H * float(i));
                result += pnoisenorm(point) * dampen;
                max += 1.0 * dampen;
                point *= lacunarity;
            }
            return  result / max;
        }
```

Listing 1: The fBm implementation

Most of it is standard fBm with a few changes. The `pnoisenorm` transform the output to be $[0; 1]$ in order to have the same range as the coordinates on the XY plane. The `fbm` method has an additional variable called `max` which keeps track of the highest possible output for the fbm function. The output of fbm grows large as H grows smaller since each sample is not dampened as much, eg. if $H \in [0 : 1]$ and *pnoisenorm* can produce a value between 0 and 1 then with 7 octaves we get that:

$$\sum_{n=0}^{n<7} 1 * 2^{-0} > \sum_{n=0}^{n<7} 1 * 2^{-1}$$

This is an issue when the value of H is configurable by the user, as the overall height of the terrain will grow as H is reduced. This causes issues with the camera placement as well as the assumption of the bounds of the terrain object. The `max` variable is used to normalize the shader output by forcing it to be in range [0 : 1] by calculating the theoretical max value for the current H. Using this approach, the output from fbm becomes more reliable.

Once calculated, the terrain sample is assigned to the Z value of the vertex coordinate after which it is transformed into clip space and assigned to gl_Position and lightPerspectivePosition. lightPerspectivePosition makes use of the same MVP matrix as used when generating the shadowmap in order to check if the resulting fragment is in visible light later in the fragment shader.

Finally, the normal is calculated as previously covered and passed onto the fragment shader.

An additional two noise samples are taken in the vertex shader: `heightNoise` and `textureNoise`. Those could technically by sampled in the fragment shader for better results, but that would require having the noise function and their depending uniforms in the fragment shader as well. The usage of those two samples will be covered in the fragment shader section.

The fragment shader applies shading to the terrain. First, the base color of the terrain is assigned based on the height of the generated terrain. The noise sample `heightNoise` from the vertex shader is used to add some variance to the threshold in order to prevent the transition to be completely uniform. The condition for the various types of terrain can be found in table 1.

Height(h)	$h \leq 0.4$	$h > 0.4$	$h > 0.66$
Type	Grass	Ground	Snow

Table 1: The terrain types and their heights

The base color were originally used as-is, but it looked very boring and plastic like. In order to add more detail another high-frequency noise sample is used to change the variance of the color slightly. The sample is `textureNoise` and is transformed to be between $[0.5; 1]$(except for ground which needed a even smaller span $[0.75; 1]$). The noise sample is multiplied to the base color which creates some variance in the texturing, such as dark patches of snow, rocky cliff-sides and a variety of grass patches. The result of added noise can be seen in 4.

Once the base color has been determined, it is time to add the shadows. First, the NDC(Normal Device Coordinates) are calculated for the fragment as seen from the light source's perspective in order to replicate the position the fragment would have had if rendered to the shadowmap. The area surrounding the fragment is then sampled using the percentage-closer technique in the shadowmap and the average of the samples are returned. The brute-force method is used to produce the result as it is simpler to understand and sufficient for this project.

We can then compare the NDC.z coordinate with the result from the shadowmap in order to determine if the fragment is in light or not. In the case that it is, a penalty denoting the light strength is set to 0.2 which will cause areas in shadow to be dimmer. Using the percentage-closer technique provided good results in how jagged the shadows look as shown in figure 5.

