SPG Project

Description

This application uses DirectX 11 to render a terrain generated by the marching cubes algorithm. The terrain is textured using tri-planar texturing and pixel shader displacement. Shadows are using variance shadow mapping to soften the their edges.

Inputs

Movement	
W, A, S, D	Movement in 3D space
Arrow Keys	Rotation in 3D space
Utility	
ESC	Close application
Z	Wireframe-mode
;	Cycle samples (MSAA)
.	Cycle quality levels
M	Confirm samples and quality levels
T	Regenerate terrain
R	Start/stop rotation of terrain
Space	Spawn particle system
Pixel Shader Displacement	
I	Decrease initial steps (PSD raycasting)
O	Increase initial steps (PSD raycasting)
K	Decrease refinement steps
L	Increase refinement steps
J	Decrease displacement depth
U	Increase displacement depth

Marching Cube

3D Texture procedurally generated density values. (+ noise)
The Marching Cubes algorithm uses a 3D texture of procedurally generated density values. Then vertices are generated around these density values by stepping in fixed sized steps over the texture and looking up specific cases of vertices to output. The vertices are interpolated between the density values so the final terrain looks just as intended.

In this application the resulting vertices are read from a geometry output shader into a new vertex buffer to cache the resulting geometry and not generate them every renderpass.

The terrain is then textured using “Tri-Planar texturing” where three textures are projected onto the surface and blended using the surface normals.

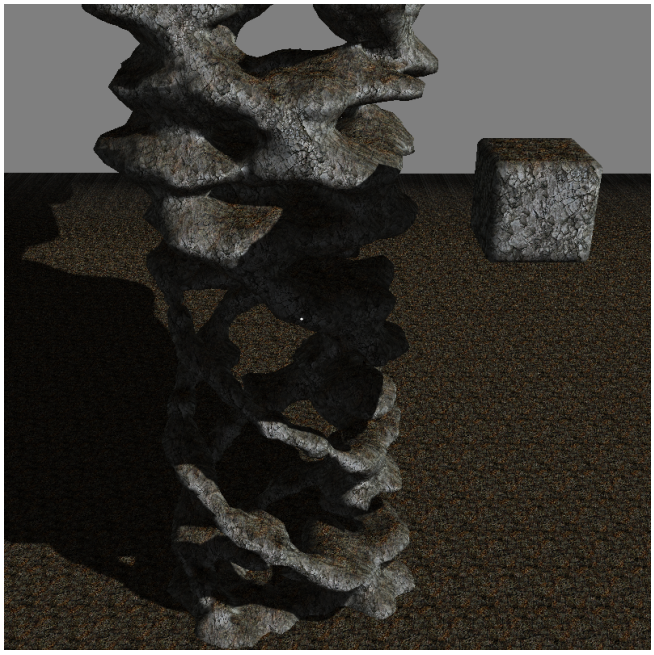


Figure 1: Generated Terrain

Pixel Shader Displacement Mapping

Pixel Shader Displacement is used to fake depth on a surface by querying depth values from a heightmap/displacement map in regular steps. To get the needed resolution initial steps and after the first hit refinement steps are used.

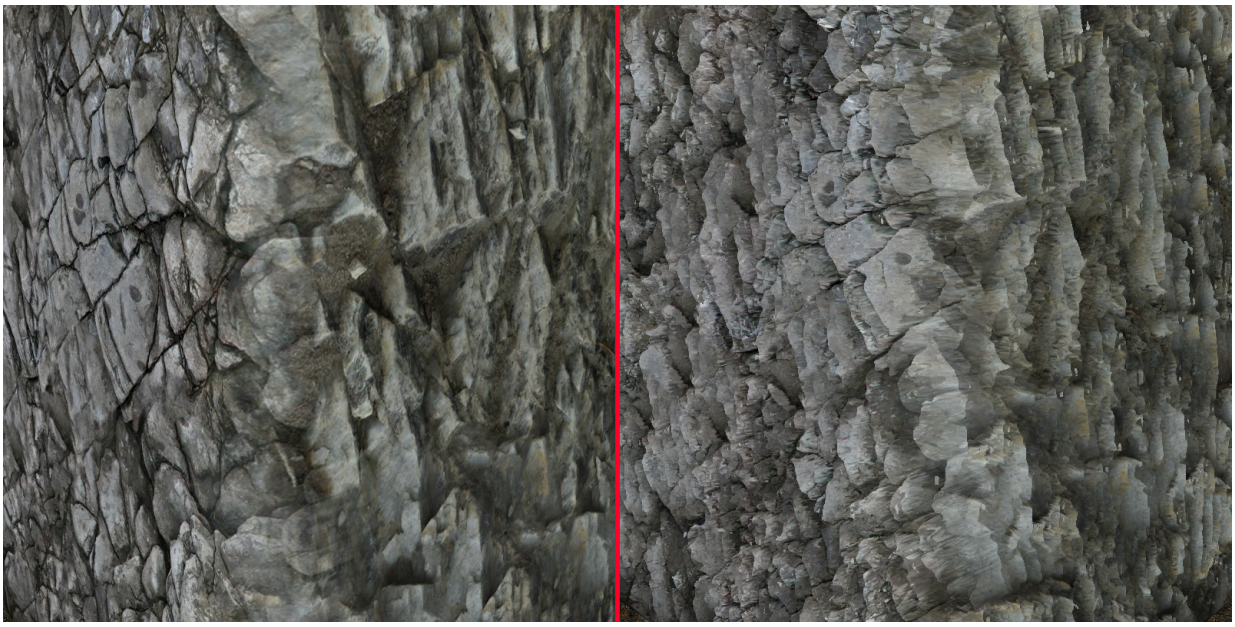
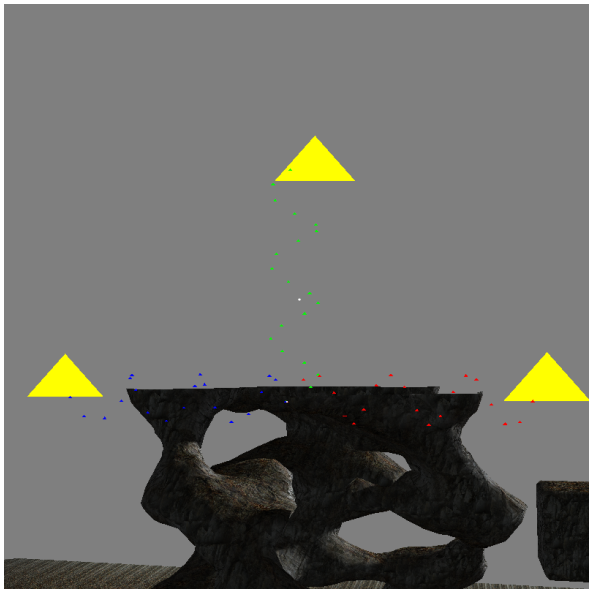


