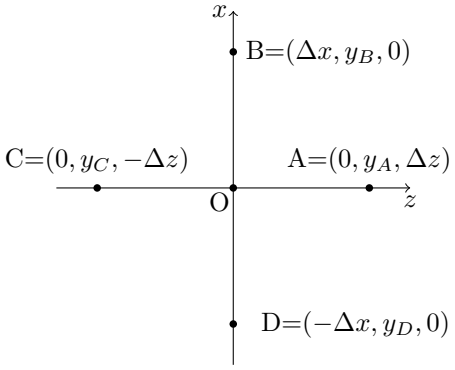
Normal calculation from heightmap in vertex shader

The calculation of normals requires us to be very strict about our axis orientation. We will always use the y-axis as the upward direction, and we use a right-handed axis-system.

We want to calculate the normal vector for certain vertex. For our calculations, let this vertex be located at height y_0 on the y-axis. Every vertex on the terrain (except for the edges) will have four direct neighbouring vertices which we will use to calculate the normal vector. We label these four neighbours A, B, C, D and this gives us the following figure: (Note: the y-axis is pointing towards you, out of the paper.)



We know that the normal vector of the triangle through OAB is given by the crossproduct of the vectors $\vec{OA}$ and $\vec{OB}$ (in that order!). Just like that, the normals of the three other triangles are given by $\vec{OB} \times \vec{OC}$, $\vec{OC} \times \vec{OD}$ and $\vec{OD} \times \vec{OA}$. The normal of the vertex in the origin is then given by the sum of these four normals. The coordinates of the points A, B, C, D are given in the picture. Here Δx and Δz are the distance between two vertices. We find:

$$\vec{OA} \times \vec{OB} = \begin{pmatrix} 0 \\ y_A - y_0 \\ \Delta z \end{pmatrix} \times \begin{pmatrix} \Delta x \\ y_B - y_0 \\ 0 \end{pmatrix} = \begin{pmatrix} -\Delta z(y_B - y_0) \\ \Delta x \cdot \Delta z \\ -\Delta x(y_A - y_0) \end{pmatrix}$$

and

$$\begin{aligned} \vec{OB} \times \vec{OC} &= \begin{pmatrix} -\Delta z(y_B - y_0) \\ \Delta x \cdot \Delta z \\ \Delta x(y_C - y_0) \end{pmatrix} \\ \vec{OC} \times \vec{OD} &= \begin{pmatrix} \Delta z(y_D - y_0) \\ \Delta x \cdot \Delta z \\ \Delta x(y_C - y_0) \end{pmatrix} \\ \vec{OD} \times \vec{OA} &= \begin{pmatrix} \Delta z(y_D - y_0) \\ \Delta x \cdot \Delta z \\ -\Delta x(y_A - y_0) \end{pmatrix} \end{aligned}$$

Now the normal of the center vertex is given by the sum of these four normals. We find:

$$\text{Normal} \propto \begin{pmatrix} 2\Delta z(y_D - y_B) \\ 4\Delta x \cdot \Delta z \\ 2\Delta x(y_C - y_A) \end{pmatrix} \propto \begin{pmatrix} (y_D - y_B)/\Delta x \\ 2 \\ (y_C - y_A)/\Delta z \end{pmatrix}$$

Note that this normal is independent of the height of the center vertex. Also, it still needs to be normalised.

Coding the vertex shader

The vertex shader works as follows: it receives ONLY texture coordinates for each vertex: a 2-dimensional vector where each component is in the range $[0, 1]$. We first map the x and z coordinate to $[-\frac{1}{2}, \frac{1}{2}]$ so that the center of the terrain is the origin, and it has width and length 1.

```
vec4 pos;
pos.x = textureCoordinate.s - 0.5;
pos.y = texture2D(heightMap, textureCoordinate).r;
pos.z = textureCoordinate.t - 0.5;
pos.w = 1.0;
vec4 position = mMatrix * pos;
```

