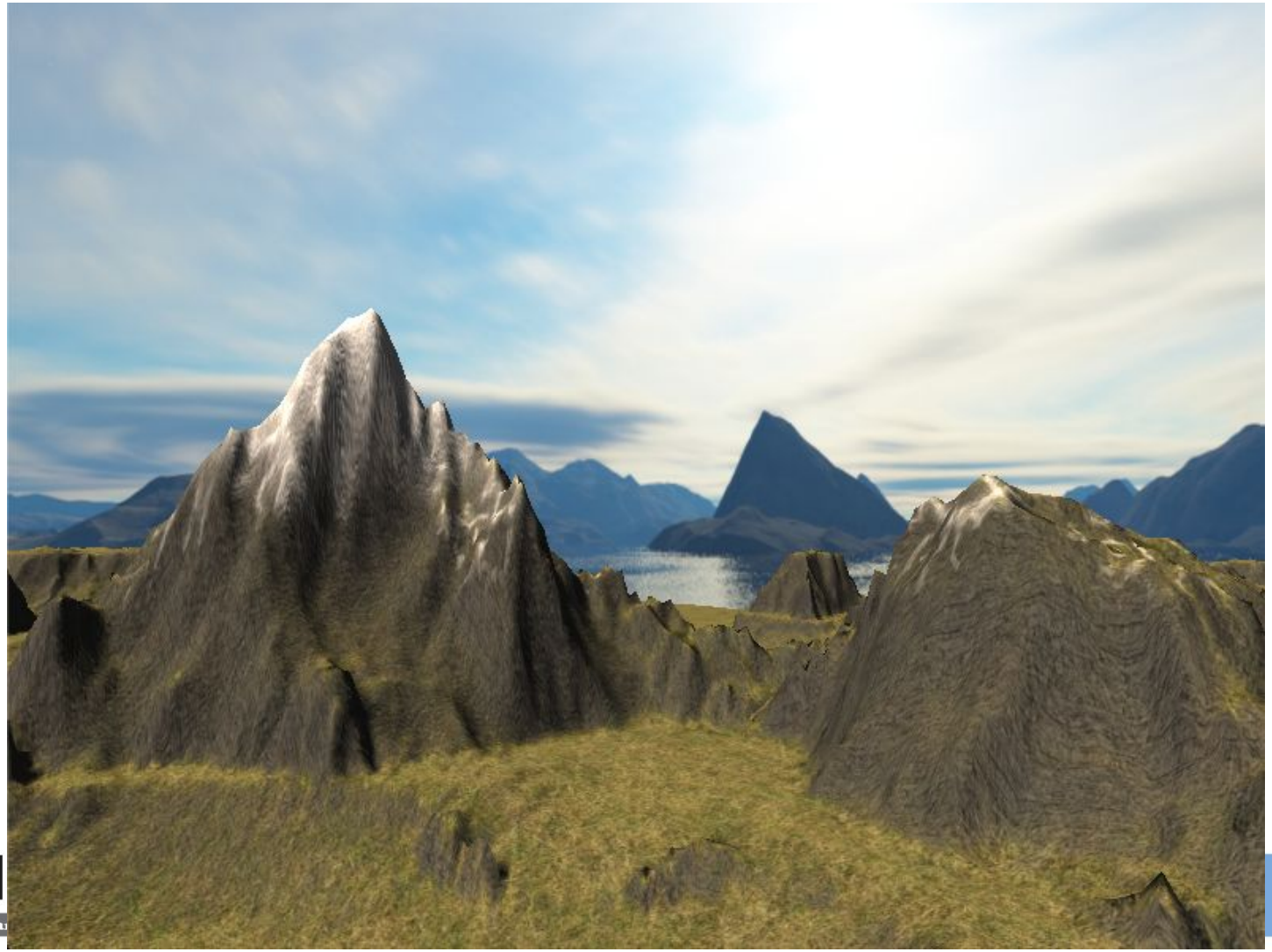
# MVD: Advanced Graphics 1

## 19 - Terrain

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

# Terrain



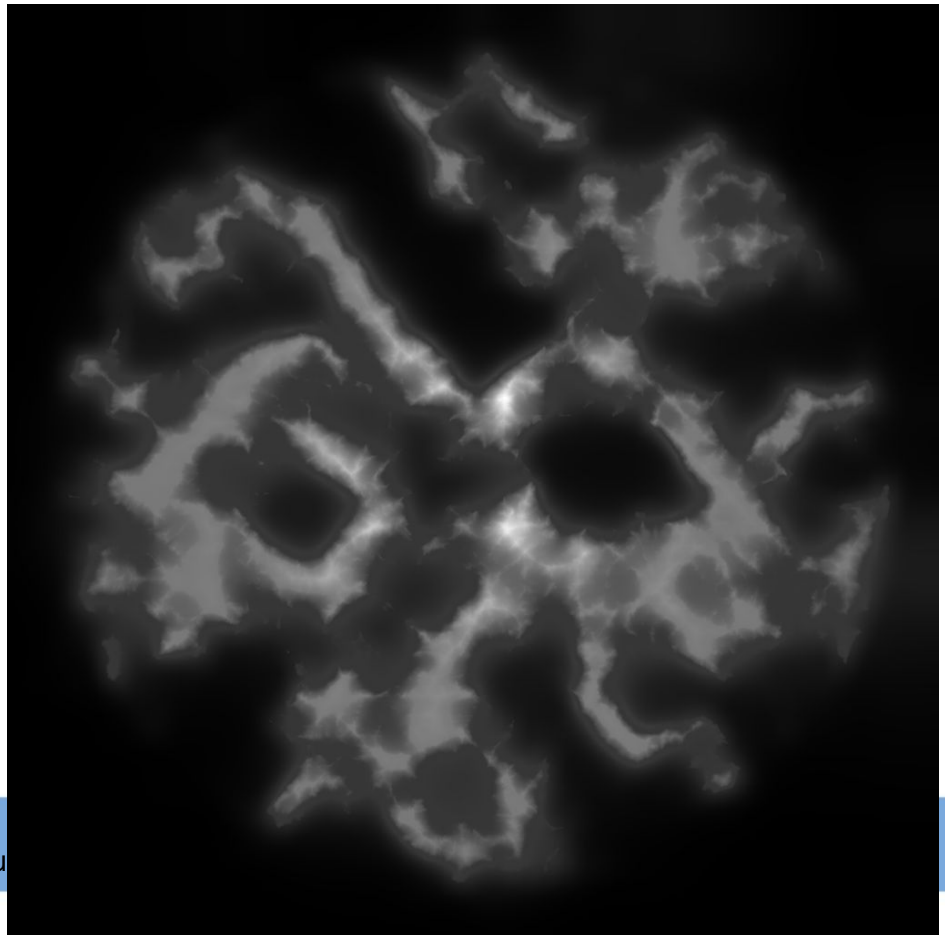
# The height map

The height map is a greyscale image which we sample to modify the height of the terrain

Sample the height map at each vertex.

Change vertex position.y based on sample

*Could you do this in the shader?*



# Creating a grid

Need:

- resolution (how many grid points per axis)
- step (distance between grid point)
- max height (the y-coord of highest point)
- height map pixel data

Making vertices:

```
foreach axis
    for each gridpoint in resolution
        add step, make grid point
```

# Loading data

New Geometry functions to create geometry

Modification of Parsers:: load texture to store pointer to texture data

```
static GLuint parseTexture(std::string filename,
                           ImageData* image_data = nullptr,
                           bool keep_data = false);
```

ImageData struct

add noise map and multiple diffuse maps to material

**if you add a new map to material must init = -1!!!**

# Terrain material

Note scaled uvs and noise_map

```
//noise map data - must be cleaned up
ImageData noise_image_data;
float terrain_height = 20.0f;

int mat_terrain_index = graphics_system_.createMaterial();
Material& mat_terrain = graphics_system_.getMaterial(mat_terrain_index);
mat_terrain.shader_id = terrain_shader->program;
mat_terrain.specular = lm::vec3(0,0,0);
mat_terrain.diffuse_map = Parsers::parseTexture("data/assets/terrain/grass01.tga");
mat_terrain.diffuse_map_2 = Parsers::parseTexture("data/assets/terrain/cliffs.tga");
mat_terrain.normal_map = Parsers::parseTexture("data/assets/terrain/grass01_n.tga");
//read texture, pass optional variables to get pointer to pixel data
mat_terrain.noise_map = Parsers::parseTexture("data/assets/terrain/heightmap1.tga",
                                              &noise_image_data,
                                              true );

mat_terrain.height = terrain_height;
mat_terrain.uv_scale = lm::vec2(100,100);
```

