# Programmatically generated landscapes

Darshan, Sagar

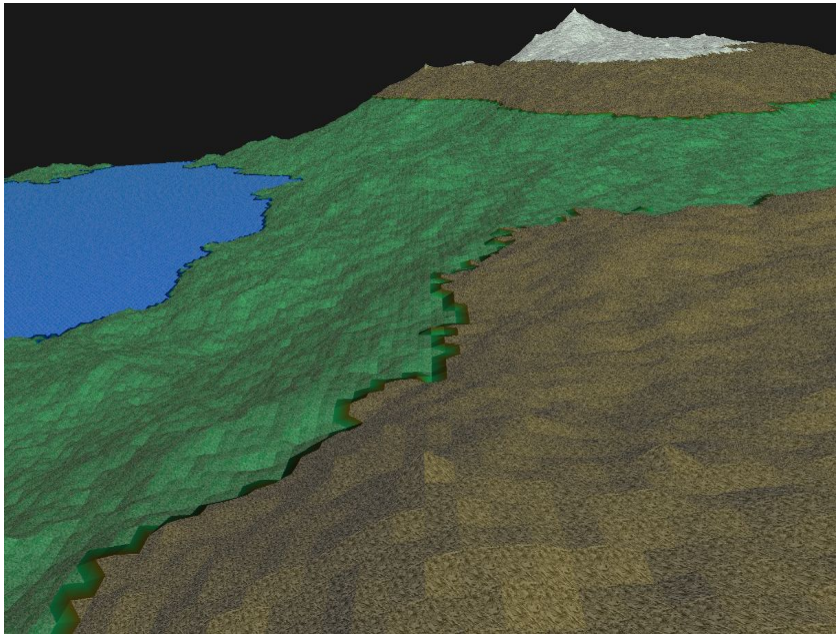
# Problem Statement

Programmatically generate a landscape which looks like this image.

- Terrain
  - Water
  - Land
  - Mountains
- Clouds
- Grass

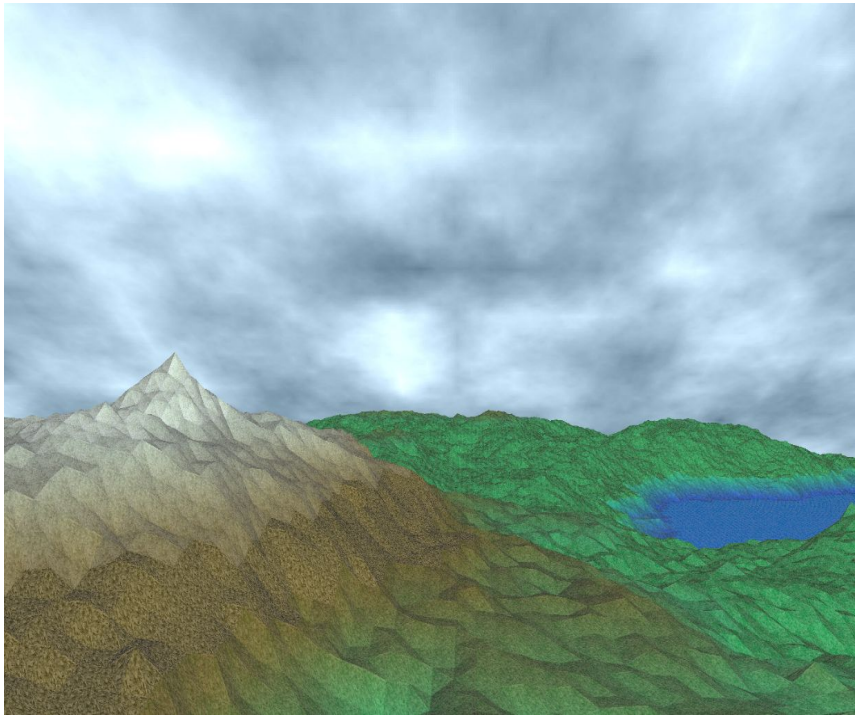
# Terrain generation: Height maps

- Generated height map for entire terrain using diamond and square algorithm.
- Terrain is divided into ocean, beach, grass, green mountains, brown mountains and ice based on height of point.
- Different textures are used for each region



