Triplanar Displacement Mapping for Terrain Rendering

S. Weiss and F. Bayer and R. Westermann

Technical University of Munich, Germany

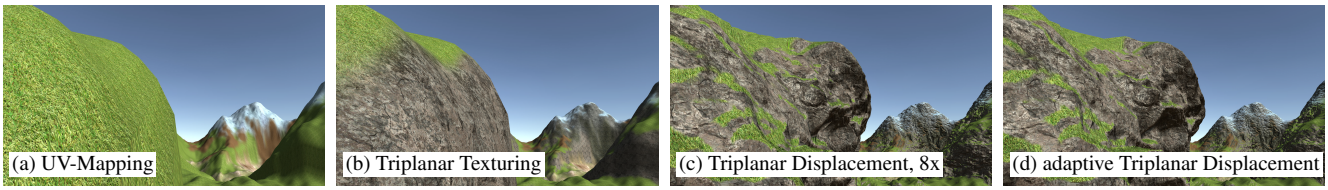

Figure 1: *Severe stretching can occur when a 2D texture is projected onto a terrain field using uv-mapping (a), which can be avoided via triplanar texture mapping (b).We propose* triplanar displacement mapping *(c,d), a combination of triplanar texture mapping with tessellation, to add geometric details to a heightfield. We extend this method via an adaptive, displacement-aware tessellation scheme (d) that achieves the same visual quality without requiring a high-resolution base mesh (c).*

Abstract

Heightmap-based terrain representations are common in computer games and simulations. However, adding geometric details to such a representation during rendering on the GPU is difficult to achieve. In this paper, we propose a combination of triplanar mapping, displacement mapping, and tessellation on the GPU, to create extruded geometry along steep faces of heightmap-based terrain fields on-the-fky during rendering. The method allows rendering geometric details such as overhangs and boulders, without explicit triangulation. We further demonstrate how to handle collisions and shadows for the enriched geometry.

1. Introduction

In computer graphics, terrains are often represented by heightmaps [GGP*19]. A heightmap is a two-dimensional scalar field, usually given at the vertices of a regular grid, where every scalar value indicates a height over some base elevation. Creating a polygonal terrain representation that can be rendered on the GPU is then performed by triangulating every cell comprised of 4 vertices into two triangles, and displacing vertices along the vertical axis. A heightmap representation, however, has limitations, since it cannot represent geometric structures like overhangs and requires high resolution to faithfully represent fine geometric details.

In scenarios where overhangs are crucial to the look of the terrain, a possible solution is to model the terrain as a surface in 3D using a high-resolution polygon model [LO10, PGGM09, BKRE19]. Alternatively, the terrain can be encoded as an implicit surface in a 3D scalar field that is represented by a voxel model, and voxel-based ray-casting is then used to render the surface [GM01, Gei07, WQK99, LO10]. Both approaches can render overhangs and caves, but increase bandwidth, memory requirements and rendering load.

We propose Triplanar Displacement Mapping (TDM), a combination of triplanar texture mapping and displacement mapping with adaptive tessellation. This method works in tandem with existing methods for rendering heightmap-based terrain representations, by adding fine geometric details to the overall shape of the terrain. In this paper we present methods to

- adaptively displace vertices using triplanar-sampled displacement maps to create overhangs and holes,
- compute correct normal vectors on the displaced and normal-mapped terrain,
- optimize the texture access patterns, and to
- include triplanar displacement mapping in existing game engines with collisions and multiple lights and shadows.

We demonstrate that our method can be applied to both heightmap- and voxel-based terrain representations,and can be easily included into existing game engines. The source code of our Unity implementation is published at `https://github.com/flojo33/Triplanar-Displacement-Mapping`.

2. Related Work

Many previous works in terrain rendering have focused on the handling of very large terrain fields with continuous level-of-detail and out-of-core rendering [DWS*97, LH04, LRC*03, Str09, SPC18, DKW10, DKW09]. For a thorough overview, let us refer to the

DOI: 10.2312/egs.20201016

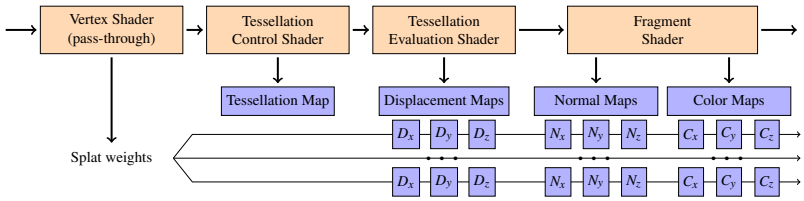

Figure 2: *Overview of our Triplanar Displacement Mapping pipeline. We apply triplanar mapping with splat maps in every stage to sample displacements, detailed normals and colors independently.*