# Terrain material

Note freeing memory!

```
//terrain
//create terrain geometry - this function is a wrapper for Geometry::createTerrain
int terrain_geometry = graphics_system_.createTerrainGeometry(  500,
                                                                0.4f,
                                                                terrain_height,
                                                                noise_image_data);

//delete noise_image data other we have a memory leak
delete noise_image_data.data;
```

# Terrain entity

```
//terrain
int terrain_entity = ECS.createEntity("Terrain");
Mesh& terrain_mesh = ECS.createComponentForEntity<Mesh>(terrain_entity);
terrain_mesh.geometry = terrain_geometry;
terrain_mesh.material = mat_blue_check_index;
terrain_mesh.render_mode = RenderModeForward;
```

# Multiresolution tiling

Simple tiling of grass texture

```
//sample grass
vec2 s_uv = v_uv * u_uv_scale;
vec3 grass_full_res = texture(u_diffuse_map, s_uv).xyz;
mat_diffuse *= grass_full_res;
```

see tiling really badly

# GLSL mix function

```
mix(colorA, colorB, ratio)
```

ratio of 0.0 = 100% colorA

ratio of 1.0 = 100& colorB

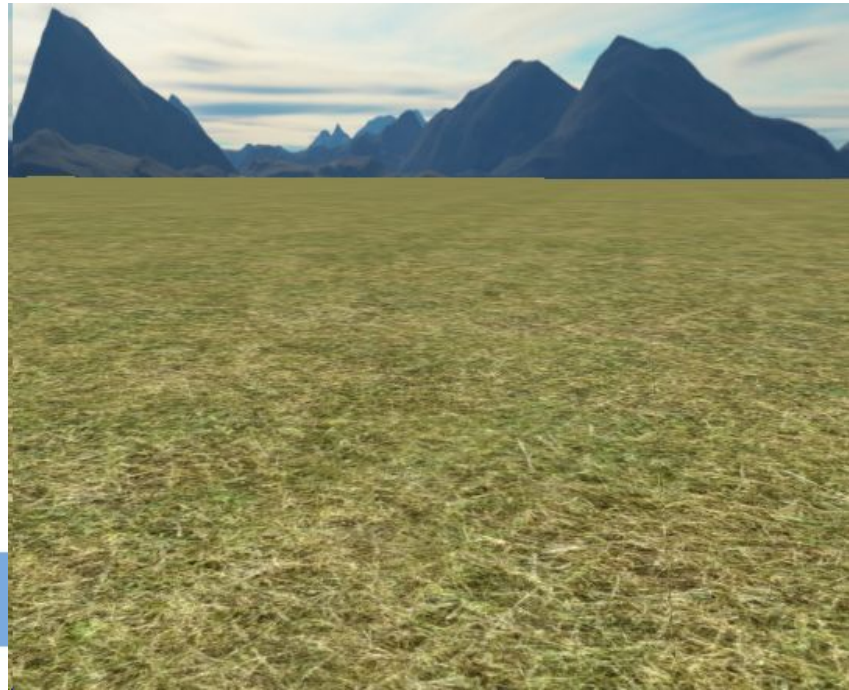
# Multiresolution tiling

Multiply scaled uvs (separately) by 0.4 and 0.1.