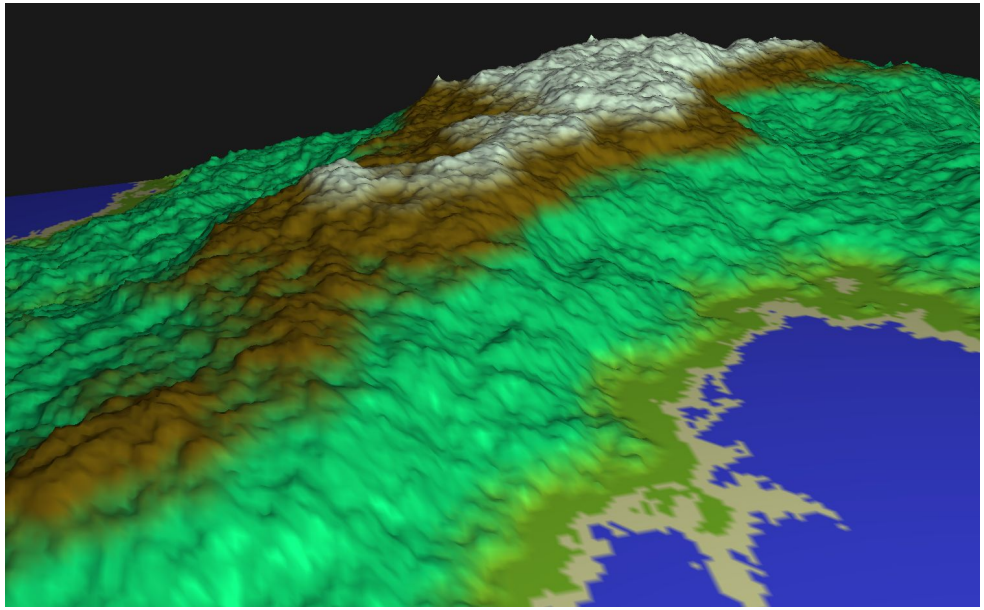
# Terrain generation: Interpolation

- Mixed texture with solid colors for better effect.
- Interpolated textures/color between regions (ice and sand) so as to not have discontinuity in the scene.
- Filled background with clouds generated on 2D plane.



# Terrain generation: Smoothing

- Smoothing: Computed one normal for each vertex by taking average of normals of all surrounding faces. **Removes triangulation!**
- Laplacian smoothing to avoid any abrupt spikes and depression in terrain.

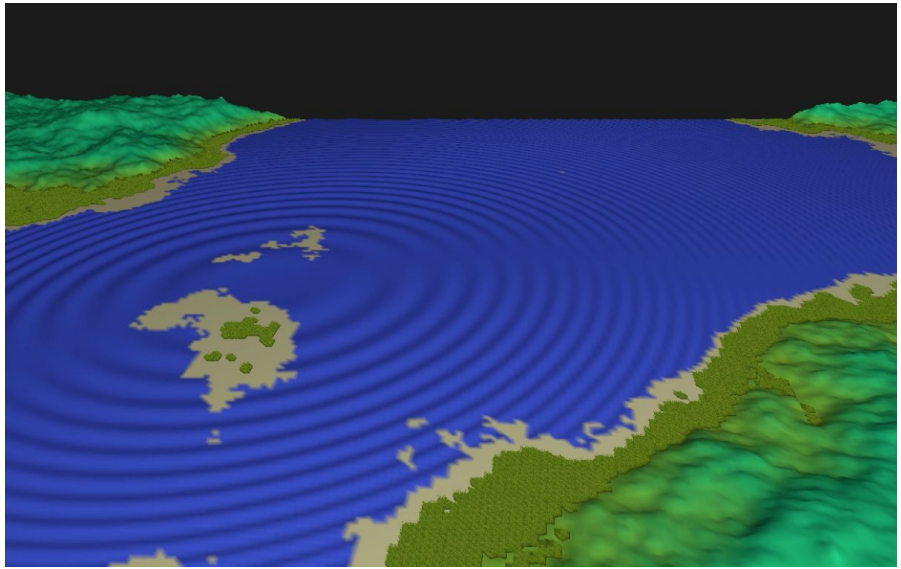


# Water waves

- Used current_time and xy position  to calculate delta z.