At last, we have the diffuse and ambient lighting. The ambient lighting is just a constant light which is added to the resulting color. The diffuse light is calculated in world space by taking the dot product between the light source and the normal estimated from the terrain. Because the two vectors are normalized, we get a result equivalent to the cosinus of the angle between them. This

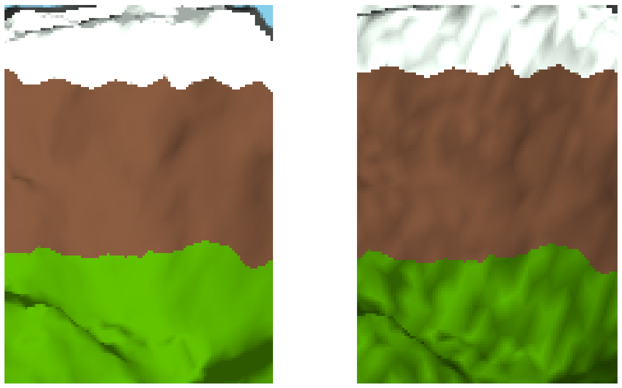

Figure 4: The terrain without(left) and with(right) noise

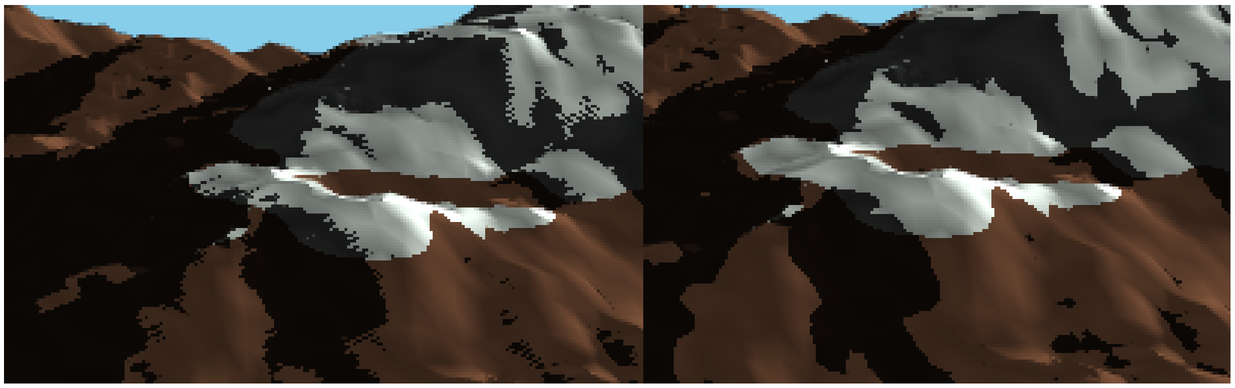

Figure 5: The terrain without(left) and with(right) percentage close filtering

causes the diffuse influence to be higher the as the angle between the light and normal gets smaller. Adding the diffuse lighting had a large impact on the visible depth and scale of the terrain as shown in figure 6.

The contribution from ambient and diffuse lighting does not add up to 1. This might not agree with the laws of physics, but having them like this creates a brighter scene which makes it easier to see the details of the terrain.

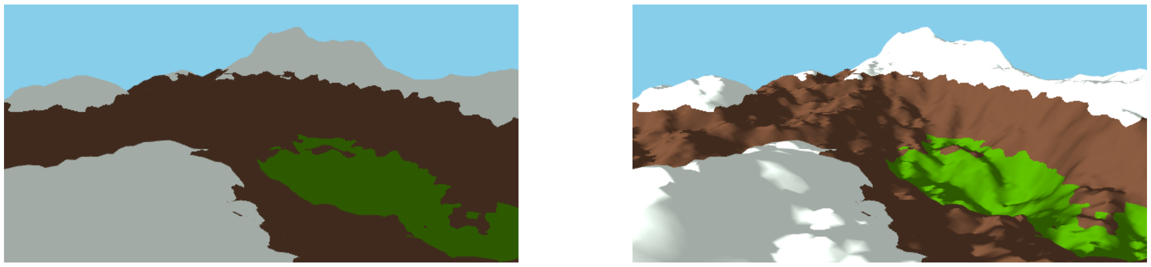

Figure 6: The terrain with only ambient lighting(left) and with ambient and diffuse(right). Self shadowing and texture noise has been removed.