Sample texture three times

Mix result:

```
//mix the three resolutions to get a result
mat_diffuse *= mix(
                    mix (grass_full_res, grass_med_res, 0.5),
                    grass_low_res,
                    0.2
                );
```





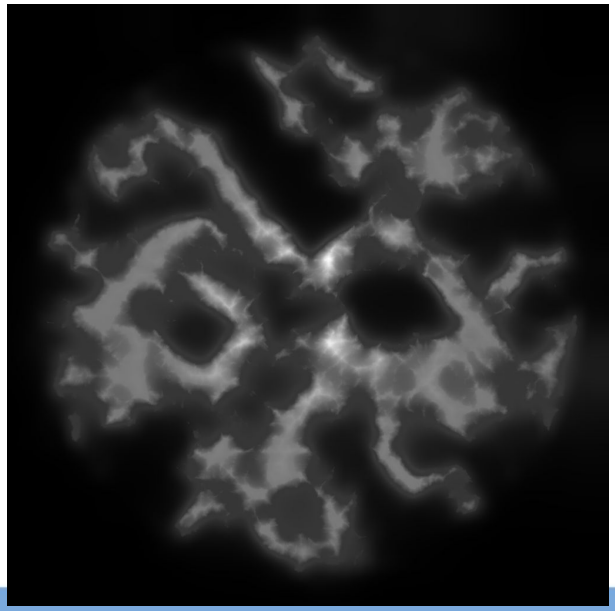
MVD
Alun Evans – alunthomas.evans@salle.url.edu

# Sampling height map data

Our image data struct has a getPixel function.

Receives pixel coords, and a int[3] array pointer

*Why int[3] and not float[3]?*



# Pixel data

Pixel is data unsigned int 0 -> 255

Get pixel fills array with pixel data

```
bool getPixel(int x, int y, int pixel[3]) {

    if (x > width || y > height) return false;

    int pixel_location = width * bytes_pp * y + x * bytes_pp;
    pixel[0] = data[pixel_location];
    pixel[1] = data[pixel_location + 1];
    pixel[2] = data[pixel_location + 2];
    return true;
}
```