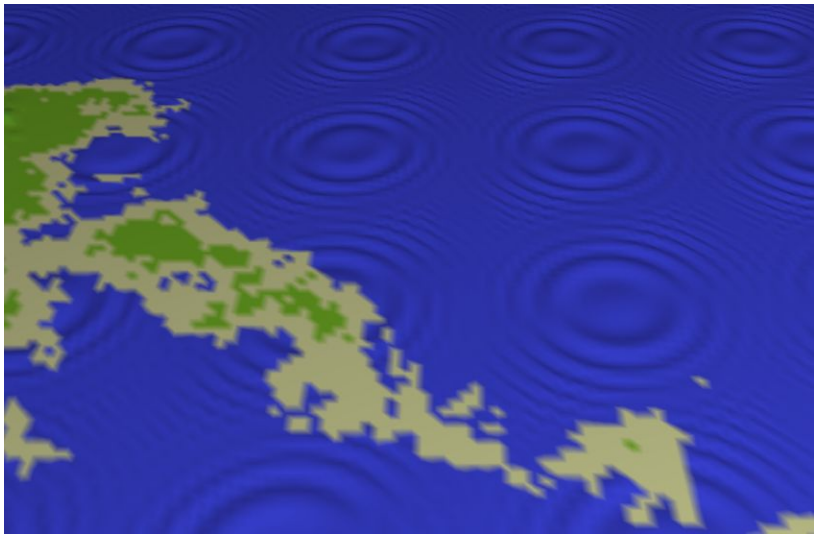
$$\Delta Z = sin(a(x - x_0)^2 + b(y - y_0)^2 + current\_time)$$

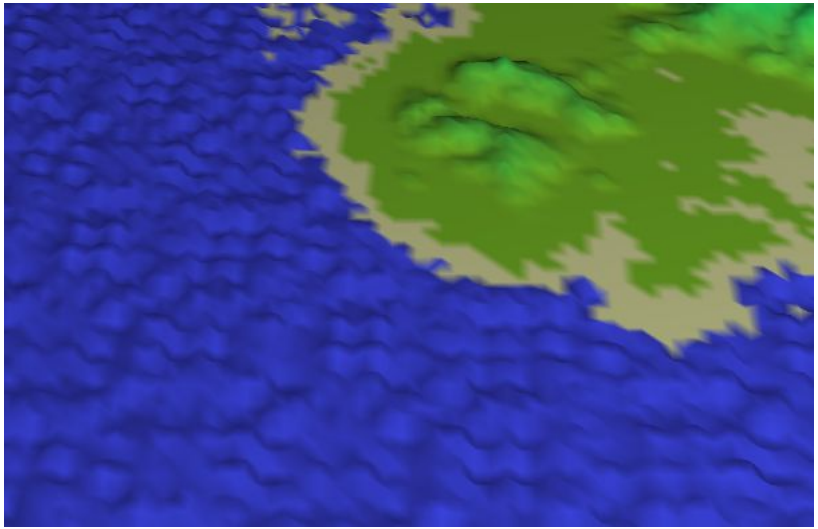
- Delta z of all surrounding points is known, hence normals of all surrounding faces is computed. And finally normal of vertex is computed.