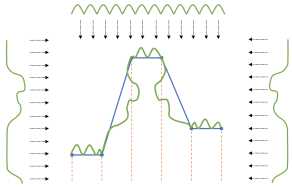

Figure 3: *Triplanar displacement mapping applied to a heightmap-based terrain. The green lines represent displacement maps projected onto the blue base heightmapped terrain.*

recent report by Galin et al. [GGP*19]. Closely related to our approach is the work by Gamito and Musgrave [GM01], which can render overhangs by ray-tracing a heightmap that is warped by a flow field. To add fine-scale geometric detail to arbitrary polygon models, displacement mapping is a common approach for uv-mapped objects [WWT*03, Don05, SKU08, ZR19]. To the best of our knowledge, however, we have not seen it being applied with triplanar mapping.

3. Method

Our proposed method adds more details to the vertical faces of cliffs based on displacement maps. It first utilizes GPU tessellation [NKF*15] to create more vertices along faces of the terrain that should be extruded or indented. Then, it projects displacement maps onto the terrain using triplanar mapping (Sec. 3.1). These maps can displace the newly generated geometry to create more realistic details like bumpy rocks and small overhangs. Next, normal maps are sampled to compensate for the displaced surface. Color maps are then sampled based on the displaced positions and sampled normals. The terrain is adaptively tessellated (Sec. 3.2), and a number of optimizations are performed to minimize texture access operations(Sec. 3.3). To support multiple terrain types, we use texture splatting [CM93]. In texture splatting, a four-channel splat map (stored per-vertex) specifies interpolation weights between four different materials. The whole pipeline is depicted in Fig. 2.

3.1. Triplanar Displacement Mapping

In traditional uv-mapping, textures are mapped to a surface mesh using texture coordinates that are stored for each vertex of the mesh. Since along steep cliffs, texture coordinates are close in texture space but represent details at large spatial distances, this often leads to severe stretching. To compensate this effect, triplanar texture mapping [Nic08, p.16], [Gol17] is often used. The idea is to use three textures using different parametrizations XY, XZ and YZ and blend them based on the normal vector. Let $\mathbf{x} = (x,y,z)^T$ be the position of the current fragment with normal $\mathbf{n}$, T_x, T_y, T_z are respectively the three textures, b_x, b_y, b_z are the corresponding blend factors. Then, the final color is computed as

$$\mathbf{c} = b_x(\mathbf{n})T_x(y,z) + b_y(\mathbf{n})T_y(x,z) + b_z(\mathbf{n})T_z(x,y). \quad (1)$$

We extend this principle of triplanar texture mapping to displace geometric details along a base terrain. In traditional displacement mapping on uv-mapped meshes, the vertex $\mathbf{x}$ is moved along its normal $\mathbf{n}$ by a value sampled from a displacement texture D, at texture coordinate $\mathbf{u}$ [SKU08]:

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha D(\mathbf{u})\mathbf{n}, \quad (2)$$

with an optional scaling factor α for the strength of the displacement.

Our main idea is to obtain the displacement value D from triplanar mapping (see Eq. (1)) instead of a single texture map. Therefore, the terrain is first tessellated adaptively (Sec. 3.2), to generate additional geometry that can be displaced. Then, the displacement is performed in the tessellation evaluation shader using the triplanar textures (Fig. 3):

$$x \leftarrow x + \alpha\left(b_x(\mathbf{n})D_x(y,z) + b_y(\mathbf{n})D_y(x,z) + b_z(\mathbf{n})D_z(x,y)\right)\mathbf{n}. \quad (3)$$

After displacing the vertices, the vertex normal does not match the geometry anymore. This can be corrected by recomputing the normals using per-fragment derivatives as presented by Mikkelsen [Mik10], which, however, leads to rather flat-looking surfaces. We avoid this by computing custom normal maps that compensate for the displacement maps using a Sobel filter [SF68].

$$\mathbf{n} = \text{normalize}\left(\left(\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix} * D, \frac{1}{\alpha}, \begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix} * D\right)^T\right) \quad (4)$$

The normals derived from these textures are then projected onto the surface using reoriented normal mapping [Gol17], and then blended using the blend weights as well. The normals are then used for the triplanar texture mapping of the color textures, see Fig. 4.