# Converting pixel data to height

Need to have max height of terrain

Need to normalize pixel data to 0->1

so

```
height = (data / 255) * max_terrain_height
```

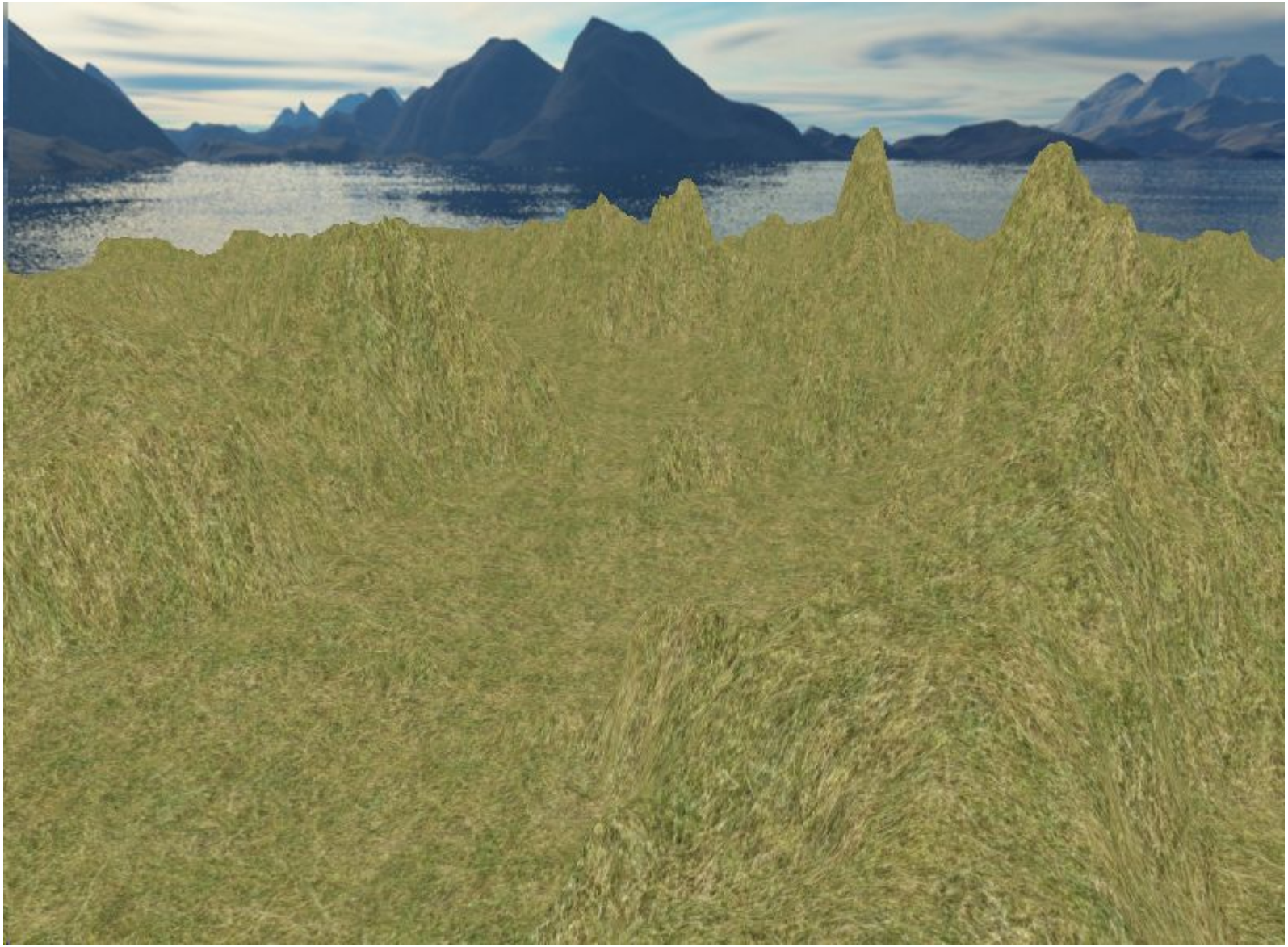
# Which pixel to sample

For each grid point in mesh, we must find pixel coord (x, y) for image map:

e.g. for x

```
normalized_grid_point_x = (vx / resolution)

x_pixel = normalized_grid_point_x * width_image
```



# Normals

Normals are calculated using partial derivative of the heightmap, in x and y

normal at pixel P = (*dx*, 2.0, *dy*)

where $dx = P_{x-1} - P_{x+1}$ and $dx = P_{y-1} - P_{y+1}$

# For this to work we need to make boundary checks before getting pixel data

```
lm::vec3 Geometry::calculateTerrainNormal(ImageData& height_map, int x, int y){
    //boundary check
    if (x == 0) x = 1; if (y == 0) y = 1;
    if (x == height_map.width) x--; if (y == height_map.height) y--;

    //obtain partial derivatives in both dimensions, using data points either side
    int hl[3]; int hr[3]; int hd[3]; int hu[3];
    height_map.getPixel(x - 1, y, hl);
    height_map.getPixel(x + 1, y, hr);
    height_map.getPixel(x, y - 1, hu);
    height_map.getPixel(x, y + 1, hd);

    lm::vec3 n((float)(hl[0] - hr[0]), 2.0, (float)(hd[0] - hu[0]));

    return n;
}
```

# Uploading multiple textures

Can upload multiple diffuse maps if you want!

```
//texture uniforms - diffuse
if (mat.diffuse_map != -1){
    shader_->setUniform(U_USE_DIFFUSE_MAP, 1);
    shader_->setTexture(U_DIFFUSE_MAP, mat.diffuse_map, 8);
} else shader_->setUniform(U_USE_DIFFUSE_MAP, 0);

//add extra diffuse maps
if (mat.diffuse_map_2 != -1) {
    shader_->setUniform(U_USE_DIFFUSE_MAP_2, 1);
    shader_->setTexture(U_DIFFUSE_MAP_2, mat.diffuse_map_2, 9);
}
else shader_->setUniform(U_USE_DIFFUSE_MAP_2, 0);
if (mat.diffuse_map_3 != -1) {
    shader_->setUniform(U_USE_DIFFUSE_MAP_3, 1);
    shader_->setTexture(U_DIFFUSE_MAP_3, mat.diffuse_map_3, 10);
}
else shader_->setUniform(U_USE_DIFFUSE_MAP_3, 0);
```