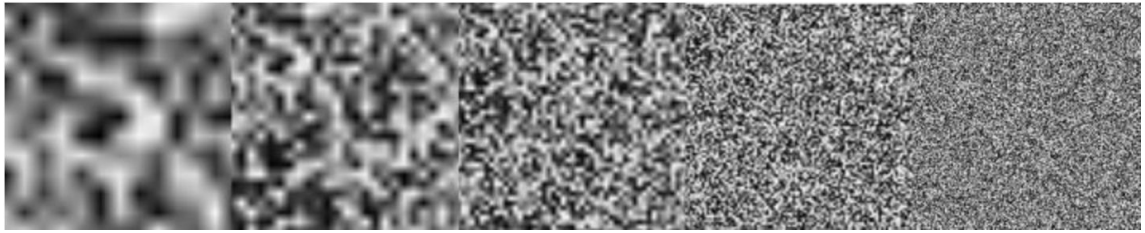
# Water waves

- Ability to position epicenter of single/multiple waves wherever desired in scene.
- Parameters to control speed of wave, roughness of sea etc





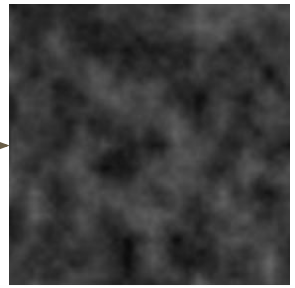
# Clouds: Perlin Noise



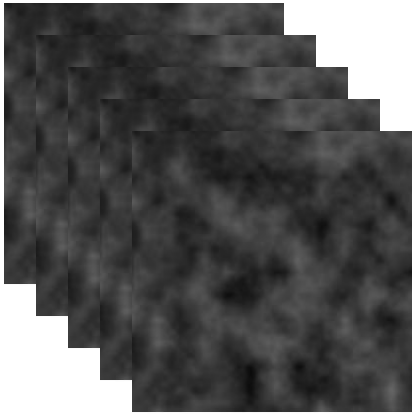
16x zoom | 8x zoom | 4x zoom | 2x zoom | 1x zoom

16x zoom 16x weight | 8x zoom 8x weight | 4x zoom 4x weight | 2x zoom 2x weight | 1x zoom 1x weight