An additional normal map can be blended onto the surface to add details at a finer resolution within a certain terrain area. This is achieved by applying reoriented normal mapping again using the normal mapping result as described before as the base surface. This adds details during the shading process but does not influence the triplanar blending weights.

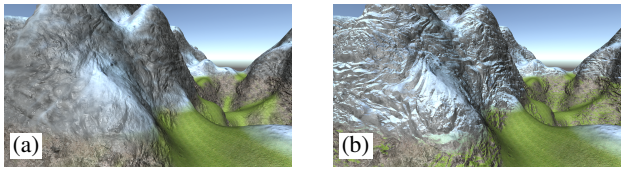

Figure 4: *Difference if detailed normal maps and color blending after normal mapping are disabled (a) or enabled (b).*

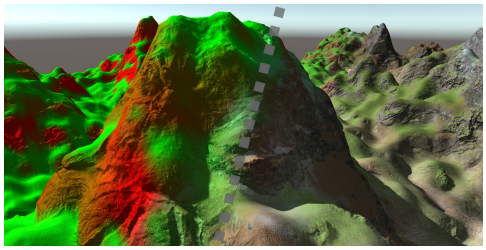

Figure 5: *The tessellation map defines regions with lower and higher roughness and correspondingly low and high tesselation, e.g., green and red determine grass and stone, respectively.*

3.2. Adaptive Level-of-Detail

Since we utilize tessellation to generate the heigh-resolution displacement geometry, the amount of tessellation is specified on a per-triangle level. For example, Cantlay [Can11] demonstrates adjustment of tessellation factors according to the camera distance, so that closer triangles are subdivided more. We extend this method by adding a factor that limits the amount of tessellation based on the expected displacement. We note that different terrain types require different strengths of displacement, and hence require different amounts of tessellation. To account for this, a maximal tessellation factor per terrain type is fist selected manually. Then, a tessellation map—stored as per-vertex attributes—is computed in a preprocess, by blending the per-material tessellation factors using triplanar mapping and texture splatting. Fig. 5 demonstrates the use of the tessellation map to keep the number of triangles low in areas with less details such as grass, and increase the tessellation factor in rough areas like boulders.

3.3. Optimized Texture Access

We support 4 different terrain types via texture splatting. Hence, our approach requires 12 textures: three for the sides of the triplanar mapping for each of the 4 different terrain types. Each of the 12 textures stores 11 attributes that are packed into three rgba-textures:

- Surface map (tessellation): surface normal X and Y, displacement map, alpha is unused
- Color map (fragment): albedo rgb, emission intensity
- Detail map (fragment): smoothness, metallic, detail normal X, Y

In total, this requires to sample up to 36 textures per fragment. This number, however, can be drastically reduced by a) reducing the area where different terrain types overlap in the splat map, b) sampling splat textures only if the weight is greater than some offset, c) sampling triplanar textures only if the blend factor is greater than some offset [Gol14], see Fig. 6. These optimizations make the transitions between terrain types appear sharper, which is often desired (Fig. 6b). However, care must be taken as to not introduce sharp discontinuities due to high offsets (Fig. 6c).

3.4. Collision and Shadows

To support collisions with the displaced terrain, the steps of the tessellation engine and the vertex displacements are replicated on the CPU, and a collision mesh that matches the rendered terrain is generated around the player. As the player moves over the terrain, the collision mesh is updated around the player asynchronously. We refer to the accompanying video for a demonstration.

Shadows using traditional shadow mapping can be used in the pipeline without major changes. Let us only note that it is crucial that during rendering the shadow map from the light position, the same tessellation as if the terrain was seen from the player camera has to be used to avoid false shadowing.