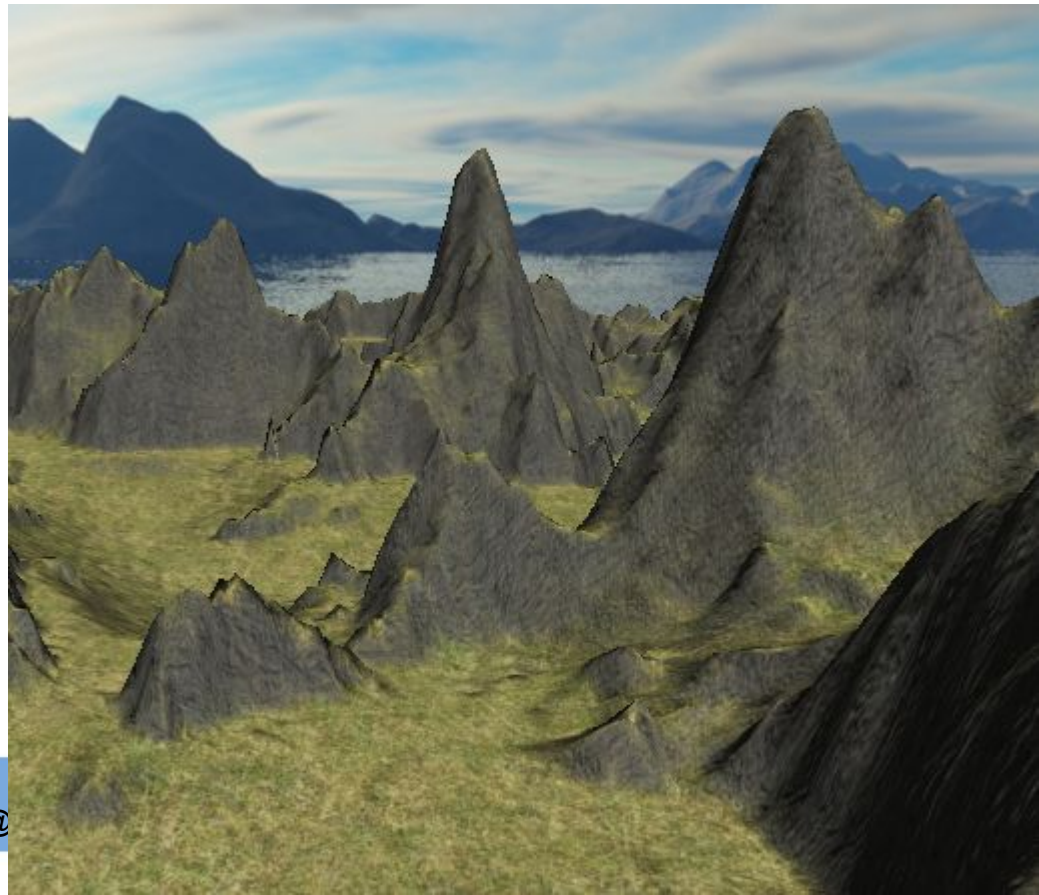
# Mix in cliffs texture

Load cliffs texture as diffuse texture 2

mix based on normal.y

Can scale N.y

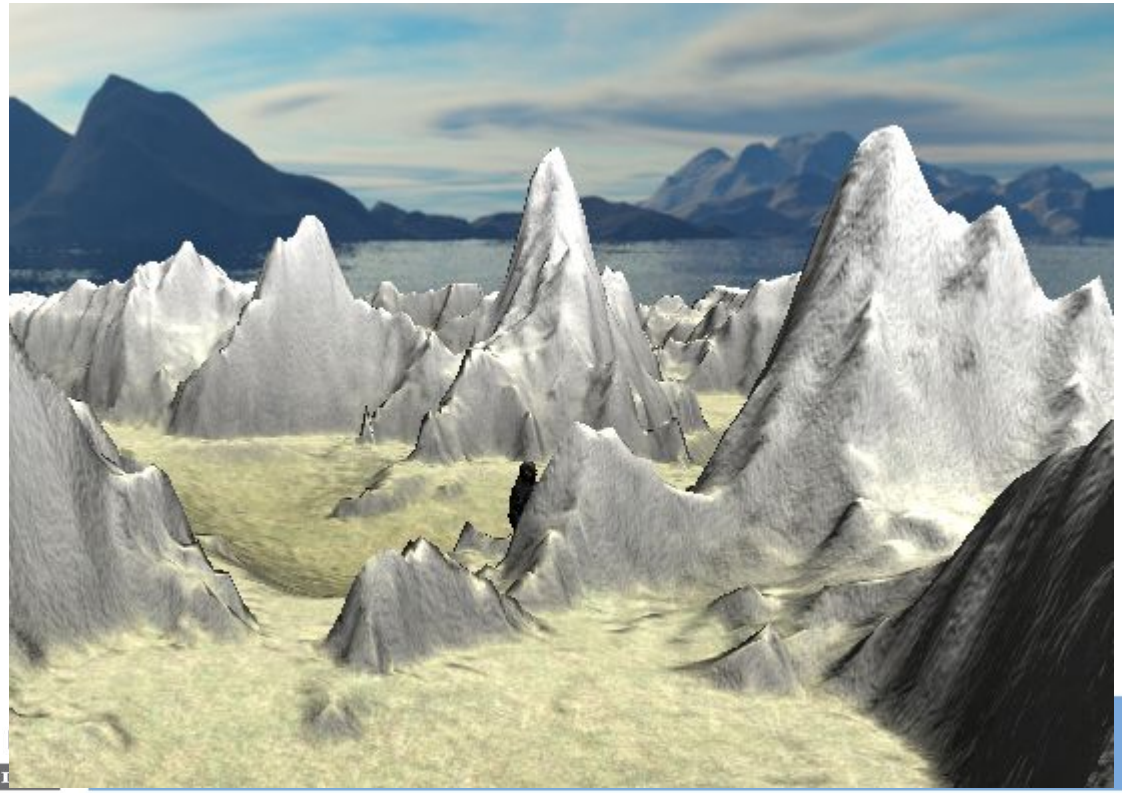
to decide quantity



# Mix in ‘snow’

Snow is just the color white (could be a texture if you want)