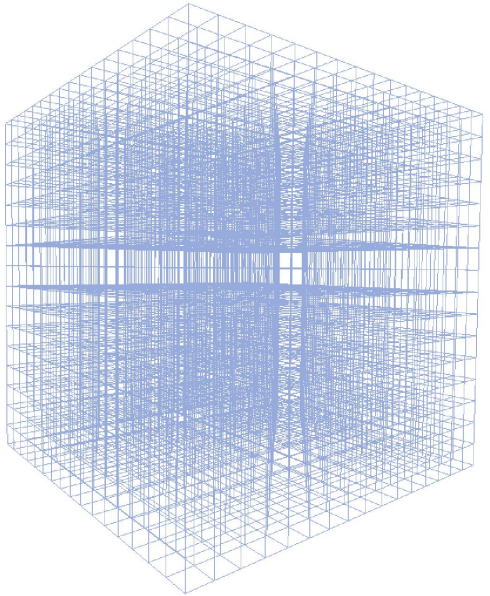
average



# Clouds: Perlin Noise



texture



3D noise = n layers of 2D noise

Don’t look at it from the wrong side
Extension: render cubes instead of quads

# Clouds: 3D Perlin Noise

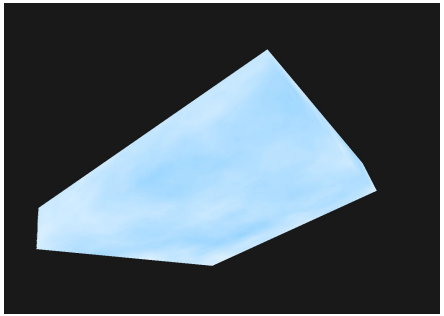


Image 1: No alpha channel

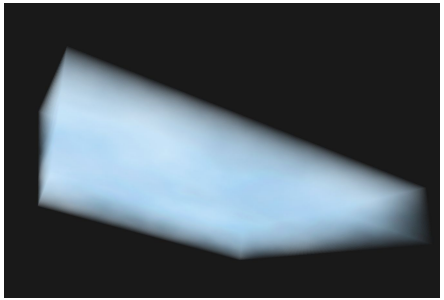


Image 2: alpha = 0.1 * perlin

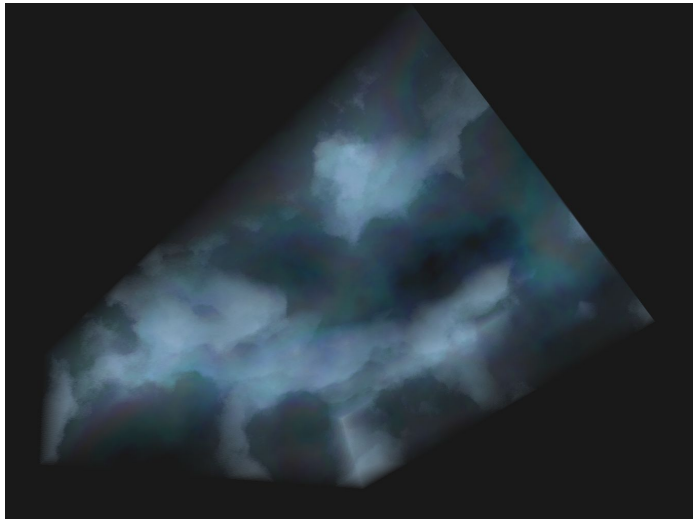


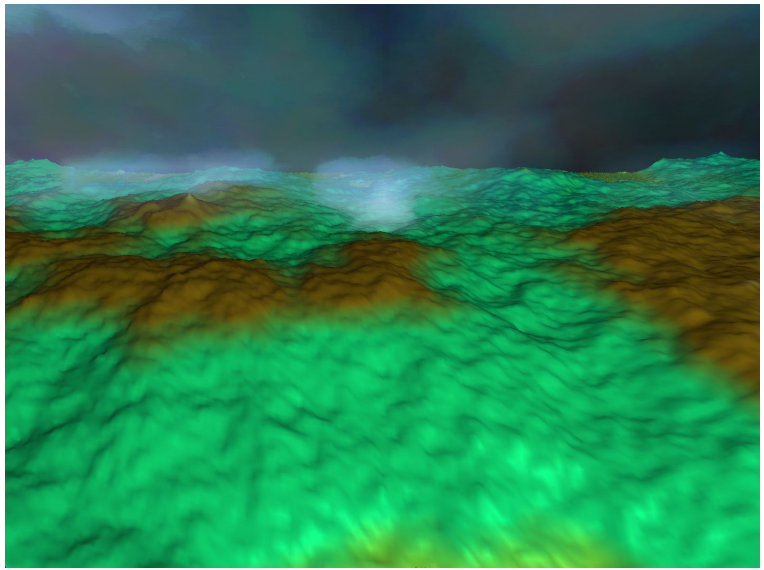
Image 3: Cloud-like shape

```
if (noise < 0.8) {
  noise = noise^2;
} else if (noise < 0.6) {
  noise = noise^4;
} else if (noise < 0.4) {
  noise = noise^8;
} else if (noise < 0.2) {
  noise = noise^16;
}
```

# Clouds: Transparency

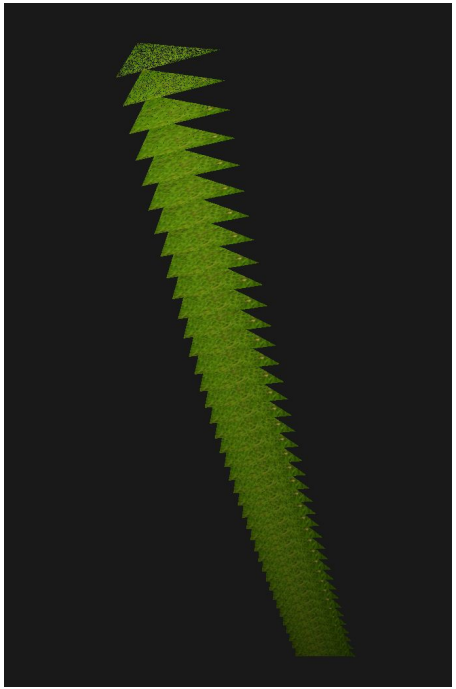
Render clouds, then render terrain

Render terrain, then render clouds



# Grass: Layers of triangles

- Add N triangles on top of the triangle for each triangle
- Texture it with an image which looks like grass
- **Problem: triangles are big**



# Grass: Strands

Create another texture for the alpha channel.