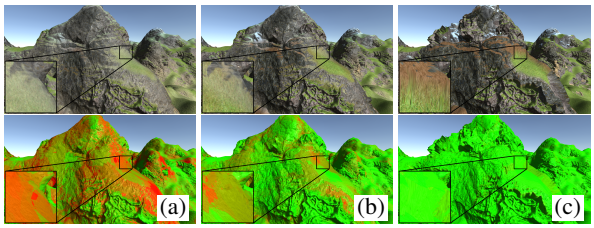

Figure 6: *Top: Influence of blend factor offset on image quality. Bottom: Number of texture samples (low (green) to high (red)), for low (a), medium (b) and high (c) optimization.*

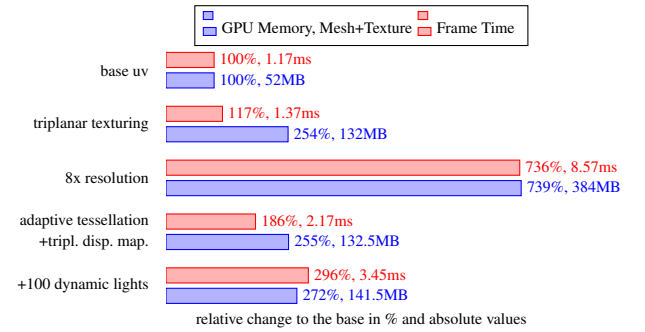

Figure 7: *Benchmarking results for a simple terrain using uv-mapping or triplanar texturing in comparison to a terrain using triplanar displacement mapping on a brute force 8x resolution grid and a dynamically tessellated terrain of also up to more than 8x resolution.*

4. Results and Benchmarks

Our system was tested on a workstation with an i7-9700k CPU, an RTX 2060 GPU and 16GB of RAM. To demonstrate the performance of triplanar displacement mapping with adaptive tessellation in comparison to a simple uv-mapped terrain, a camera was set up to fly over a Perlin Noise-based [EMP*03, p. 69] infinite terrain for a fixed number of frames (250.000) using different settings. The average frame time and memory was recorded for different shader variants and terrain resolutions (Fig. 7). As one can see, our method introduces only a slight overhead in both render time and required memory, but offers much better visual quality (see Fig. 1 for a visual comparison). To achieve the same quality using heightmaps without tessellation and displacement mapping, a grid of 8x the resolution needed and the memory requirements increase accordingly from approx. 52MB to 384MB. Our method also fits well into a deferred lighting pipeline, i.e., up to 100 dynamic lights can be used at a frame time of only 3.45ms. We refer to the accompanying video for an extended visual comparison and examples.

Furthermore, our method can also be applied to different terrain meshes as indicated in Fig. 8. Here the mesh was created using the marching cubes algorithm to render a surface based on 3D noise functions [Gei07]. This demonstrates that our proposed method can be applied to arbitrary geometry where an explicit uv-mapping is still unfeasible.

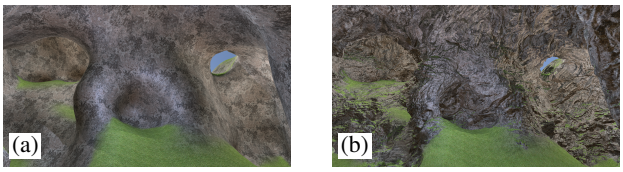

Figure 8: *Cave inside a voxel terrain without (a) and with (b) Triplanar Displacement Mapping.*

5. Conclusion

We have presented a method to add fine geometric details to triplanar textured terrain using displacement mapping. It allows to create additional geometric details like overhangs that greatly enhance the degree of realism. We have demonstrated that our approach works in tandem with existing methods to render heightmap- or voxel-based terrain, and can easily be included in existing game engines and supports deferred lighting and collision tests. Our approach requires only slightly more memory than traditional triplanar mapped terrain with splat maps. With the presented optimization to reduce the number of texture fetches, we achieve interactive frame rates even with many dynamic light sources. It vastly outperforms approaches using high-resolution 3D polygon models of the same visual quality regarding both frame rates and memory. In the future, we would like to investigate how this method can be included in an RTX raytracing framework to support reflections and global illumination in real-time.

References

[BKRE19] BECHER M., KRONE M., REINA G., ERTL T.: Feature-based volumetric terrain generation and decoration. *IEEE Transactions on Visualization and Computer Graphics 25*, 2 (2 2019), 1283–1296. 1

[Can11] CANTLAY I.: Directx 11 terrain tessellation. *Nvidia whitepaper 8*, 11 (2011), 3. 3

[CM93] CRAWFIS R. A., MAX N.: Texture splats for 3d scalar and vector field visualization. In *Proceedings of the 4th conference on Visualization'93* (1993), IEEE Computer Society, pp. 261–266. 2

[DKW09] DICK C., KRÜGER J. H., WESTERMANN R.: Gpu ray-casting for scalable terrain rendering. In *Eurographics (Areas Papers)* (2009), Citeseer, pp. 43–50. 1

