Introduction to 3D Game Programming with DirectX 9.0c: A Shader Approach

FAQ

1. The terrain code takes a long time to load for heightmaps 513×513 and higher. How can I speed it up?

   The `D3DXComputeNormals` function is what takes the most time. The `D3DXComputeNormals` is quite generic and needs to find which triangles share a vertex for the normal averaging. But our terrain topology has a definite structure, and we can compute the normals much faster ourselves by applying a simple finite difference scheme. Replacing `D3DXComputeNormals` with the code below should shave off several seconds for 513×513 sized heightmaps and below, but it might not be enough for larger heightmaps like 1025×1025 and above. In these cases, you may want to precompute the normals and store than on disk so they just need to be read and load time, not calculated.

   We approximate the partial derivatives using central difference:

$$\begin{aligned}\frac{\partial h}{\partial x}\left(x_j, z_i\right) &= \frac{h\left(x_j + \Delta x, z_i\right) - h\left(x_j - \Delta x, z_i\right)}{2\Delta x} = \frac{h\left(x_{j+1}, z_i\right) - h\left(x_{j-1}, z_i\right)}{2\Delta x} \\ \frac{\partial h}{\partial z}\left(x_j, z_i\right) &= \frac{-\left[h\left(x_j, z_i + \Delta z\right) - h\left(x_j, z_i - \Delta z\right)\right]}{2\Delta z} = \frac{h\left(x_j, z_{i-1}\right) - h\left(x_j, z_{i+1}\right)}{2\Delta z}\end{aligned}.$$

   (Recall that based on our grid layout, as i increases, z_i decreases (i.e., it moves along negative *z*-axis), hence the -1 factor in $\partial h / \partial z$ since we want the partial derivative in the positive *z*-axis direction.)

   Now, the two tangent vectors at the point $\left(x_j, h\left(x_j, z_i\right), z_i\right)$ are given by:

$$\begin{aligned}\vec{T}_x &= \left(1, \frac{\partial h}{\partial x}\left(x_j, z_i\right), 0\right) \\ \vec{T}_z &= \left(0, \frac{\partial h}{\partial z}\left(x_j, z_i\right), 1\right)\end{aligned}.$$

   And the normal at this point is thus:

$$\vec{n} = \frac{\vec{T}_z \times \vec{T}_x}{\left\|\vec{T}_z \times \vec{T}_x\right\|}.$$

We compute the normals like this over the interior points. For the boundary points, we just set the normal to $(0,1,0)$. If necessary, a better approximation for the boundary points would be to use either a forward difference approximation or a ghost-point method.

Finally, the code is as follows:

```
//=================================================================
// Compute normals with finite difference over interior nodes.
float invTwoDX = 1.0f / (2.0f*mDX);
float invTwoDZ = 1.0f / (2.0f*mDZ);
for(DWORD i = 1; i < mVertRows-1; ++i)
{
	for(DWORD j = 1; j < mVertCols-1; ++j)
	{
		float t = mHeightmap(i-1,j);
		float b = mHeightmap(i+1,j);
		float l = mHeightmap(i,j-1);
		float r = mHeightmap(i,j+1);
		D3DXVECTOR3 tanZ(0.0f, (t-b)*invTwoDZ, 1.0f);
		D3DXVECTOR3 tanX(1.0f, (r-l)*invTwoDX, 0.0f);

		D3DXVECTOR3 n;
		D3DXVec3Cross(&n, &tanZ, &tanX);
		D3DXVec3Normalize(&n, &n);

		v[i*mVertCols+j].normal = n;
	}
}
```