- Use a parameter *grass_density* (how thick is the grass)
- Set the initial alpha to 0
- Sample points in the texture area and set alpha to 1

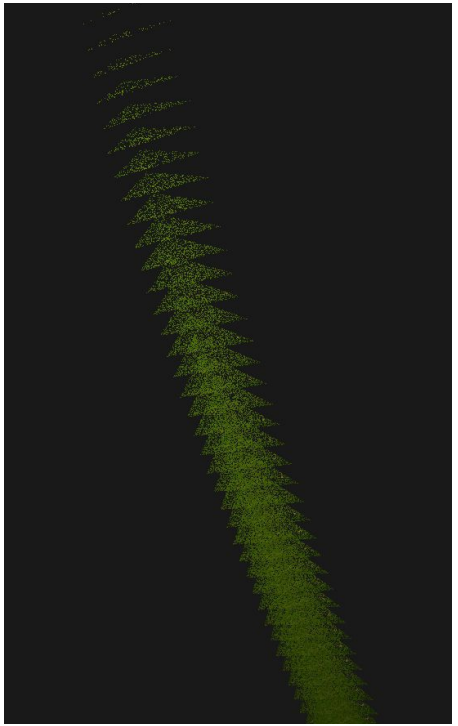
Cast a fake shadow, points in the lower layers appear darker.

```
fakeShadow = 0.6 + 0.4 * layer;
texColor = texture(...grass...) * fakeShadow;
```

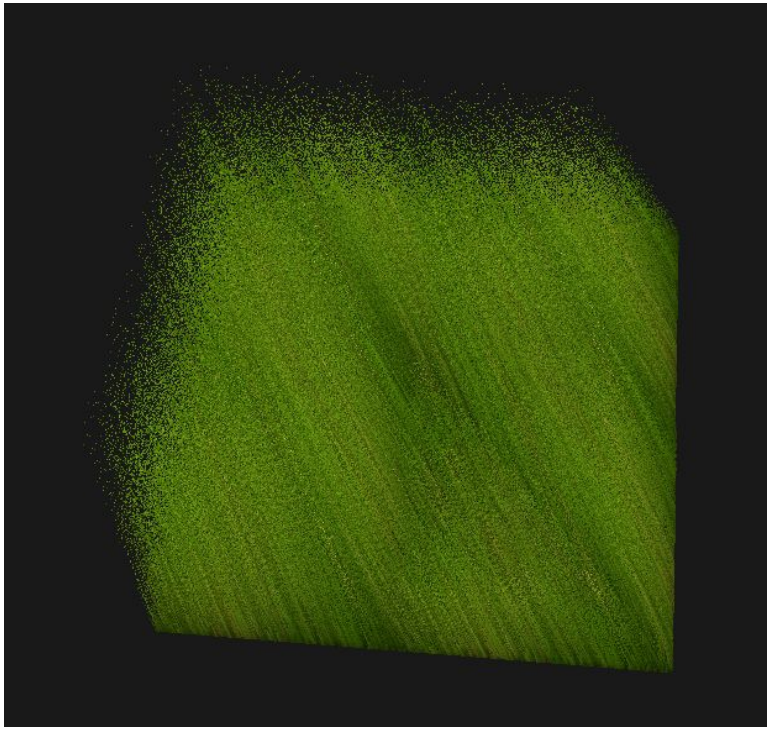
A grass strand becomes thinner at a higher layer.