Figure 2: No PSD vs. PSD

Particle System on GPU

The particle system consists of vertices with data used to represent single particles. Because the particles don’t interact with each other, parallelization is easy. Updating values is realized in a dedicated renderpass, where the output values are written in a double buffered geometry shader output stream. A second renderpass is needed to visualize the particles. Visualization and behaviour of a particle is depends on its pre defined type. The particle system is spawned using a raycast on the KD-Tree containing all triangles in the scene. The so called emitter particle is set on the cpu and fed into the updating renderpass, spawning all other particles on gpu.



Softshadows

This application uses Variance Shadow Mapping to softing all shadow maps. First the shadow map is generated on its dedicated renderpass. The resulting texture is bound to the shaders that need it.
Shadows are then calculated as usual in the pixel shader, but its edges are blurred and smoothed using the Chebyshev formula. (Mind the light bleeding!)

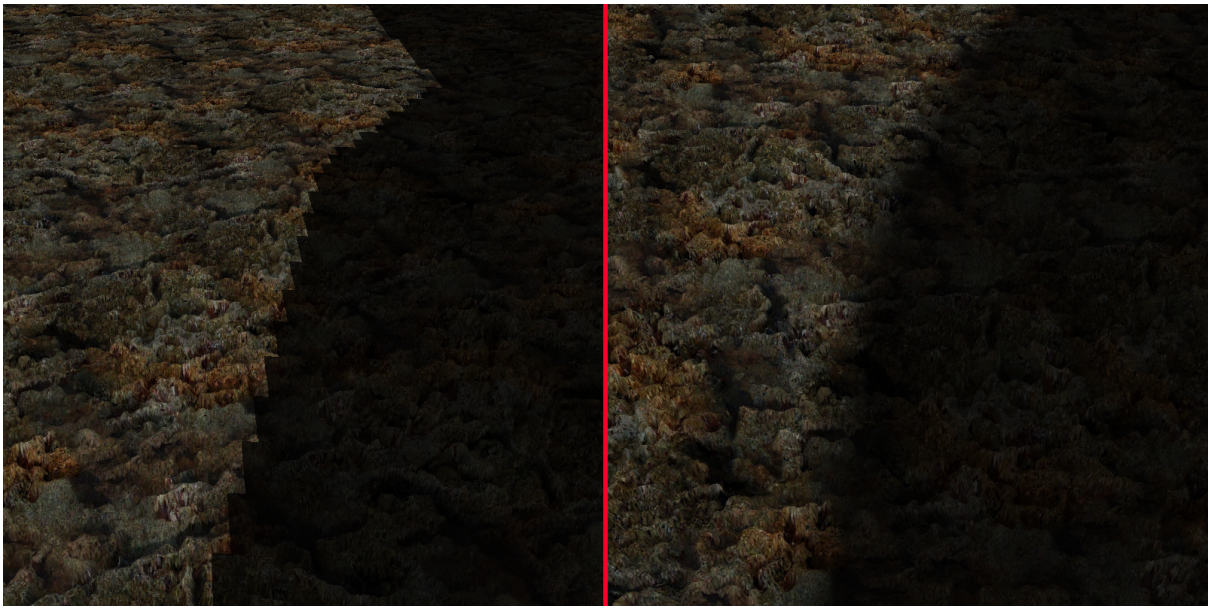


Figure 4: Hard vs Soft Shadows

Tessellation

Currently all surfaces are tessellated (flat) by running through the Hull -> Tessellator -> Domain Shader. The tessellation factors are fixed. Of course projection of the vertices has to be applied after tessellation.