snow_mix = vertex_position.y / maximum_height_of_terrain

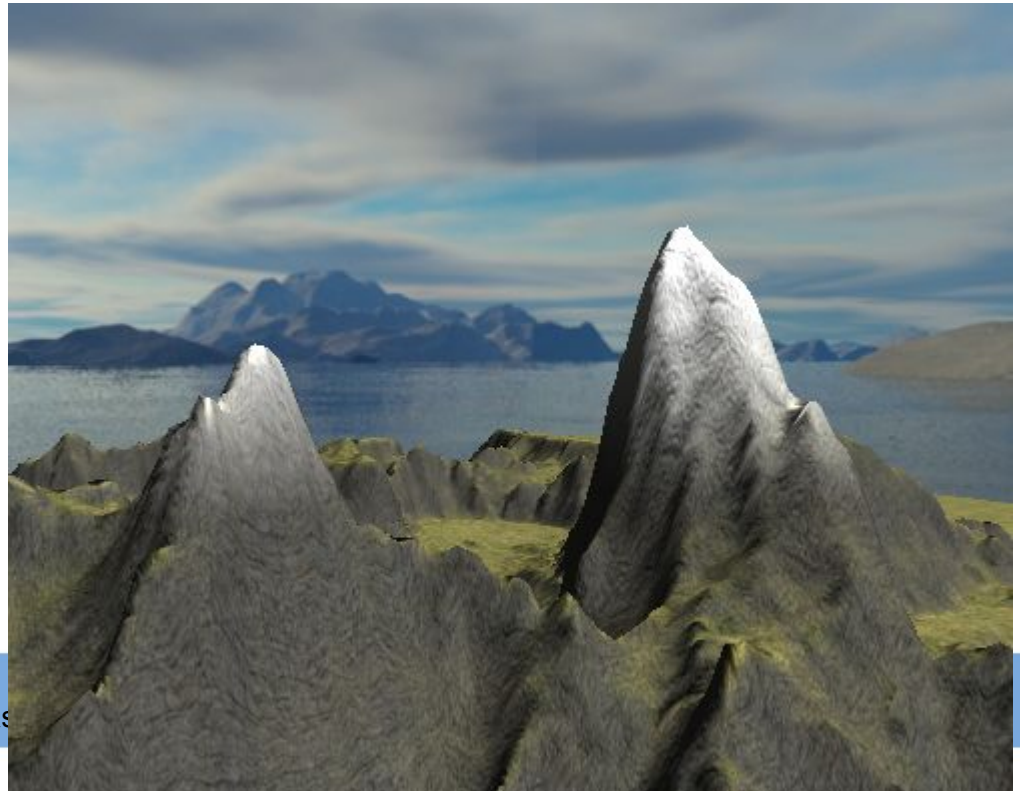


# Non-linear snow

Multiply mix by a nonlinear function so that snow is only on highest peaks

snowmix = snowmix * *x* - *y*

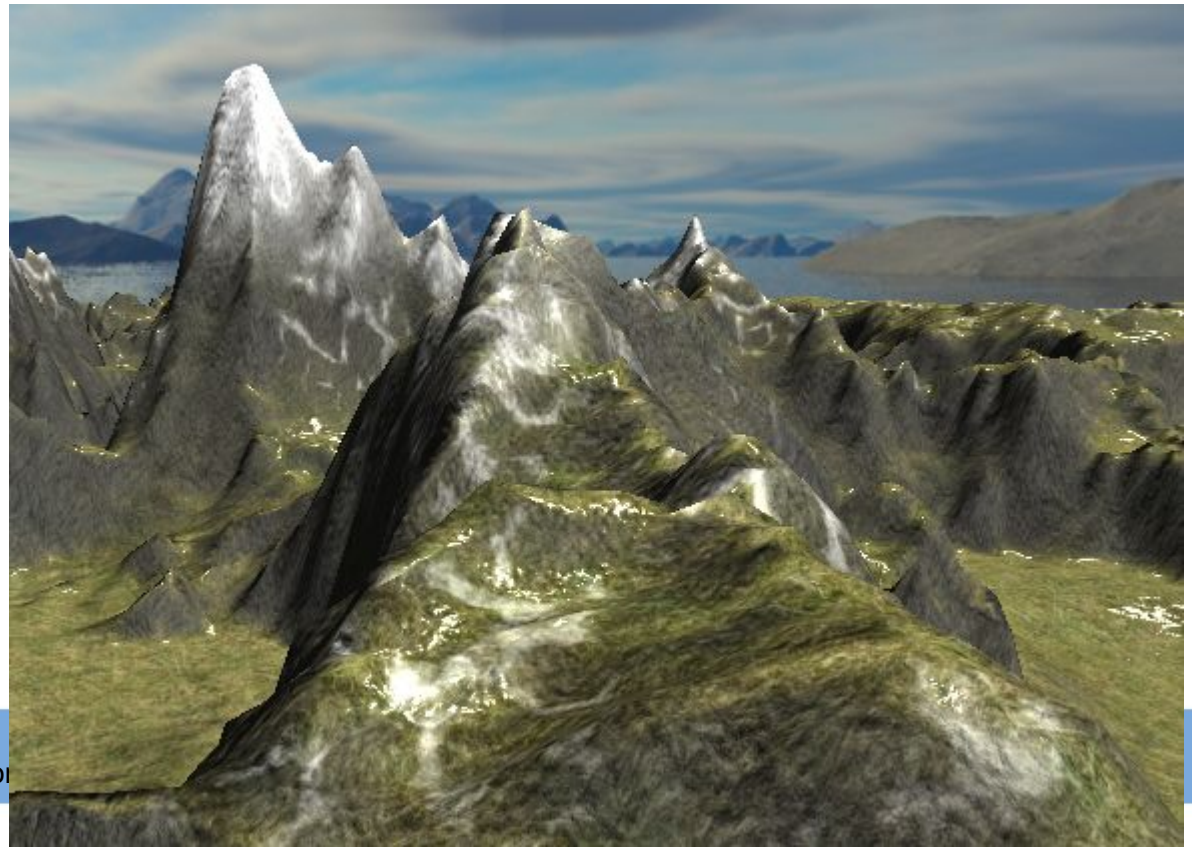
play with x and y to see levels



# Mixing with noise map

Modify snow by sampling a noise map

This is the same noise texture used to create terrain



# More ideas

Paths - use values of a greyscale texture to mix a path/bricks texture

Water - we we will look at this later but for now you could put a simple plane with a reflection shader. Use fresnel's law to modify reflection amount based on camera angle with normal