- For each strand, compute the max_layer it should be seen at
- Set a lower alpha in the fragment shader for higher layers based on this number

```
maxLayer = pow(i / strandsPerLayer / layers, 0.7);
```



# Grass: Putting it together



# Grass: Animation

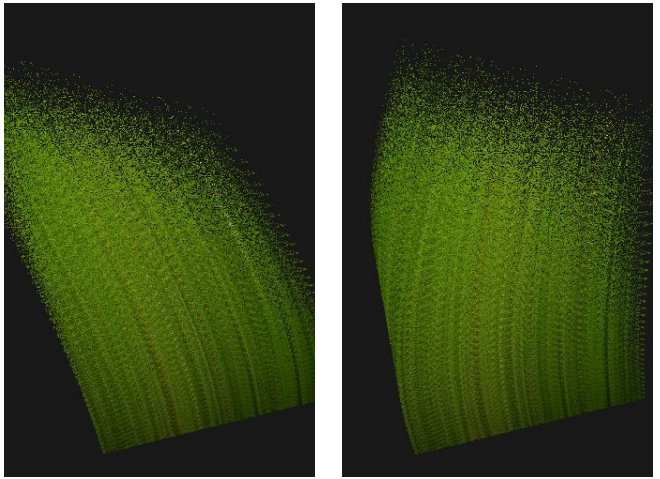
Points in the higher layer are displaced more than the roots.

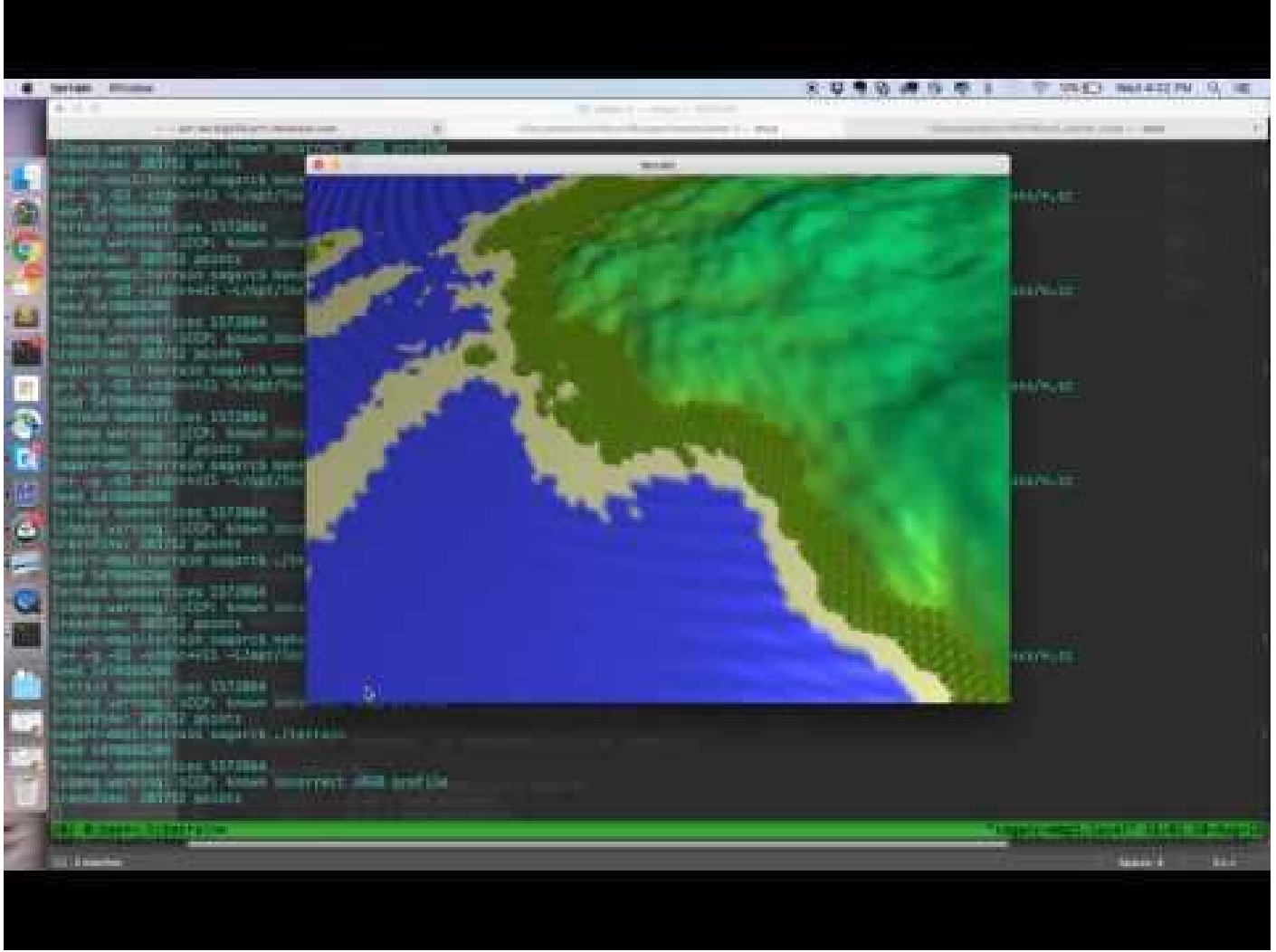
Compute a number *displacement* in each iteration:

```
glm::vec3 gravity(0.0f, -0.8f, 0.0f);
glm::vec3 force(sin(glfwGetTime()) * 0.5f, 0.0f, 0.0f);
glm::vec3 disp = gravity + force;
```