Results

The final results is a scene with a self-shadowing landscape that has been generated in realtime using simplex noise. The webpage provides three sliders: *Noise* which changes the noise scale and creates landscapes of mountains with varying densities. *Smooth* which increases the H value of the fBm fractal, affecting the smoothness of the landscape and at last *Rotate* which rotates the camera around the perimeter of the landscape. With the default settings of the webpage, the noise function has put us right on the edge of a small valley surrounded by mountains. [!hbt]

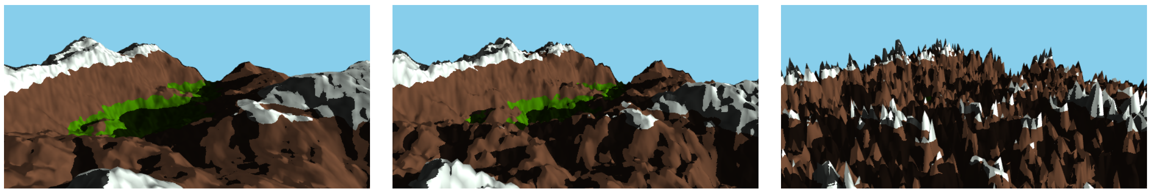

Figure 7: The terrain by lowering the smoothness. First is with full smoothness, 2nd is with slightly less and the terrain quickly goes towards noise from then.

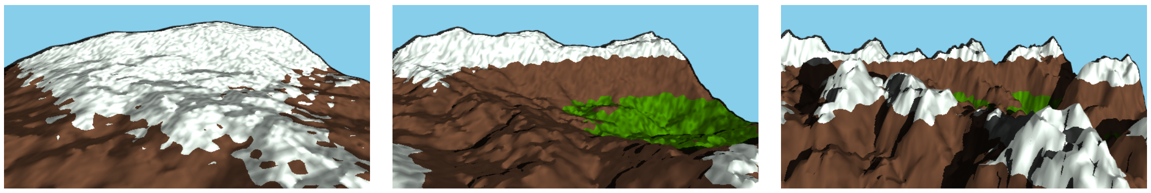

Figure 8: The terrain using with varying noise scaling. First image is just a large hill, 2nd a small mountain range and 3rd a lot of mountain peaks.

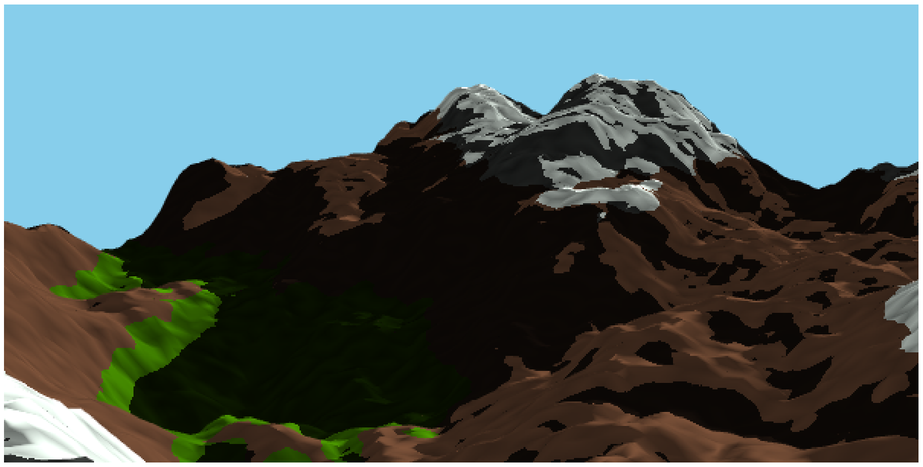

Figure 9: The shadow cast by a large mountain

Discussion

The terrain generation works with good results. The shadows are surprisingly well behaved considering how many small bumps on the terrain it have to work with. There are some artifacts with shadows if looking at them from the front. An example is shown in 10. This could be due to an imprecision in the shadowmapping which makes it believe that the top of the mountain should be in shadow. It could also be because of the Percentage-Closer filtering since there is a lot of shadows on the other side of the mountain, which could cause the mean to be more biased towards shadow. The shadows in the image are also at the points furthest from the light source and hence will be subject to the most magnification, especially since the shadowmap is rendered from a low angle.