[DKW10] DICK C., KRÜGER J., WESTERMANN R.: Gpu-aware hybrid terrain rendering. *Proceedings of IADIS computer graphics, visualization, computer vision and image processing 10* (2010), 3–10. 1

[Don05] DONNELLY W.: Per-pixel displacement mapping with distance functions. *GPU gems 2*, 22 (2005), 3. 2

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: Roaming terrain: real-time optimally adapting meshes. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)* (1997), IEEE, pp. 81–88. 1

[EMP*03] EBERT D., MUSGRAVE F., PEACHEY D., PERLIN K., WORLEY S., MARK W., HART J.: *Texturing and Modeling: A Procedural Approach: Third Edition*. Elsevier Inc., United States, 2003. 3

[Gei07] GEISS R.: Generating complex procedural terrains using the gpu. *GPU gems 3* (2007), 7–37. 1, 4

[GGP*19] GALIN E., GUÃL'RIN E., PEYTAVIE A., CORDONNIER G., CANI M.-P., BENES B., GAIN J.: A review of digital terrain modeling. *Computer Graphics Forum 38*, 2 (2019), 553–577. doi:10.1111/cgf.13657. 1, 2

[GM01] GAMITO M. N., MUSGRAVE F. K.: Procedural landscapes with overhangs. In *10th Portuguese Computer Graphics Meeting* (2001), vol. 2, Citeseer. 1, 2

[Gol14] GOLLENT M.: Triplanar mapping texturing arbitrary surfaces, 2014. Accessed 02/21/2020. URL: https://gdcvault.com/play/1020394/Landscape-Creation-and-Rendering-in. 3

[Gol17] GOLUS B.: Normal mapping for a triplanar shader, 9 2017. Accessed 02/21/2020. URL: https://medium.com/@bgolus/normal-mapping-for-a-triplanar-shader-10bf39dca05a. 2

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. In *ACM Transactions on Graphics (TOG)* (2004), vol. 23, ACM, pp. 769–776. 1

[LO10] LENGYEL E. S., OWENS J. D.: *Voxel-based terrain for real-time virtual simulations*. University of California, Davis, 2010. 1

[LRC*03] LUEBKE D., REDDY M., COHEN J. D., VARSHNEY A., WATSON B., HUEBNER R.: *Level of detail for 3D graphics*. Morgan Kaufmann, 2003. 1

[Mik10] MIKKELSEN M. S.: Bump mapping unparametrized surfaces on the gpu. *Journal of Graphics, GPU, and Game Tools 15*, 1 (2010), 49–61. 2

[Nic08] NICHOLSON K.: *GPU Based Algorithms for Terrain Texturing*. Master's thesis, University of Canterbury, Christchurch, New Zealand, 2008. 2

[NKF*15] NIESSNER M., KEINERT B., FISHER M., STAMMINGER M., LOOP C., SCHÄFER H.: Real-time rendering techniques with hardware tessellation. *Computer Graphics Forum 35* (9 2015). 2

[PGGM09] PEYTAVIE A., GALIN E., GROSJEAN J., MÉRILLOU S.: Arches: a framework for modeling complex terrains. *Computer Graphics Forum 28* (4 2009), 457 – 467. 1

[SF68] SOBEL I., FELDMAN G.: A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in* (1968), 271–272. 2

[SKU08] SZIRMAY-KALOS L., UMENHOFFER T.: Displacement mapping on the gpu-state of the art. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 1567–1592. 2

[SPC18] SILVESTRE A., PEREIRA J., COSTA V.: A real-time terrain ray-tracing engine. In *2018 International Conference on Graphics and Interaction (ICGI)* (11 2018), pp. 1–8. doi:10.1109/ITCGI.2018.8602735. 1

[Str09] STRUGAR F.: Continuous distance-dependent level of detail for rendering heightmaps. *Journal of graphics, GPU, and game tools 14*, 4 (2009), 57–74. 1

[WQK99] WAN M., QU H., KAUFMAN A.: Virtual flythrough over a voxel-based terrain. In *Proceedings IEEE Virtual Reality (Cat. No. 99CB36316)* (1999), IEEE, pp. 53–60. 1

[WWT*03] WANG L., WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.-Y., SHUM H.-Y., SHUM H.-Y.: View-dependent displacement mapping. In *ACM Transactions on graphics (TOG)* (2003), vol. 22, ACM, pp. 334–339. 2

[ZR19] ZIRR T., RITSCHEL T.: Distortion-free displacement mapping. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 53–62. 2