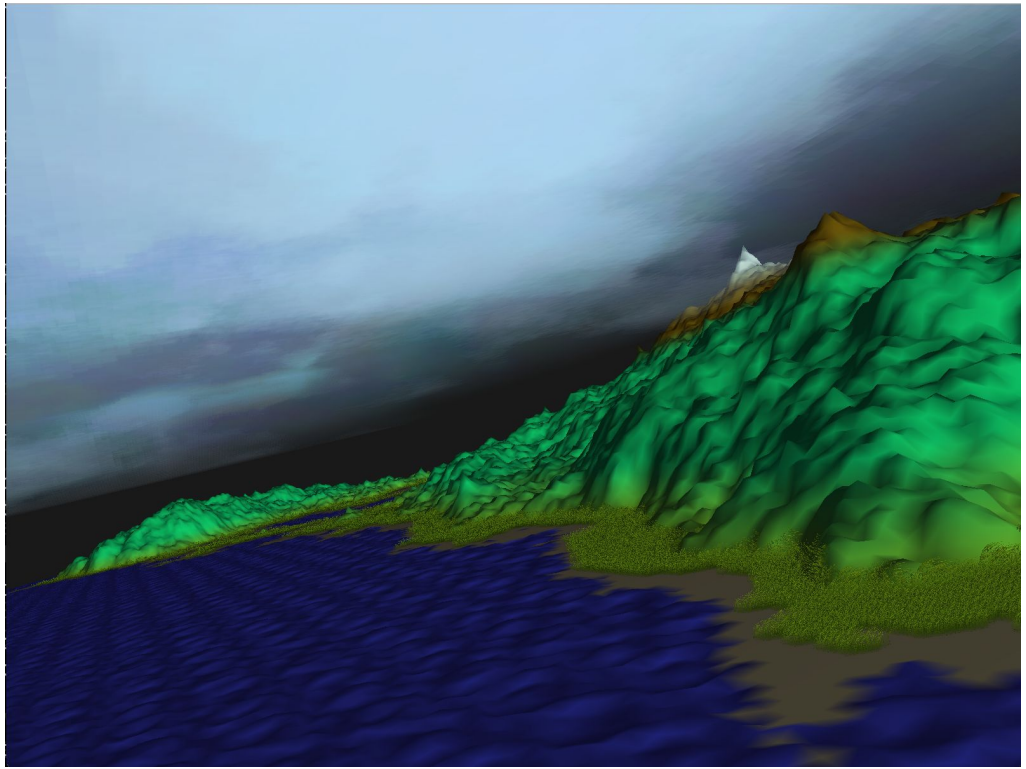
Displace higher layers more than the lower layers

```
vec3 layerDisplacement = pow(layer, 3.0) * displacement;
vec4 newPos = vec4(pos + layerDisplacement, 1.0);
gl_Position = projection * modelView * newPos;
```





# Final Results



# References

http://www.catalinzima.com/xna/tutorials/fur-rendering/

http://lodev.org/cgtutor/randomnoise.html

https://github.com/rgruener/Terrain_Generator/

http://www.gameprogrammer.com/fractal.html

# Thank you