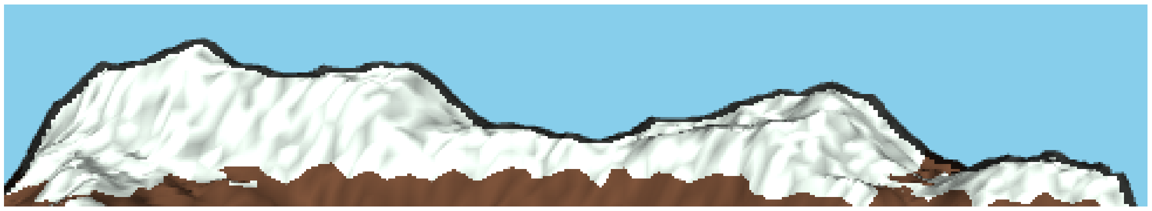

Figure 10: Shadow artifact along a mountain ridge

The opposite to overshadowing can be seen in places where the geometry is tall and narrow as shown in figure 11. This can again be explained with a small imprecision in the shadowmapping.

There is still room for improvement in certain areas. Some of them have been mentioned where applicable throughout this rapport. A most interesting feature would be to add the functionality of moving through the terrain and have more generate with time. This would be able to showcase the real benefit of procedural terrain, a huge amount of content generated. Other areas are mostly cosmetic, such as making the transition from grass to ground more natural and making the icecaps more reflective. A final area that was looked into during development was depth-of-fog but was scrapped due to time constraints.

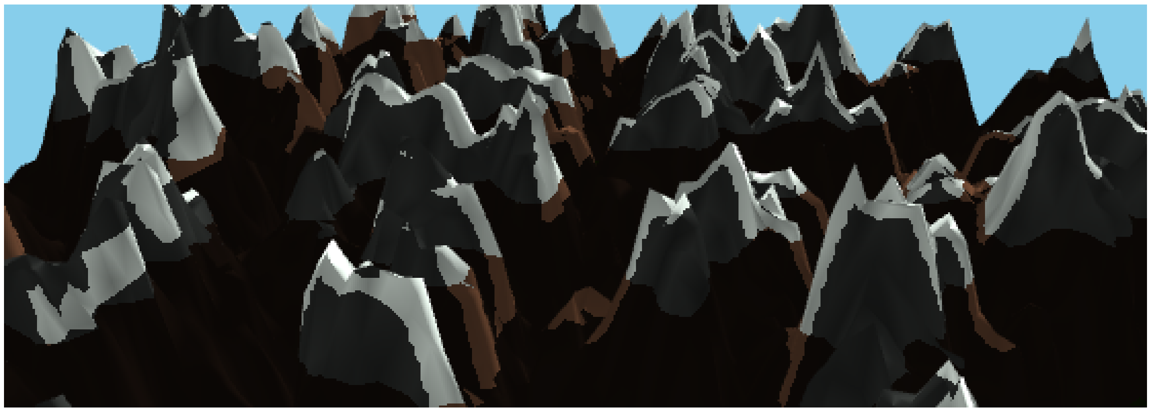

Figure 11: Shadow artifact along a mountain peak

References

[1] Edward Angel and Dave Shreiner. Interactive Computer Graphics. 7th ed. Pearson. ISBN: 978-1-292-01934-5.

[2] David S. Ebert et al. Texturing and Modeling: A Procedural Approach. 3rd. 2003. Chap. 14, 16. ISBN: 978-1-55860-848-1.

[3] "Finite difference". URL: `https://en.wikipedia.org/wiki/Finite_difference` (visited on 12/12/2019).

[4] NVidia. GPU Gems. Shadow Map Antialiasing. Vol. 1. Chap. 11. URL: `https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch11.html`.

[5] Inigo Quilez. "terrain raymarching". URL: `https://www.iquilezles.org/www/articles/terrainmarching/terrainmarching.htm` (visited on 12/12/2019).