We multiply by `mMatrix`. This move matrix allows scaling, translation and rotation of the terrain. This matrix also gives us the values of Δx and Δz. We also need a value Δy since the y value in the texture is in the range $[0, 1]$ whereas the value that we used in the calculations above are in world space coordinates. If we note the height values that we obtain from the texture by y_A, y_B, y_C, y_D then we will have to multiply these by Δy and this would give us

$$\text{Normal} \propto \begin{pmatrix} \Delta y(y_D - y_B)/\Delta x \\ 2 \\ \Delta y(y_C - y_A)/\Delta z \end{pmatrix} \propto \begin{pmatrix} (y_D - y_B)/\Delta x \\ 2/\Delta y \\ (y_C - y_A)/\Delta z \end{pmatrix}$$

Now we obtain the heights of the points A, B, C, D and this is done by a simple texture lookup:

```
float A = texture2D(heightMap, textureCoordinate + vec2(0.0,textureDelta)).r;
float B = texture2D(heightMap, textureCoordinate + vec2(textureDelta,0.0)).r;
float C = texture2D(heightMap, textureCoordinate + vec2(0.0,-textureDelta)).r;
float D = texture2D(heightMap, textureCoordinate + vec2(-textureDelta,0.0)).r;
```

The value `textureDelta` is passed to the shader as a uniform float, and is given by `1.0/textureWidth`. Now we have to find the values for Δx, Δy and Δz and then we are done. We could get these from `mMatrix` however this would be an expensive calculation that is not neccesary, since these values would be the same for each vertex. Therefore we will do this on the CPU and pass the information to the vertex shader through a new uniform variable: `uniform vec3 terrainDimensions;`. This will contain the terrain dimensions. That means that we can not use `mMatrix` for scaling, but only for rotations and translations. Then we find:

```
float deltaX = terrainDimensions.x * textureDelta;
float deltaY = terrainDimensions.y;
float deltaZ = terrainDimensions.z * textureDelta;
```

Note that the distance between two neighbouring points on the texture is `textureDelta` and if we multiply this by `terrainDimensions.x` (or z) then we find the distance for the points in world space. We can calculate the final normal:

```
normal = vec3((D-B)/deltaX, 2.0/deltaY, (C-A)/deltaZ);
normalOut = normalize((mMatrix * vec4(normal,0.0)).xyz);
```

We still multiply by `mMatrix` because this will allow us to rotate the terrain if needed. Note that we use `vec4(normal,0.0)` so that we will not get the translations stored in `mMatrix`.

Now there is one more thing we have to fix: since we took scaling out of `mMatrix` we also lost the scaling of the position vector which was multiplied by `mMatrix`. This means that in the first block of shader code we have to change `pos.x` textureCoordinate.s - 0.5;= to `pos.x` terrainDimensions.x*(textureCoordinate.s - 0.5);=. This is it, we now have a fully functional vertex shader that will calculate the normal vectors from the heightmap.

Full code

This is the code used in the final vertex shader. The local variables `deltaX` (and y, z) have been written directly into the normal.

```
//Shader input
attribute vec2 textureCoordinate;
uniform sampler2D heightMap;
uniform float textureDelta;
uniform vec3 terrainDimensions;
uniform mat4 mMatrix;

//Shader code
vec4 pos;
pos.x = terrainDimensions.x*(textureCoordinate.s-0.5);
pos.y = terrainDimensions.y*(texture2D(heightMap, textureCoordinate).r);
pos.z = terrainDimensions.z*(textureCoordinate.t-0.5);
pos.w = 1.0;
vec4 position = mMatrix * pos;

float A = texture2D(heightMap, textureCoordinate + vec2(0.0,textureDelta)).r;
float B = texture2D(heightMap, textureCoordinate + vec2(textureDelta,0.0)).r;
float C = texture2D(heightMap, textureCoordinate + vec2(0.0,-textureDelta)).r;
float D = texture2D(heightMap, textureCoordinate + vec2(-textureDelta,0.0)).r;

vec4 normal = vec4( (D-B)/terrainDimensions.x, 2.0/terrainDimensions.y, (C-A)/terrainDimensions.z , 0.0 );
normalVarying = normalize( (mMatrix*normal).xyz );
```

Note that the real shader also contains some code that passes the light direction to the fragment shader, but I left this out because this code is the same for all our shaders and has nothing to do with this part of calculating terrain vertex normals.